IMS/ESA

Application Programming:Transaction Manager

Version 6

SC26-8729-05

IMS/ESA

Application Programming:Transaction Manager

Version 6

SC26-8729-05

Note

Before using this information and the product it supports, be sure to read the general information under “Notices” on page xiii.

Sixth Edition (July 2000) (Softcopy Only)

This edition replaces and makes obsolete the previous edition, SC26–8729–04. This edition is available in softcopy format only. The technical changes for this edition are summarized under “Summary of Changes” on page xix and are indicated by a vertical bar to the left of a change.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address below.

A form for readers’ comments is provided at the back of this publication. If the form has been removed, address your comments to:

IBM Corporation, BWE/H3

P.O. Box 49023

San Jose, CA, 95161-9023

U.S.A.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 1974, 2000. All rights reserved.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices xiii
Programming Interface Information xiii
Trademarks xiv
Product Names. xiv

Preface xv
Summary of Contents xv
Prerequisite Knowledge xv
Terminology xvi
Change Indicators xvi
Syntax Diagrams xvi

Summary of Changes xix
Changes to the Current Edition of the Book for Version 6 xix
Changes to This Book for Version 6 xix
Library Changes for Version 6 xix

Part 1. Writing Application Programs 1

Chapter 1. How Application Programs Work with the IMS Transaction Manager 7
Application Program Environments 7
The Application Programming Interface 7
Your Application in the System 8
Using LU 6.2 Devices 9
How IMS TM Schedules Application Programs 10
Getting Started with DL/I 10
Relationship of AIB and PCB with Language Interfaces 11
Language Unique Interfaces 12
Language Independent Interfaces 12
Using DL/I Calls 12
Message Call Functions 12
System Service Call Functions 13
Status Codes, Return Codes, and Reason Codes 13
Exceptional Conditions 14
Error Routines 14
How Your Program Processes Messages 14
Message Types. 14
What Happens When a Message is Processed 17
Results of a Message: I/O PCB 19
How IMS TM Edits Messages 19
Printing Output Messages 20
Using Basic Edit 20
Using Intersystem Communication Edit 21
Using Message Format Service 21
Using LU 6.2 User Edit Exit (Optional) 28
DB2 Considerations 28

Chapter 2. Defining Application Program Elements 31
Formatting DL/I Calls for Language Interfaces 31
Application Programming for Assembler Language 32
Format 32
Parameters 33

© Copyright IBM Corp. 1974, 2000

Example DL/I Call Formats . . . 34
Application Programming for C Language . . . 34
Format . . . 34
Parameters . . . 35
I/O Area . . . 36
Example DL/I Call Formats . . . 37
Application Programming for COBOL . . . 37
Format . . . 37
Parameters . . . 38
Example DL/I Call Formats . . . 39
Application Programming for Pascal . . . 39
Format . . . 39
Parameters . . . 41
Example DL/I Call Formats . . . 42
Application Programming for PL/I . . . 42
Format . . . 42
Parameters . . . 43
Example DL/I Call Formats . . . 44
Relationship of Calls to PCB Types . . . 44
Specifying the I/O PCB Mask . . . 45
Specifying the Alternate PCB Mask . . . 48
Specifying the AIB Mask . . . 49
Specifying the I/O Areas . . . 51
Using the AIBTDLI Interface . . . 51
Overview . . . 51
Defining Storage for the AIB . . . 52
Specifying the Language-Specific Entry Point . . . 52
Assembler Language . . . 52
C Language . . . 52
COBOL . . . 53
Pascal . . . 53
PL/I . . . 53
Interface Considerations . . . 54
PCB Lists . . . 54
Format of a PCB List . . . 54
Format of a GPSB PCB List . . . 54
PCB Summary . . . 55
Using Language Environment . . . 55
The CEETDLI interface to IMS . . . 56
LANG= Option on PSBGEN for PL/I Compatibility with Language Environment . . . 56
Special DL/I Situations . . . 56
Mixed-Language Programming . . . 57
Using Language Environment Routine Retention . . . 57
Using the Extended Addressing Capabilities of MVS/ESA . . . 57
Preloaded Programs . . . 57
DCCTL . . . 57

Chapter 3. Writing DL/I Calls for Transaction Management . . . 59
AUTH Call . . . 59
Format . . . 59
Parameters . . . 60
I/O Area . . . 60
Usage . . . 63
Restrictions . . . 64
CHNG Call . . . 64

Format 64
Parameters 65
Usage 66
Error Codes 71
Restrictions 72
CMD Call 72
Format 72
Parameters 72
Usage 73
Restrictions 73
GCMD Call 74
Format 74
Parameters 74
Usage 74
Restrictions 75
GN Call 75
Format 75
Parameters 75
Usage 76
Restrictions 76
GU Call 76
Format 76
Parameters 76
Usage 77
Restrictions 78
ISRT Call 78
Format 78
Parameters 78
Usage 79
Restrictions 80
PURG Call 80
Parameters 80
Usage 81
Restrictions 82
SETO Call 82
Format 82
Parameters 82
Usage 84
Error Codes 86
Restrictions 87

Chapter 4. Writing DL/I Calls for System Services 89
APSB Call 90
Format 90
Parameters 90
Usage 90
Restrictions 91
CHKP (Basic) Call 91
Format 91
Parameters 91
Usage 92
Restrictions 92
CHKP (Symbolic) Call 92
Format 92
Parameters 92
Usage 93

Restrictions 93
DPSB Call 93
Format 93
Parameters 94
Usage 94
Restrictions 94
GMSG Call 94
Format 94
Parameters 94
Usage 95
Restrictions 96
GSCD Call 96
Format 96
Parameters 96
Usage 97
Restrictions 97
ICMD Call. 97
Format 97
Parameters 97
Usage 99
Restrictions 99
INIT Call 99
Format 100
Parameters. 100
Usage. 100
INQY Call 102
Format 102
Parameters. 102
Usage. 103
Restrictions. 109
LOG Call 109
Format 109
Parameters. 109
Usage. 111
Restrictions. 111
RCMD Call 111
Format 111
Parameters. 111
Usage. 112
Restrictions. 112
ROLB Call 112
Format 112
Parameters. 112
Usage. 113
Restrictions. 113
ROLL Call 114
Format 114
Parameters. 114
Usage. 114
Restrictions. 114
ROLS Call 114
Format 115
Parameters. 115
Usage. 115
Restrictions. 116
SETS/SETU Call. 116

Format 116
Parameters 116
Usage 117
Restrictions 117
SYNC Call 118
Format 118
Parameters 118
Usage 118
Restrictions 118
XRST Call 118
Format 119
Parameters 119
Usage 120
Restrictions 121

Chapter 5. More about Message Processing 123
Sending Messages to Other Terminals and Programs 123
Sending Messages to Other Terminals 124
Sending Messages to Other Application Programs 126
How the VTAM I/O Facility Affects Your VTAM Terminal 127
Communicating with Other IMS TM Systems Using MSC 128
Implications of MSC for Program Coding 128
Receiving Messages from Other IMS TM Systems 129
Sending Messages to Alternate Destinations in Other IMS TM Systems 130
IMS Conversations 131
A Conversational Example 131
Conversational Structure 132
Replying to the Terminal 136
Using ROLB, ROLL, and ROLS in Conversations 137
Passing the Conversation to another Conversational Program 137
Message Switching in APPC Conversations 139
Processing Conversations with APPC 141
Standard IMS Application Programs 142
Modified IMS Application Programs 142
CPI-C Driven Application Programs 143
Processing Conversations with OTMA 145
Backing out to a Prior Commit Point: ROLL, ROLB, and ROLS Calls 145
Using ROLL 146
Using ROLB 146
Using ROLS 148
Backing out to an Intermediate Backout Point: SETS/SETU and ROLS 149
Using SETS/SETU 149
Using ROLS 150
Writing a Message-Driven Program 151
Coding DC Calls and Data Areas 152
Your Input 152
Skeleton MPP 152
Coding Your Program in Assembler Language 153
Coding Your Program in C Language 153
Coding Your Program in COBOL 155
Coding Your Program in Pascal 157
Coding Your Program in PL/I 160

Part 2. Message Format Service 163

Chapter 6. Introduction to MFS 169

Advantages of Using MFS 169
Simplify Development and Maintenance 169
Improve Online Performance of a Terminal 170
MFS Control Blocks 171
MFS Examples 171
Relationship Between MFS Control Blocks and Screen Format 175
Overview of MFS Components and Operation 177
MFS Language Utility 178
MFS Service Utility 179
MFS Device Characteristics Table Utility 179
MFS Message Editor 179
MFS Pool Manager 179
MFSTEST Pool Manager 180
Devices and Logical Units That Operate with MFS 180
Using Distributed Presentation Management (DPM) 182

Chapter 7. Message Formatting Functions 183
Input Message Formatting 183
How MFS Is Selected 183
How MFS Formats Input Messages 186
General Rules for Multiple DPAGE Input 200
3270 and SLU 2 Input Substitution Character 201
Input Format Control for ISC (DPM-Bn) Subsystems 201
Input Message Formatting 201
Input Modes 202
Paging Requests 203
Output Message Formatting 203
How MFS Is Selected 203
How MFS Formats Output Messages 204
Output Format Control for ISC (DPM-Bn) Subsystems 228
Format Control 228
Function Management (FM) Headers 228
Paged Output Messages 228
Output Modes 229
Variable-Length Output Data Stream 230
FILL=NULL Specification 231
Trailing Blank Compression 232
Data Structure Name 235
Version Identification 235
Your Control of MFS 235
Operator Logical Paging 236
Operator Control Tables 237
3270 or SLU 2-Only Feature Definitions 237
Paging Action at the Device 238
Unprotected Screen Option 242
The 3290 in Partitioned Format Mode 243
The 3180 in Partitioned Format Mode 245
MFS Format Sets Supplied by IMS 246
System Message Format 246
Multisegment System Message Format 246
Output Message Default Format 247
Block Error Message Format 247
/DISPLAY Command Format 247
Multisegment Format 247
MFS 3270 or SLU 2 Master Terminal Format 247
MFS Sign-On Device Formats 247

MFS Formatting for the 3270 or SLU 2 Master Terminal 247
MFS Device Characteristics Table 249
Version Identification Function for DPM Formats 250

Chapter 8. MFS Application Program Design 251
Relationships Between MFS Control Blocks 251
Device Considerations Relative to Control Block Linkages 256
Format Library Member Selection 258
3270 or SLU 2 Screen Formatting 261
3290 Screen Formatting 263
3180 Screen Formatting 265
Performance Factors 265
All MFS-Supported Devices 265
3270 or SLU 2 Display Devices 266
3270 or SLU 2 Devices with Large Screens 267
SLU P and ISC Subsystems with DPM 267
Loading Programmed Symbol Buffers 268

Chapter 9. Application Programming Using MFS 273
Input Message Formats 273
Logical Pages 273
Device-Dependent Input Information (3270 or SLU 2) 273
Output Message Formats 275
Logical Pages 275
Segment Format 276
Field Format (Options 1 and 2) 277
Field Format (Option 3) 278
Device-Dependent Output Information 278
Dynamic Attribute Modification 281
Dynamic Modification of Extended Field Attributes 283
Dynamic Modification of EGCS Data 289
Dynamic Modification of DBCS/EBCDIC Mixed Data 290
Specification of Message Output Descriptor Name 291
MFS Bypass for the 3270 or SLU 2 292

Chapter 10. MFS Language Utility 311
Utility Control Statements 311
Control Statement Syntax 311
Summary of Control Statements 314
EXEC Statement Parameters 316
Message Definition Statements 317
Format Definition Statements 330
Partition Set Definition Statements 386
Table Definition Statements 389
Compilation Statements 391

Part 3. IMS Adapter for REXX 397

Chapter 11. IMS Adapter for REXX 399
Addressing Other Environments 400
REXX Transaction Programs 400
IMS Adapter for REXX Overview Diagram 402
IVPREXX Sample Application 403
REXXTDLI Commands 404
Addressable Environments 405
REXXTDLI Calls 405

Return Codes . . . 405
Parameter Handling . . . 406
Example DL/I Calls . . . 407
REXXIMS Extended Commands . . . 408
DLIINFO . . . 409
IMSRXTRC . . . 410
MAPDEF . . . 410
MAPGET . . . 413
MAPPUT . . . 413
SET . . . 414
SRRBACK and SRRCMIT . . . 415
STORAGE . . . 416
WTO, WTP, and WTL . . . 417
WTOR . . . 418
IMSQUERY Extended Functions . . . 418

Chapter 12. IMS Adapter for REXX Exit Routine . . . 421
Environment . . . 421
Parameters . . . 421

Chapter 13. Sample Execs Using REXXTDLI . . . 425
SAY Exec: For Expression Evaluation . . . 425
PCBINFO Exec: Display PCBs Available in Current PSB . . . 426
PART Execs: Database Access Example . . . 429
PARTNUM Exec: Show Set of Parts Near a Specified Number . . . 430
PARTNAME Exec: Show a Set of Parts with a Similar Name . . . 431
DFSSAM01 Exec: Load the Parts Database . . . 431
DOCMD: IMS Commands Front End . . . 432
IVPREXX: MPP/IFP Front End for General Exec Execution . . . 436

Part 4. For Your Reference . . . 437

Chapter 14. Summary of TM Message and System Service Calls . . . 439
Transaction Management Call Summary . . . 439
System Service Call Summary . . . 440

Chapter 15. DL/I Status Codes . . . 443
Status Code Tables . . . 443
Categories of DL/I Status Codes . . . 443
Status Code Explanations . . . 452

Chapter 16. DL/I Return and Reason Codes . . . 475
Return and Reason Code Tables . . . 475
DL/I Return and Reason Code Explanations . . . 491

Part 5. Appendixes . . . 501

Appendix A. Sample Applications . . . 503

Appendix B. MFS Definitions for Intersystem Communication . . . 505

Appendix C. Device Compatibility with Previous Versions of MFS . . . 507
Using STACK/UNSTACK to Convert MFS Device Formats to Symbolic Name Formats . . . 508
3270 Device Format Conversion Example . . . 509

3270 Printer and SLU 1 Compatibility 511
SLU P Compatibility 512
IBM 3278-52/3283-52 and IBM 5550 Family (as 3270) Compatibility 512
Existing 3270 and IBM 5550 Family (as 3270) Compatibility 512

Appendix D. Spool API 515
Understanding Parsing Errors 515
Keywords 515
Status Codes 515
Error Codes 515
Diagnosis Examples 516
Understanding Allocation Errors 519
Understanding Dynamic Output for Print Data Sets 519
CHNG Call with PRTO Option 519
CHNG Call with TXTU Option 520
CHNG Call with OUTN Option 520
Sample Program Using the Spool API 520
Application PCB Structure 520
GU Call to I/O PCB 521
CHNG Call to Alternate PCB 521
ISRT Call to Alternate PCB 522
Program Termination 523

Appendix E. Using the DL/I Test Program (DFSDDLT0) 525
Control Statements 525
Planning the Control Statement Order 527
ABEND Statement 528
Examples of ABEND Statement 528
CALL Statement 528
CALL FUNCTION Statement 528
CALL DATA Statement 531
OPTION DATA Statement 533
FEEDBACK DATA Statement 534
Call Functions 534
Examples of DL/I Call Functions 537
CALL FUNCTION Statement with Column-Specific SSAs 546
DFSDDLT0 Call Functions 547
Examples of DFSDDLT0 Call Functions 548
COMMENT Statement 549
Conditional COMMENT Statement 549
Unconditional COMMENT Statement 549
Example of COMMENT Statement 549
COMPARE Statement 549
COMPARE DATA Statement 550
COMPARE AIB Statement 551
COMPARE PCB Statement 552
Examples of COMPARE DATA and PCB Statements 554
IGNORE Statement 556
Example of IGNORE (N or .) 556
OPTION Statement 556
Example of OPTION Control Statement 557
PUNCH Statement 557
Example of PUNCH CTL Statement 559
Example of PUNCH CTL Statement for All Parameters 559
STATUS Statement 559
Examples of STATUS Statement 562

WTO Statement 563
Example of WTO Statement 563
WTOR Statement 563
Example of WTOR Statement 564
JCL Requirements 564
SYSIN DD Statement 565
SYSIN2 DD Statement 565
PRINTDD DD Statement 565
PUNCHDD DD Statement 565
Using the PREINIT Parameter for DFSDDLT0 Input Restart 566
Execution of DFSDDLT0 in IMS Regions 567
Explanation of DFSDDLT0 Return Codes 568
Hints on Using DFSDDLT0 568
To Load a Database 568
To Print the Segments in a Database 569
To Retrieve and Replace a Segment 569
To Delete a Segment 570
To Do Regression Testing 570
To Use as a Debugging Aid 570
To Verify How a Call Is Executed 570

Bibliography 571
IMS/ESA Version 6 Library 571

Index 573

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Subject to IBM's valid intellectual property or other legally protectable rights, any functionally equivalent product, program, or service may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, NY 10594
U.S.A.

Licensees of this program who wish to have information about it for the purpose of enabling (1) the exchange of information between independently created programs and other programs (including this one) and (2) the mutual use of the information that has been exchanged, should contact:

IBM Corporation
555 Bailey Avenue, W92/H3
P.O. Box 49023
San Jose, CA 95161-9023

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

Programming Interface Information

This book is intended to help the application programmer write IMS application programs. This book primarily documents General-use Programming Interface and Associated Guidance Information provided by IMS.

General-use programming interfaces allow the customer to write programs that obtain the services of IMS.

However, this book also documents Product-sensitive Programming Interface and Associated Guidance Information provided by IMS.

Product-sensitive programming interfaces allow the customer installation to perform tasks such as diagnosing, modifying, monitoring, repairing, tailoring, or tuning of IMS. Use of such interfaces creates dependencies on the detailed design or implementation of the IBM software product. Product-sensitive programming interfaces should be used only for these specialized purposes. Because of their dependencies on detailed design and implementation, it is to be expected that programs written to such interfaces may need to be changed to run with new product releases or versions, or as a result of service.

© Copyright IBM Corp. 1974, 2000

Product-sensitive Programming Interface and Associated Guidance Information is identified where it occurs, either by an introductory statement to a chapter or section or by the following marking:

Product-sensitive programming interface

Product-sensitive Programming Interface and Associated Guidance Information...

End of Product-sensitive programming interface

Trademarks

The following terms, are trademarks of the IBM Corporation in the United States or other countries or both:

AD/Cycle
C/370
C/MVS
C++/MVS
CICS
COBOL/370
DATABASE 2
DB2
IBM
IMS
IMS/ESA
Language Environment
MVS
MVS/ESA
MVS/XA
RACF
SAA
Series/1
VTAM

Other company, product, and service names, which may be denoted by a double asterisk (**), may be trademarks or service marks of others.

Product Names

In this book, the following licensed programs have shortened names:

- “C/C++ for MVS/ESA” is referred to as either “C/MVS” or “C++/MVS”.
- “COBOL for MVS & VM” is referred to as “COBOL”.
- “DB2 for MVS/ESA” is referred to as “DB2”.
- “Language Environment for MVS & VM” is referred to as “Language Environment”.
- “PL/I for MVS & VM” is referred to as “PL/I”.

Preface

This book is a guide to application programming in a Data Communication (DC) environment. This book provides guidance for the tasks involved in creating and running application programs. It covers basic information on coding transaction management message calls for DC programs, and it provides information on creating REXX EXECs under Time-Sharing Option Extensions (TSO/E).

This book is designed for IMS/ESA (hereafter referred to in this book as IMS) application and system programmers who use the DC environment of the IMS Transaction Manager (TM). The combination of the IMS Transaction Manager and the IMS Database Manager is equivalent to IMS DB/DC.

This book also contains information on the Data Communications Control (DCCTL) environment. DCCTL is generated by IMS TM, contains no database components, and is designed to function as a transaction manager for non-IMS database management systems.

Summary of Contents

This publication has five parts:

- Part 1. Writing Application Programs provides basic information on coding DL/I calls for IMS TM application programs.
- Part 2. Message Format Service discusses application programming with MFS.
- Part 3. IMS Adapter for REXX discusses the IMS interface for REXX (REXXTDLI), and provides information you can use to interactively develop REXX EXECs under TSO/E and execute them in IMS MPPs, BMPs, IFPs, or batch regions.
- Part 4. For Your Reference provides additional reference information you need to write your application program.
- Part 5, Appendixes contains the following:
  - Sample exit routines
  - Sample applications
  - MFS definitions for intersystem communication
  - Device compatibility with previous versions of MFS
  - Spool API
  - Using the DL/I test program

Prerequisite Knowledge

IBM offers a wide variety of classroom and self-study courses to help you learn IMS. For a complete list, see the IMS home page on the World Wide Web at: http://www.software.ibm.com/data/ims

Before using this book, you should understand the concepts of application design presented in *IMS/ESA Application Programming: Design Guide*, which assumes you understand basic IMS concepts and the IMS/ESA environments.

This book is an extension to *IMS/ESA Application Programming: Design Guide*. The IMS concepts explained in this manual are limited to those concepts pertinent to developing and coding application programs. You should also know how to use assembler language, C language, COBOL, Pascal, or PL/I.

© Copyright IBM Corp. 1974, 2000

Terminology

In this book, the term *external subsystems* refers to subsystems that are not CCTL subsystems, unless indicated otherwise. One example of an external subsystem is DATABASE 2 (DB2).

For definitions of terminology used in this book and references to related information in other books, see *IMS/ESA Master Index and Glossary*.

Change Indicators

Technical changes are indicated in this publication by a vertical bar (|) to the left of the changed text. If a figure has changed, a vertical bar appears to the left of the figure caption.

Syntax Diagrams

The following rules apply to the syntax diagrams used in this book:

Arrow symbols

Read the syntax diagrams from left to right, from top to bottom, following the path of the line.

►►— Indicates the beginning of a statement.

—► Indicates that the statement syntax is continued on the next line.

►— Indicates that a statement is continued from the previous line.

—►◄ Indicates the end of a statement.

Diagrams of syntactical units other than complete statements start with the ►— symbol and end with the —► symbol.

Conventions

- Keywords, their allowable synonyms, and reserved parameters, appear in uppercase for MVS and OS/2 operating systems, and lowercase for UNIX operating systems. These items must be entered exactly as shown.
- Variables appear in lowercase italics (for example, *column-name*). They represent user-defined parameters or suboptions.
- When entering commands, separate parameters and keywords by at least one blank if there is no intervening punctuation.
- Enter punctuation marks (slashes, commas, periods, parentheses, quotation marks, equal signs) and numbers exactly as given.
- Footnotes are shown by a number in parentheses, for example, (1).
- A ␣ symbol indicates one blank position.

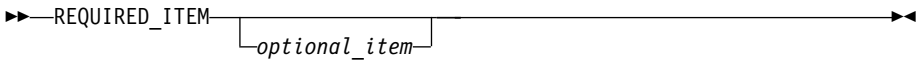
Required items

Required items appear on the horizontal line (the main path).

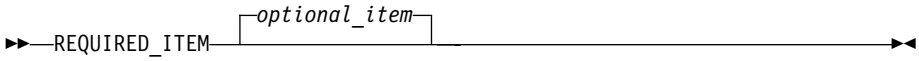
```
►►—REQUIRED_ITEM———————————————————————————————————————►◄
```

Optional Items

Optional items appear below the main path.

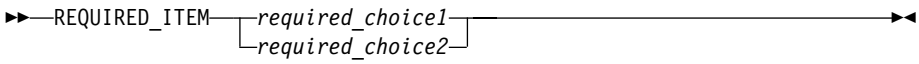


If an optional item appears above the main path, that item has no effect on the execution of the statement and is used only for readability.

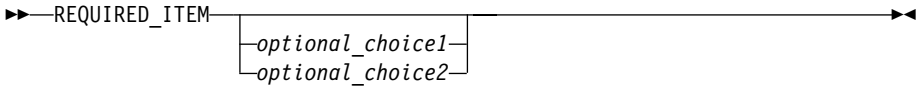


Multiple required or optional items

If you can choose from two or more items, they appear vertically in a stack. If you *must* choose one of the items, one item of the stack appears on the main path.

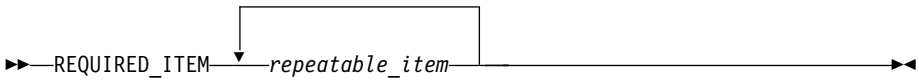


If choosing one of the items is optional, the entire stack appears below the main path.

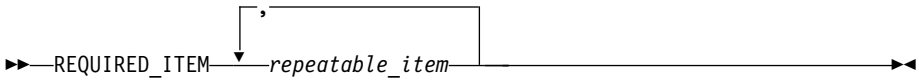


Repeatable items

An arrow returning to the left above the main line indicates that an item can be repeated.



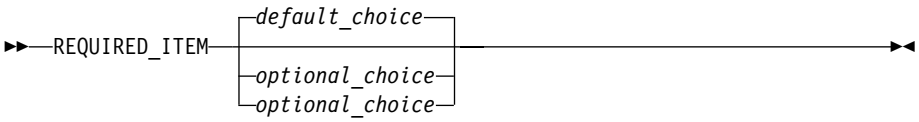
If the repeat arrow contains a comma, you must separate repeated items with a comma.



A repeat arrow above a stack indicates that you can specify more than one of the choices in the stack.

Default keywords

IBM-supplied default keywords appear above the main path, and the remaining choices are shown below the main path. In the parameter list following the syntax diagram, the default choices are underlined.

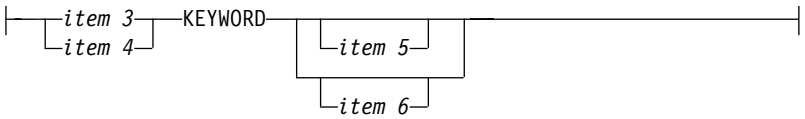


IMS-specific syntax information

Fragments

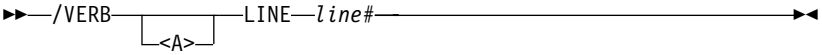
Sometimes a diagram must be split into fragments. The fragments are represented by a letter or fragment name, set off like this: | A |. The fragment follows the end of the main diagram. The following example shows the use of a fragment.

A:



Substitution-block

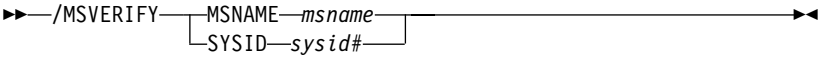
Sometimes a set of several parameters is represented by a substitution-block such as **<A>**. For example, in the imaginary /VERB command you could enter `/VERB LINE 1`, `/VERB EITHER LINE 1`, or `/VERB OR LINE 1`.



where <A> is:

Parameter endings

Parameters with number values end with the symbol '#', parameters that are names end with 'name', and parameters that can be generic end with '*'.



The MSNAME keyword in the example supports a name value and the SYSID keyword supports a number value.

Summary of Changes

Changes to the Current Edition of the Book for Version 6

This edition, which is in softcopy format only, includes technical and editorial changes.

Changes to This Book for Version 6

This book contains new and changed information about the following line items:

- Distributed Sync Point
- Shared Queues
- New Time-Stamp Format

Library Changes for Version 6

The IMS/ESA Version 6 library differs from the IMS/ESA Version 5 library in these major respects:

- *IMS/ESA Common Queue Server Guide and Reference*

  This new book describes the IMS Common Queue Server (CQS).

- *IMS/ESA DBRC Guide and Reference*

  This new book describes all the functions of IMS Database Recovery Control (DBRC).

- The IMS Application Programming summary books (*IMS/ESA Application Programming: Database Manager Summary*, *IMS/ESA Application Programming: Transaction Manager Summary*, and *IMS/ESA Application Programming: EXEC DLI Commands for CICS and IMS Summary*) are no longer included with the IMS library.
- The Softcopy Master Index is not included.
- All information about IRLM 1.5 and data sharing using IRLM 1.5 has been removed from the IMS V6 books. If you use IRLM 1.5, and want to migrate to using IRLM 2.1 and Sysplex data sharing, see *IMS/ESA Release Planning Guide*.
- The chapter that was titled ″Database Control (DBCTL) Interface″ in the *IMS/ESA Customization Guide* has been revised for Open Databse Access (ODBA) and moved to ″Appendix A, Using the Database Resource Adapter (DRA)″ in the *IMS/ESA Application Programming: Database Manager*.

© Copyright IBM Corp. 1974, 2000

Part 1. Writing Application Programs

Chapter 1. How Application Programs Work with the IMS Transaction Manager . . . 7
Application Program Environments . . . 7
The Application Programming Interface . . . 7
Your Application in the System . . . 8
The DB/DC Environment . . . 8
The DCCTL Environment . . . 8
The TM Batch Environment . . . 9
Using LU 6.2 Devices . . . 9
How IMS TM Schedules Application Programs . . . 10
Getting Started with DL/I . . . 10
Relationship of AIB and PCB with Language Interfaces . . . 11
Language Unique Interfaces . . . 12
Language Independent Interfaces . . . 12
AIBTDLI . . . 12
CEETDLI . . . 12
Using DL/I Calls . . . 12
Message Call Functions . . . 12
System Service Call Functions . . . 13
Status Codes, Return Codes, and Reason Codes . . . 13
Exceptional Conditions . . . 14
Error Routines . . . 14
How Your Program Processes Messages . . . 14
Message Types . . . 14
What Input Messages Look Like . . . 15
What Output Messages Look Like . . . 16
What Happens When a Message is Processed . . . 17
Results of a Message: I/O PCB . . . 19
How IMS TM Edits Messages . . . 19
Printing Output Messages . . . 20
Using Basic Edit . . . 20
Editing Input Messages . . . 20
Editing Output Messages . . . 21
Using Intersystem Communication Edit . . . 21
Editing Input Messages . . . 21
Editing Output Messages . . . 21
Using Message Format Service . . . 21
Terminals and MFS . . . 22
MFS Input Message Formats . . . 22
MFS Output Message Formats . . . 27
Using LU 6.2 User Edit Exit (Optional) . . . 28
DB2 Considerations . . . 28

Chapter 2. Defining Application Program Elements . . . 31
Formatting DL/I Calls for Language Interfaces . . . 31
Application Programming for Assembler Language . . . 32
Format . . . 32
Parameters . . . 33
Example DL/I Call Formats . . . 34
Application Programming for C Language . . . 34
Format . . . 34
Parameters . . . 35
I/O Area . . . 36

© Copyright IBM Corp. 1974, 2000

Example DL/I Call Formats . . . 37
Application Programming for COBOL . . . 37
Format . . . 37
Parameters . . . 38
Example DL/I Call Formats . . . 39
Application Programming for Pascal . . . 39
Format . . . 39
Parameters . . . 41
Example DL/I Call Formats . . . 42
Application Programming for PL/I . . . 42
Format . . . 42
Parameters . . . 43
Example DL/I Call Formats . . . 44
Relationship of Calls to PCB Types . . . 44
Specifying the I/O PCB Mask . . . 45
Specifying the Alternate PCB Mask . . . 48
Specifying the AIB Mask . . . 49
Specifying the I/O Areas . . . 51
Using the AIBTDLI Interface . . . 51
Overview . . . 51
Defining Storage for the AIB . . . 52
Specifying the Language-Specific Entry Point . . . 52
Assembler Language . . . 52
C Language . . . 52
COBOL . . . 53
Pascal . . . 53
PL/I . . . 53
Interface Considerations . . . 54
CEETDLI . . . 54
AIBTDLI . . . 54
PCB Lists . . . 54
Format of a PCB List . . . 54
Format of a GPSB PCB List . . . 54
PCB Summary . . . 55
Using Language Environment . . . 55
The CEETDLI interface to IMS . . . 56
LANG= Option on PSBGEN for PL/I Compatibility with Language Environment . . . 56
Special DL/I Situations . . . 56
Mixed-Language Programming . . . 57
Using Language Environment Routine Retention . . . 57
Using the Extended Addressing Capabilities of MVS/ESA . . . 57
Preloaded Programs . . . 57
DCCTL . . . 57

Chapter 3. Writing DL/I Calls for Transaction Management . . . 59
AUTH Call . . . 59
Format . . . 59
Parameters . . . 60
I/O Area . . . 60
I/O area before the AUTH call . . . 61
I/O area after the AUTH call . . . 62
Usage . . . 63
Restrictions . . . 64
CHNG Call . . . 64
Format . . . 64

Parameters . . . 65
Usage . . . 66
In the OTMA environment . . . 67
Advanced Print Function Options . . . 67
APPC Options . . . 69
Options List Feedback Area . . . 70
Error Codes . . . 71
Restrictions . . . 72
CMD Call . . . 72
Format . . . 72
Parameters . . . 72
Usage . . . 73
Restrictions . . . 73
GCMD Call . . . 74
Format . . . 74
Parameters . . . 74
Usage . . . 74
Restrictions . . . 75
GN Call . . . 75
Format . . . 75
Parameters . . . 75
Usage . . . 76
Restrictions . . . 76
GU Call . . . 76
Format . . . 76
Parameters . . . 76
Usage . . . 77
Restrictions . . . 78
ISRT Call . . . 78
Format . . . 78
Parameters . . . 78
Usage . . . 79
Spool API Functions . . . 79
Restrictions . . . 80
PURG Call . . . 80
Parameters . . . 80
Usage . . . 81
In the OTMA environment . . . 81
Spool API Functions . . . 82
Restrictions . . . 82
SETO Call . . . 82
Format . . . 82
Parameters . . . 82
Usage . . . 84
In the OTMA environment . . . 85
Advanced Print Function Options . . . 85
APPC Options . . . 85
Options List Feedback Area . . . 86
Error Codes . . . 86
Restrictions . . . 87

Chapter 4. Writing DL/I Calls for System Services . . . 89
APSB Call . . . 90
Format . . . 90
Parameters . . . 90
Usage . . . 90

Restrictions 91
CHKP (Basic) Call 91
Format 91
Parameters 91
Usage 92
Restrictions 92
CHKP (Symbolic) Call 92
Format 92
Parameters 92
Usage 93
Restrictions 93
DPSB Call 93
Format 93
Parameters 94
Usage 94
Restrictions 94
GMSG Call 94
Format 94
Parameters 94
Usage 95
Restrictions 96
GSCD Call 96
Format 96
Parameters 96
Usage 97
Restrictions 97
ICMD Call. 97
Format 97
Parameters 97
Usage 99
Restrictions 99
INIT Call 99
Format 100
Parameters. 100
Usage. 100
Determining Database Availability: INIT DBQUERY 100
Automatic INIT DBQUERY 101
Performance Considerations for the INIT Call (IMS Online Only) 101
INQY Call 102
Format 102
Parameters. 102
Usage. 103
Querying Information from the PCB: INQY Null 103
Querying Data Availability: INQY DBQUERY 106
Querying the Environment: INQY ENVIRON. 107
Querying the PCB Address: INQY FIND 108
Querying the Program Name: INQY PROGRAM 109
INQY Return Codes and Reason Codes 109
Map of INQY Subfunction to PCB Type 109
Restrictions. 109
LOG Call 109
Format 109
Parameters. 109
Usage. 111
Restrictions. 111
RCMD Call 111

Format 111
Parameters 111
Usage 112
Restrictions 112
ROLB Call 112
Format 112
Parameters 112
Usage 113
Restrictions 113
ROLL Call 114
Format 114
Parameters 114
Usage 114
Restrictions 114
ROLS Call 114
Format 115
Parameters 115
Usage 115
Restrictions 116
SETS/SETU Call 116
Format 116
Parameters 116
Usage 117
Restrictions 117
SYNC Call 118
Format 118
Parameters 118
Usage 118
Restrictions 118
XRST Call 118
Format 119
Parameters 119
Usage 120
Starting Your Program Normally 120
Restarting Your Program 120
Restrictions 121

Chapter 5. More about Message Processing 123
Sending Messages to Other Terminals and Programs 123
Sending Messages to Other Terminals 124
To One Alternate Terminal 124
To Several Alternate Terminals 124
Sending Messages to Other Application Programs 126
How the VTAM I/O Facility Affects Your VTAM Terminal 127
Communicating with Other IMS TM Systems Using MSC 128
Implications of MSC for Program Coding 128
Receiving Messages from Other IMS TM Systems 129
Sending Messages to Alternate Destinations in Other IMS TM Systems 130
IMS Conversations 131
A Conversational Example 131
Conversational Structure 132
What the SPA Contains 134
What Messages Look Like in a Conversation 136
Saving Information in the SPA 136
Replying to the Terminal 136
Using ROLB, ROLL, and ROLS in Conversations 137

Passing the Conversation to another Conversational Program 137
Restrictions on Passing the Conversation. 137
Defining the SPA Size 138
Conversational Processing and MSC 138
Ending the Conversation 139
Message Switching in APPC Conversations 139
DFSAPPC Format 140
Option Keywords. 140
Processing Conversations with APPC 141
Standard IMS Application Programs. 142
Standard IMS Application Programs and MSC 142
Modified IMS Application Programs 142
Modified IMS Application Programs and MSC 143
CPI-C Driven Application Programs 143
Ending the APPC Conversation 144
Coding a Conversational Program 144
Processing Conversations with OTMA 145
Backing out to a Prior Commit Point: ROLL, ROLB, and ROLS Calls 145
Using ROLL 146
Using ROLB 146
In MPPs and Transaction-Oriented BMPs. 147
In Batch Programs 147
Using ROLS 148
Backing out to an Intermediate Backout Point: SETS/SETU and ROLS. . . . 149
Using SETS/SETU 149
Using ROLS 150
Writing a Message-Driven Program 151
Coding DC Calls and Data Areas. 152
Your Input 152
Skeleton MPP. 152
Coding Your Program in Assembler Language 153
Coding Your Program in C Language 153
Coding Your Program in COBOL 155
Coding Your Program in Pascal 157
Coding Your Program in PL/I 160

Chapter 1. How Application Programs Work with the IMS Transaction Manager

Your application program uses IMS Transaction Manager (IMS TM) to process input and output messages, and uses Data Language I (DL/I) to communicate with IMS. This chapter provides an overview of the transaction management process.

In this Chapter:

- "Application Program Environments"
- "The Application Programming Interface"
- "Getting Started with DL/I" on page 10
- "Using DL/I Calls" on page 12
- "How Your Program Processes Messages" on page 14
- "How IMS TM Edits Messages" on page 19
- "DB2 Considerations" on page 28

Application programming techniques and the application programming interface are discussed here as they apply to IMS. IMS furnishes transaction management functions for the Database Data Communication (DB/DC) and the Data Communications Control (DCCTL) environments.

Related Reading: If your installation uses IMS Database Manager (IMS DB), refer to *IMS/ESA Application Programming: Database Manager* for information on the database functions required by your application programs.

Application Program Environments

IMS has various environments in which you can execute application programs. The three IMS online environments are:

- DB/DC
- DBCTL
- DCCTL

The two IMS batch environments are:

- DB batch, which is generated from DB/DC and DBCTL class system generations
- TM batch, which is generated from DCCTL class system generations

This book explains the DB/DC, DCCTL, and TM batch environments.

Related Reading: For information on DBCTL and DB batch environments, see *IMS/ESA Application Programming: Database Manager*.

The Application Programming Interface

This section provides an overview of the role your application program plays in the IMS TM system. For additional, system-level information on IMS TM, see *IMS/ESA Administration Guide: Transaction Manager* and *IMS/ESA Administration Guide: System*.

© Copyright IBM Corp. 1974, 2000

Your Application in the System

The IMS environments described within this subsection are DB/DC, DCCTL, and TM batch.

The DB/DC Environment

An application program must reside in a dependent region of IMS. Figure 1 shows how an application program can be positioned in a DB/DC environment.

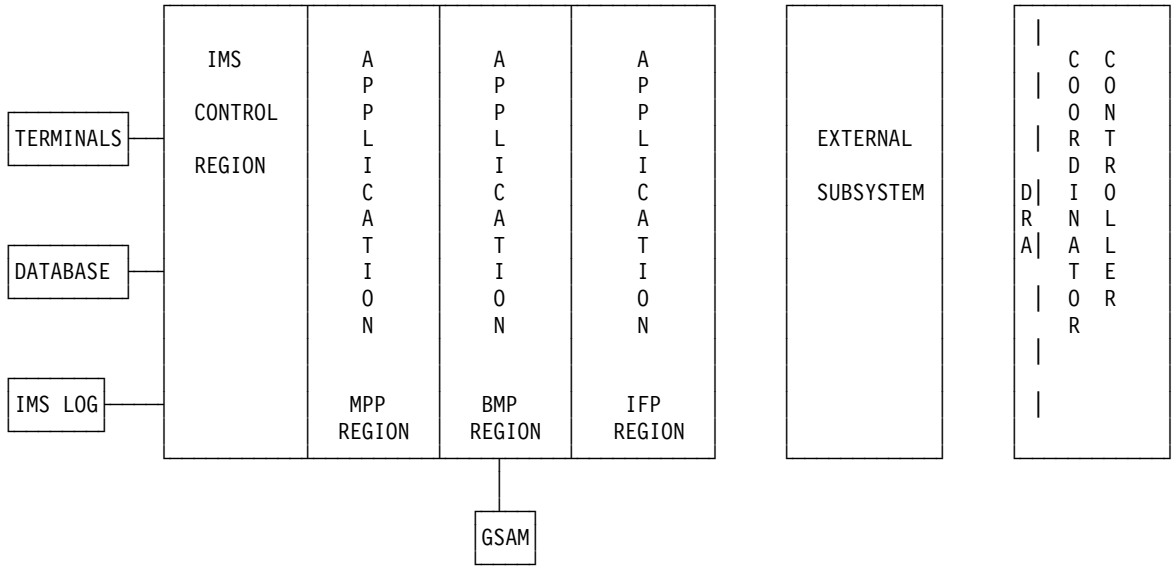


Figure 1. Application View of DB/DC Environment

Messages from terminals and application programs are processed by the IMS control region. The requested database segments or messages from other parts of the system (for example, status codes, system messages, or responses from terminals) are also processed by IMS and returned to the application.

The online environment can be used to access other types of database subsystems using the External Subsystem Attach facility (ESAF). It permits applications running with IMS to obtain data from external subsystems, such as DB2. Programming considerations for DB2 are described in the section “DB2 Considerations” on page 28.

The transaction management portion of the IMS DB/DC environment can be used separately to provide transaction management for external subsystems. This is the DCCTL environment.

The DCCTL Environment

The DCCTL environment functions like IMS TM in a DB/DC environment, except that DCCTL has no inherent database facilities. Instead, the DCCTL environment is used to access external subsystems, such as DB2. GSAM databases, which contain sequential non-IMS data sets, can be accessed by BMPs.

Most DL/I message processing and system service calls are supported in DCCTL. Supported calls are listed in “Transaction Management Call Summary” on page 439.

DL/I calls that require access to IMS databases are not valid. Figure 2 shows the DCCTL environment with an external subsystem.

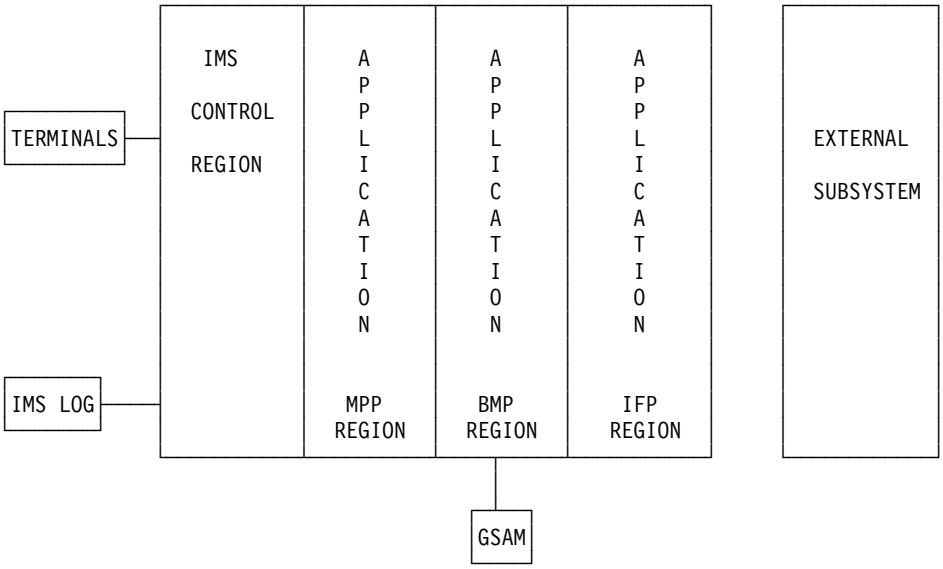


Figure 2. Application View of the DCCTL Environment

Application programs in the DCCTL environment can execute in the MPP, BMP, or IFP dependent regions. Except for the restriction on DL/I calls to databases, application programs in the DCCTL environment are the same as IMS TM applications.

Related Reading: For more information on IMS TM environments, refer to *IMS/ESA Administration Guide: System* or *IMS/ESA Administration Guide: Transaction Manager*.

The TM Batch Environment

TM Batch is the batch environment generated from DCCTL system generations. The TM Batch environment consists of a single address space which contains both IMS code and the application program. The batch region can be started as either a DLI or DBB type batch region. TM Batch application programs have access to DB2 databases through structured query language (SQL) calls, and to GSAM databases through DL/I calls.

Restriction: TM Batch applications cannot access DL/I databases. To access DB2, use the DB2 Batch Attach facility.

Related Reading: For more information on Batch Attach facility, see *DATABASE 2 Application Programming and SQL Guide*.

Using LU 6.2 Devices

Your applications can originate from or send messages to LU 6.2 devices. A standard IMS application program with no modification can send messages to LU 6.2 devices by specifying the devices as destinations in an alternate PCB or I/O PCB. To fully utilize the LU 6.2 protocol, you must use the Common Programming Interface (CPI) communications interface.

The Application Programming Interface

IMS TM and MVS provide support for the Advanced Program-to-Program Communication (APPC) facilities used for CPI Communications driven application programs. CPI-C driven applications use IMS TM to issue schedule requests, but rely on APPC/MVS to schedule and manage transactions.

Related Reading: For more information on writing application programs for APPC/IMS see *IMS/ESA Application Programming: Design Guide*. For more information on LU 6.2 and APPC, see *IMS/ESA Administration Guide: Transaction Manager*.

How IMS TM Schedules Application Programs

IMS TM begins the scheduling process for an application program when a message generated from a terminal or another application program requires processing. The transaction manager assigns this input message, or transaction, to an available dependent region and verifies that the application program is available to process the message. While the application processes the message, IMS TM controls availability to other requests for scheduling.

The PSB (defined by the PSBGEN utility) describes an application program to IMS TM and contains the program control blocks (PCBs) required by the application. If your application program requires only the I/O PCB and one modifiable alternate PCB, you can define the application with a generated PSB (GPSB) with the APPLCTN macro. PSBGEN is not required for GPSBs.

Related Reading: GPSBs and PSBs are discussed in more detail in “Chapter 5. More about Message Processing” on page 123.

Getting Started with DL/I

The information in this section applies to all programs that run in IMS environments. Figure 3 on page 11 shows the main elements in an IMS application program. The numbers on the right correspond to the notes that follow the figure.

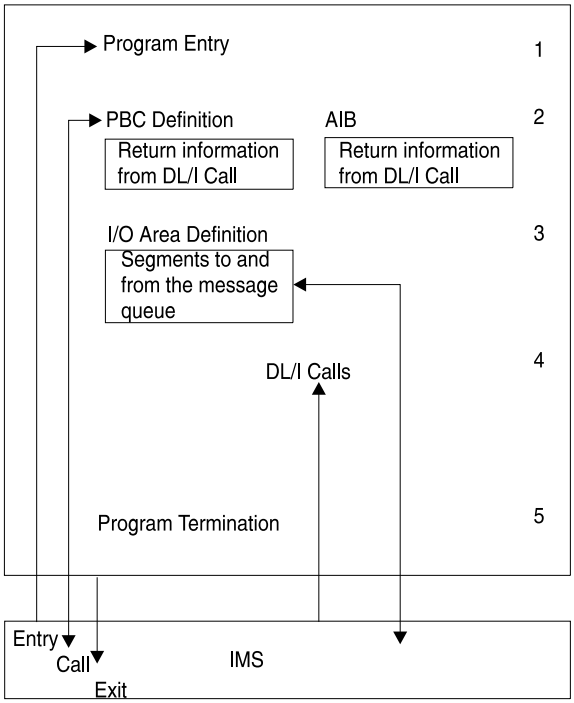


Figure 3. DL/I Program Elements

Notes to Figure 3:

1. **Program entry**. IMS passes control to the application program with a list of any associated PCBs.
2. **PCB or AIB**. IMS describes the results of each DL/I call using the AIBTDLI interface in the application interface block (AIB) and, when applicable, the program communication block (PCB). To find the results of a DL/I call, your program must use the PCB referenced in the call. To find the results of the call using the AIBTDLI interface, your program must use the AIB.

   Your application program can use the PCB address returned in the AIB to find the results of the call. To use the PCB, the program defines a mask of the PCB and can then reference the PCB after each call to find out about the success or failure of the call. An application program cannot change any fields in a PCB; it can only check the PCB to determine what happened when the call was completed.
3. **Input/output (I/O) area**. IMS passes segments to and from the program in the program's I/O area.
4. **DL/I calls**. The program issues DL/I calls to perform the requested function.
5. **Program Termination**. The program returns control to IMS TM when it has finished its processing. In a batch program, your program can set the return code and pass it to the next step in the job. If your program does not use the return code in this way, it is a good idea to set it to zero as a programming convention. Your program can also use the return code for this purpose in BMPs.

   Restriction: MPPs cannot pass return codes.

Relationship of AIB and PCB with Language Interfaces

IMS provides several language interfaces. These interfaces are either language unique or language independent.

Language Unique Interfaces

Language unique interfaces require the application to use the PCB address as one of the parameters. When IMS returns the results of the call to the application, the PCB mask must be used to analyze the call result. The following are language-unique interfaces:

- ASMTDLI: Assembler language interface to IMS
- CTDLI: C language interface to IMS
- CBLTDLI: COBOL language interface to IMS
- PASTDLI: PASCAL language interface to IMS
- PLITDLI: PL/I language interface to IMS

Language Independent Interfaces

AIBTDLI

AIBTDLI can be used by all IMS-supported languages. The application uses the AIB address as one of the parameters. When IMS returns the results of the call to the application, the AIB contains the address of the PCB used. You use the AIB mask to analyze the AIB and the call result. Similarly, you use the PCB mask to analyze the PCB and the call result.

CEETDLI

CEETDLI can only be used by programs running under either Language Environment for MVS & VM or under Language Environment for OS/390 & VM. The application can use either the PCB address or the AIB address as one of the parameters passed on IMS calls. If the AIB address is passed on the call, then after IMS returns the results of the call to the application, the AIB will contain the PCB address.You then use the AIB mask to analyze the AIB and the call result. If the PCB address was passed on the call, then after IMS returns the results of the call to the application, you use the PCB mask to analyze the PCB and the call result.

Using DL/I Calls

A DL/I call consists of a call statement and a list of parameters. The parameters for the call provide information IMS needs to execute the call. This information consists of the call function, the name of the data structure IMS uses for the call, the data area in the program into which IMS returns, and any condition the retrieved data must meet.

You can issue calls to perform transaction management functions (message calls) and to obtain IMS TM system services (system service calls):

Message Call Functions

The IMS TM message processing calls are:

AUTH	Authorization
CHNG	Change
CMD	Command
GCMD	Get Command
GN	Get Next
GU	Get Unique
ISRT	Insert

PURG	Purge
SETO	Set Options

System Service Call Functions

The IMS TM system service calls are:

APSB	Allocate PSB
CHKP	Checkpoint (Basic)
CHKP	Checkpoint (Symbolic)
DPSB	Deallocate PSB
GMSG	Get Message
GSCD [1]	Get System Contents Directory
ICMD	Issue Command
INIT	Initialize
INQY	Inquiry
LOG	Log
RCMD	Retrieve Command
ROLB	Roll Back
ROLL	Roll
ROLS	Roll Back to SETS
SETS	Set Synchronization Point
SETU	Set Synchronization Point (Unconditional)
SYNC	Synchronization
XRST	Extended Restart

Related Reading: The DL/I calls are discussed in detail in “Chapter 3. Writing DL/I Calls for Transaction Management” on page 59 and “Chapter 4. Writing DL/I Calls for System Services” on page 89. Reference tables for the calls appear in “Transaction Management Call Summary” on page 439.

Status Codes, Return Codes, and Reason Codes

To provide information on the results of each call, IMS TM places a 2-character status code in the PCB after each IMS TM call your program issues. Your program should check the status code after every IMS TM call it issues. If it does not, it might continue processing even though the last call caused an error.

The status codes your program should test for are those that indicate *exceptional but valid conditions*. Your program should first check for blanks, which indicate that the call was completely successful. If the status code IMS TM returns after a call is not one that you expected, your program should branch to an error routine.

Status codes returned in the PCB, return and reason codes returned in the AIB, or both, supply information for your calls.

1. GSCD is a Product-sensitive programming interface.

Related Reading: For detailed information on these codes, see “Chapter 15. DL/I Status Codes” on page 443 and “Chapter 16. DL/I Return and Reason Codes” on page 475.

Exceptional Conditions

Some status codes do not mean that your call was successful or unsuccessful; they just give you information about the results of the call. Your program uses this information to determine what to do next. The meanings of these status codes depend on the call.

In a typical program, the status codes that you should test for apply only to the get calls. Some status codes indicate exceptional conditions for other calls. When your program is retrieving messages, these are situations that you should expect and for which you should provide routines other than error routines. For example, QC means that no additional input messages are available for your program in the message queue, and QD means that no additional segments are available for this message.

Error Routines

If, after checking for blanks and exceptional conditions in the status code, you find that there has been an error, your program should branch to an error routine and print as much information as possible about the error before terminating. Print the status code as well. Information that is helpful in understanding the error include: which call was being executed when the error occurred, what were the parameters of the IMS call, and what were the contents of the PCB.

Two kinds of errors can occur in your program. First, programming errors are usually your responsibility; they are the ones you can find and fix. These errors are caused by things like an invalid parameter, an invalid call, or an I/O area that is too long. The other kind of error is something you cannot usually fix; this is a system or I/O error. When your program has this kind of error, the system programmer or the equivalent specialist at your installation should be able to help.

Because every application program should have an error routine available to it, and because each installation has its own ways of finding and debugging program errors, installations usually provide their own standard error routines.

How Your Program Processes Messages

To retrieve and send messages, an IMS TM application program issues calls to IMS TM. When your program issues a call to retrieve a message, IMS TM places the input message in the I/O area you name in the call. Before you issue a call to send a message, you must build the output message in an I/O area in your program.

Message Types

An operator at a terminal can send four kinds of messages to IMS TM. The destination of an IMS TM message identifies which kind of message is being sent:

- **Another terminal**. A logical terminal name in the first 8 bytes means that this is a message switch destined for another terminal. For a user at a logical terminal to send a message to another logical terminal, the user enters the name of the receiving logical terminal followed by the message. The IMS TM control region routes the message to the specified logical terminal. This kind of message does not involve an MPP.

- **An application program**. A transaction code in the first 8 bytes means that the message is destined for an application program. IMS TM uses a transaction code to identify MPPs and transaction-oriented BMPs. To use a particular application program to process requests, the user enters the transaction code for that application program.
- **IMS TM**. A "/" (slash) in the first byte means that the message is a command destined for IMS TM.
- **Message switch service**. A system service DFSAPPC request is destined for the message switch service.

An application program can send three kinds of messages:

- Commands. A "/" in the first byte of the message text means that the message is a command for IMS TM. Programmers design applications to issue commands when they want a program to perform tasks that an operator at a terminal usually performs. This is called *automated operator interface* (AOI) and is described in *IMS/ESA Customization Guide*.

  Use the `CMD` call to issue commands. Do not use the `ISRT` call for issuing commands, because a message created with `ISRT` can contain a slash in the first byte without being a command.
- Messages to logical terminals by specifying a logical terminal name.
- Program-to-program switches using a transaction code.

The messages that your program receives and sends are made up of segments. Use a `GU` call to retrieve the first segment of a new message, and use `GN` calls to retrieve the remaining segments of the message. Figure 4 shows three messages. Message A contains one segment; message B contains two segments; and message C contains three segments.

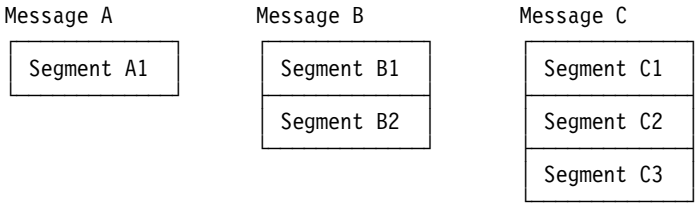


Figure 4. Message Segments

To retrieve message A, you only have to issue a `GU` call. To retrieve messages B and C, issue one `GU` call to retrieve the first segment, then a `GN` call for each remaining segment. This assumes that you know how many segments each message contains. If you do not know this, issue `GN` calls until IMS TM returns a QD status code, indicating that all of the segments for that message have been retrieved.

What Input Messages Look Like

The input message that an application program receives from a terminal or another program always has these fields: the length field, the ZZ field, and the text field. Figure 5 on page 16 shows the format of an input message.

For the AIBTDLI, ASMTDLI, CBLTDLI, CEETDLI, CTDLI and PASTDLI interfaces:

LL	ZZ	TRANCODE	Text
2	2	8	variable

For the PLITDLI interface:

LLLL	ZZ	TRANCODE	Text
4	2	8	variable

Figure 5. Input Message Format

The contents of the input message fields are:

LL or LLLL
The length field contains the length of the input message segment in binary, including LL (or LLLL) and ZZ. IMS TM supplies this number (in the length field when you retrieve the input message).

For the AIBTDLI, ASMTDLI, CBLTDLI, CEETDLI, CTDLI, and PASTDLI interfaces, define the LL field as 2 bytes long.

For the PLITDLI interface, define the LLLL field as 4 bytes long. The value in the LLLL field is the input message length minus 2 bytes. For example, if the text is 12 bytes, then the fullword LLLL contains a value of 24. This value is the total of LLLL (4 bytes) + ZZ (2 bytes) + TRANCODE (8 bytes) + text (12 bytes) − 2 bytes.

TRANCODE
The TRANCODE is the transaction code for the incoming message.

ZZ The ZZ field is a 2-byte field that is reserved for IMS TM. Your program does not modify this field.

Text
This field contains the message text sent from the terminal to the application program. The first segment of a message can also contain the transaction code associated with the program in the beginning of the text portion of the message. Input messages do not have to include the transaction code, but you can provide it for consistency.

The contents of the text field in the input message, and formatting of the contents when your program receives the message, depends on the editing routine your program uses.

What Output Messages Look Like

The format of the output message that you build to send back to a terminal or to another program is similar to the format of the input message, but the fields contain different information.

Output messages contain four fields: the length field, the Z1 field, the Z2 field, and the text field. Figure 6 on page 17 shows the format of an output message.

For the AIBTDLI, ASMTDLI, CBLTDLI, CEETDLI, CTDLI and PASTDLI interfaces:

LL	Z1	Z2	Text
2	1	1	variable

For the PLITDLI interface:

LLLL	Z1	Z2	Text
4	1	1	variable

Figure 6. Output Message Format

The contents of the output message fields are:

LL or LLLL
The output and input message length fields are the same. It contains the length of the message in binary, including the LL (or LLLL), Z1, and Z2 fields. For output message segments, supply this length when you are ready to send the message segment.

For the AIBTDLI, ASMTDLI, CBLTDLI, CEETDLI, CTDLI, and PASTDLI interfaces, the LL field must be 2 bytes long. For the PLITDLI interface, the LLLL field must be 4 bytes long and contains the length of the message segment, minus 2 bytes.

Z1 The Z1 field is a 1-byte field that must contain binary zeros. It is reserved for IMS TM.

Z2 The Z2 field is a 1-byte field that can contain special device-dependent instructions (such as instructions to ring the alarm bell, instructions to disconnect a switched line, or paging instructions) or device-dependent information (such as information about structured field data or bypassing MFS).

If you do not use any of these instructions, the Z2 field must contain binary zeros. For MFS, this field contains the number of the option that is being used for this message.

Text
The text portion of the message segment contains the data that you want to send to the logical terminal or to an application program. (Message texts are typically EBCDIC characters.) The length of the text depends on the data that you want to send.

What Happens When a Message is Processed

What a program does when it receives a message depends on the kind of message it receives. A transaction code associates a request for information from a terminal with the application program that can process and respond to that request. IMS TM schedules an MPP when there are messages to be processed that contain the transaction code associated with that MPP.

Example: For example, suppose you have an MPP that processes the transaction code “INVINQ” for inventory inquiry. The MPP receives a request from a user at a

terminal for information on the inventory of parts. When the user enters the transaction code for that application program, IMS TM schedules the application program that can process the request.

When you enter INVINQ and one or more part numbers, the MPP sends your program the quantity of each part on hand and the quantity on order.

When you enter INVINQ at the terminal, IMS TM puts the message on the message queue for the MPP that processes INVINQ. Then, after IMS TM has scheduled the MPP, the MPP issues GU and GN calls to retrieve the message. To retrieve the messages from LTERM1, the application program issues a GU for the first segment of a message, then issues GN calls until IMS TM returns a QD status code. This means that the program has retrieved all of the segments of that message. The program then processes the request, and sends the output message to the queue for your logical terminal. (The logical terminal name is in the I/O PCB.) When the MPP sends the output message, IMS TM sends it to the queue for that logical terminal, and the message goes to the physical terminal. Figure 7 shows the flow of a message between the terminal and the MPP.

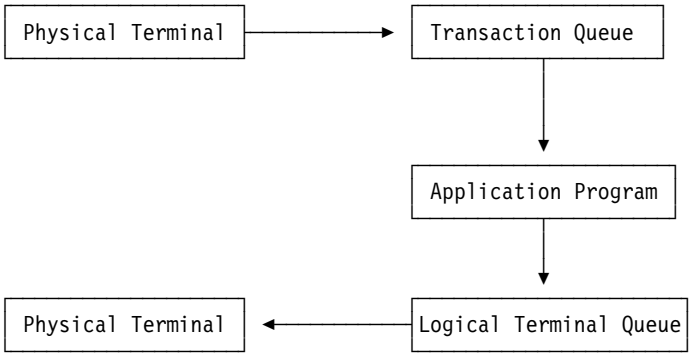


Figure 7. Transaction Message Flow

Figure 8 on page 19 shows the calls you use, the status codes, and what the input and output for the inventory inquiry would look like. To show you how you use GU and GN to retrieve messages, and how you insert multiple-segment messages, this example shows messages containing three segments. If input and output messages in this example were single segment messages, the program would issue only a GU to retrieve the entire message, and only one ISRT to send the message.

The message formats shown in Figure 8 on page 19 are examples; not all messages are in this format. When the program receives the input message in the I/O area, the first field of each segment contains the length of that segment. This is the LL field in the figure. For clarity, Figure 8 on page 19 shows this length in decimal; in the input message, however, it is in binary. The second field (ZZ) is reserved for IMS TM; it is 2 bytes long. The text of the message follows the reserved 2 bytes. The first message segment contains the transaction code in the 8 bytes following the ZZ field. These are the first 8 bytes of the text portion of the message.

The format of the output messages is the same. You do not need to include the name of the logical terminal, because it is in the first 8 bytes of the I/O PCB.

PART, QTY, and ON ORDER in Figure 8 are headings. These are values that you can define as constants that you want to appear on the terminal screen. To include headings in MFS output messages, define them as literals.

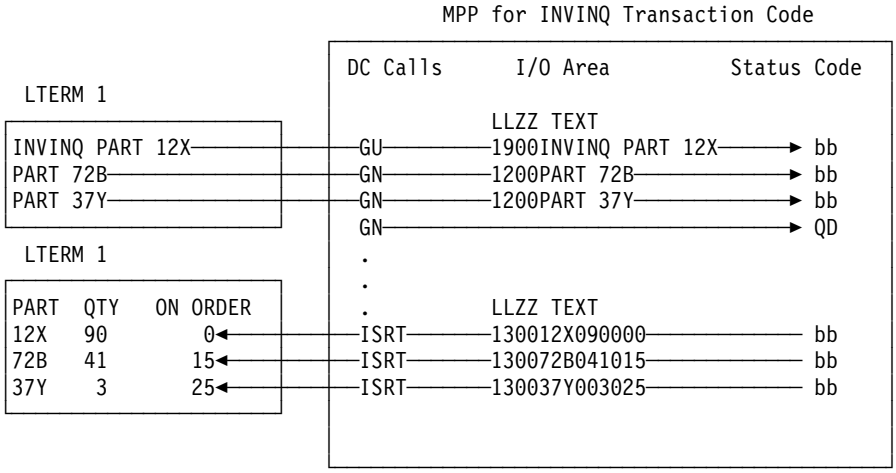


Figure 8. Inventory Inquiry MPP Example

Results of a Message: I/O PCB

After your program issues a call, IMS TM returns information about the results of the call in the I/O PCB. To find out about the results of the call, your application program must check the information that IMS TM returns to the I/O PCB.

When your application program retrieves a message, IMS TM returns the following information about the message to the I/O PCB:

- The name of the terminal that sent the message.
- A 2-character status code describing the results of the call. If the program receives a status code of QC after issuing a call to retrieve a message, no more messages are available for the program to process.
- The current date, time, and sequence number for the message.
- The user ID of the person at the terminal or the transaction code for the program that sent the message.

Because the I/O PCB resides outside of your program, you define a mask of the PCB in your program to check the results of IMS TM calls. The mask contains the same fields in the same order as the I/O PCB.

Related Reading: For more information on I/O PCB masks, see “Specifying the I/O PCB Mask” on page 45.

How IMS TM Edits Messages

When an application program passes messages to and from a terminal, IMS TM edits the messages before the program receives the message from the terminal and before the terminal receives the message from the application program. IMS TM gives you many choices about how you want your messages to appear both on the terminal screen and in the program’s I/O area. You need to know which editing routines have been specified for your program and how they affect your programming.

How IMS TM Edits Messages

The three editing routines available to non-LU 6.2 terminals in IMS TM are:

Basic Edit
Performs basic edit functions if you do not use MFS and if the message does not originate at an LU 6.1 device. You must provide control characters for some formatting functions.

Intersystem Communication (ISC) Edit
Provides the default edit for messages that originate from an LU 6.1 device. You can enter binary data in addition to text.

Message Format Service (MFS)
Formats messages through control blocks. You define the way the messages look with the control blocks.

For LU 6.2 devices, use the LU 6.2 Edit Exit to edit input and output messages.

Related Reading: For more information on LU 6.2, see *IMS/ESA Administration Guide: Transaction Manager*. For more information on LU 6.2 Edit Exit, see *IMS/ESA Customization Guide*.

Printing Output Messages

You must provide the horizontal and vertical control characters that are necessary to format your output messages.

To print your output at a printer terminal, include these control characters where necessary within the text of the message:

X'05' Skip to the tab stop, but stay on the same line.

X'15' Start a new line at the left margin.

X'25' Skip to a new line, but stay at the same place horizontally.

If you want to skip multiple lines, you can start a new line (X'15'), then skip as many lines as necessary (X'25').

Using Basic Edit

If you do not use MFS or an LU 6.1 device, IMS TM does some editing automatically. The editing IMS TM does to the first message segment is different from the editing IMS TM does for subsequent message segments. See *IMS/ESA Administration Guide: Transaction Manager* for a complete description of Basic Edit.

Editing Input Messages

When IMS TM receives the first segment of an input message for your application program, IMS TM:

- Removes leading and trailing control characters.
- Removes leading and trailing blanks.
- Removes backspaces (from a printer terminal).
- Translates to uppercase, if this is specified with the EDIT=UC specification on the system definition TRANSACT macro.

If the message segment contains a password, IMS TM edits the segment by:

- Removing the password and inserting a blank in place of the password.
- Removing the password if the first character of the text is a blank. IMS TM does not insert the blank.
- Left-justifying the text of the segment.

For subsequent input message segments, IMS TM does not remove leading blanks from the text of the message. The other formatting features are the same.

Editing Output Messages

For output messages, Basic Edit:

- Changes nongraphic characters in the output message before the data goes to the output device.
- Inserts any necessary idle characters after new line, line feed, and tab characters.
- Adds line control characters for the operation of the communication line.

Using Intersystem Communication Edit

Intersystem Communication (ISC) Edit is the default edit for messages from LU 6.1 devices. It is not valid for any other device types. One advantage of using ISC edit is that IMS TM does not edit the text of a message, allowing you to enter binary data.

Editing Input Messages

The editing IMS TM does to input messages depends on whether the Function Management (FM) header contains the SNA-defined primary resource name (PRN) parameter. In either case, IMS TM removes the FM header before the input message is received by the application program.

If the FM header *does not* contain the PRN parameter:

- IMS TM removes leading control characters and blanks when it receives the first segment of an input message for your application program.
- If the message segment contains a password, IMS TM removes the password and inserts a blank where the password was.
- IMS TM does not edit the text of the message (the data following the password).

If the FM header *does* contain the PRN parameter:

- The PRN is treated as the transaction code and is received by your application program as the first field in the message segment.
- The message segment is not edited by IMS TM.

Editing Output Messages

ISC Edit does not edit output messages.

Using Message Format Service

Message Format Service (MFS) is a part of IMS TM that uses control blocks that you define to format messages between a terminal and an MPP. The MFS control blocks indicate to IMS TM how you want your input and output messages arranged:

- For input messages, MFS control blocks define how the message that the terminal sends to your MPP is arranged in the I/O area.
- For output messages, MFS control blocks define how the message that your MPP sends to the terminal is arranged on the screen or at the printer. You can also define words or other data that appear on the screen (headings, for example) but do not appear in the program's I/O area. This data, called a *literal*, can be a field in the output message from the application program or a field in the input message from the terminal.

For detailed information on MFS, see "Part 2. Message Format Service" on page 163.

Terminals and MFS

Whether your program uses MFS depends on the types of terminals and secondary logical units (SLUs) your network uses. You can bypass MFS formatting of an output message for a 3270 device or for SLU Type 2 devices. When MFS is bypassed, you construct the entire 3270 data stream from within your program.

Restriction: MFS cannot be used with LU 6.2 devices (APPC).

Related Reading: For more information on LU 6.2 and APPC, see *IMS/ESA Administration Guide: Transaction Manager*.

The decisions about using MFS are high-level design decisions that are separate from the tasks of application design and application programming; many installations that use MFS have a specialist who designs MFS screens and message formats for all applications that use MFS.

MFS makes it possible for an MPP to communicate with different types of terminals without having to change the way it reads and builds messages. When the MPP receives a message from a terminal, the message's format in the MPP I/O area depends on the MFS options specified and not on what kind of terminal sent it. MFS shields the MPP from the physical device that is sending the message in the same way that a DB PCB shields the program from what the data in the database actually looks like and how it is stored.

MFS Input Message Formats

When you define a message to MFS, you do so in fields—just as you define fields within a database segment. When you define the fields that make up a message segment, you give MFS information such as:

- The length of the field
- The fill character to use when the length of the input data is less than the length that has been defined for the field
- Whether the data in the field is left-justified or right-justified
- If the field is truncated, whether it is truncated on the left or right

The order and length of these fields within the message segment depends on the MFS option that your program is using. You specify the MFS option in the MID. Which option to use for an application program is a design decision based on how complex the input data is, and how much it varies; the language the application program is written in; and the complexity of the application program. Performance factors are also involved in this decision.

The Z2 field in MFS messages contains the MFS formatting option being used to format the messages to and from your program. If something is wrong in the way that IMS TM returns the messages to your I/O area, and you suspect that the problem might be with the MFS option used, you can check this field to see if IMS TM is using the correct option. A X'00' in this field means that MFS did not format the message at all.

One way to understand how each of the MFS options formats your input and output messages is to look at examples of each option.

Example: Suppose that you have defined the four message segments shown in Figure 9 on page 23. Each of the segments contains a 2-byte length field and a 2-byte ZZ field. The first segment contains the transaction code that the person at

the terminal entered to invoke the application program. The number of bytes defined for each field appears below the name of the field in the figure.

When you use the PLITDLI interface, you must define the length field as a binary fullword, LLLL. When you use the AIBTDLI, ASMTDLI, CBLTDLI, CEETDLI, CTDLI, or PASTDLI interfaces, you must define the length field as a halfword, LL. The value provided by the PL/I application program must represent the actual segment length minus 2 bytes. For example, if the output text is 10 bytes, then the value of the fullword LLLL is 14 and is the sum of the length of LLLL (4 bytes − 2 bytes) + Z1 (1 byte) + Z2 (1 byte) + TEXT (10 bytes).

Segment 1

LL	ZZ			
0027	XXXX	TRANCODE	PATIENT#	NAME
2	2	8	5	10

Segment 2

0054	xxxx	ADDRESAF
2	2	50

Segment 3

0016	xxxx	CHARGES	PAYMENTS
2	2	6	6

Segment 4

0024	xxxx	TREATMNT	DOCTOR
2	2	10	10

Figure 9. Message Segment Formats

For these examples, assume the following:

- The transaction code is defined in the MID as a literal.
- All of the fields are left-justified.
- The fill character is defined as a blank. When the length of the data in a field is less than the length that has been defined for that field, MFS pads the field with fill characters. Fill characters can be:
  - Blanks
  - An EBCDIC character
  - An EBCDIC graphic character
  - A null, specified as X'3F'

  When you specify that the fill character is to be a null, MFS compresses the field to the length of the data if that length is less than the field length.

  The fields of the message segments in Figure 9 are arranged on the terminal screen in the format shown in Figure 10 on page 24.

Example: Assume the person enters the name of a patient, and the charges and payments associated with that patient.

```
PATIENT#:          NAME:  MC ROSS

ADDRESAF:

CHARGES: 106.50    PAYMENTS:  90.00

TREATMENT:

DOCTOR:
```

Figure 10. Terminal Screen for MFS Example

MFS provides three options for message format:

Option 1 Use this option when the program receives and transmits most of the fields in the message segments.

Option 2 Use this option when the program processes multisegment messages where most of the fields are transmitted but some of the segments are omitted.

Option 3 Use this option when the program receives and transmits only a few of the fields within a segment.

A description of each of these choices follows.

Option 1 Format: The way in which option 1 formats messages depends on whether you have defined a null as the fill character for any of the fields in the segment.

If none of the fields in the message were defined as having a fill character of null:

- The program receives all the segments in the message.
- Each segment is the length that was specified for it in the MID.
- Each segment contains all its fields.
- Each field contains data, data and fill characters, or all fill characters.

Figure 11 on page 25 shows what the message segments that the application program received would look like.

Segment 1

LL	Z	Z			
0027	XX	01	TRANCODE	blanks	MCROSSbbbb
2	1	1	8	5	10

Segment 2

0054	XX	01	blanks
2	1	1	50

Segment 3

0016	XX	01	010650	009000
2	1	1	6	6

Segment 4

0024	XX	01	blanks	blanks
2	1	1	10	10

Figure 11. Option 1 Message Format

The message format for option 1 output messages is the same as the input message format. The program builds output messages in an I/O area in the format shown above. The program can truncate or omit fields in one of two ways:

- Inserting a short segment
- Placing a null character in the field

If one or more of the fields are defined as having a null fill character, the message is different. In this case, the message has these characteristics:

- If a field has been defined as having a fill character of null and the terminal offers not data, the field is eliminated from the message segment.
- If all of the fields in a segment have a null fill character and none of the fields contains any literals, the segment is eliminated from the message.
- If only some of the fields in a segment have a null fill character, any field containing nulls is eliminated from the segment. The relative positions of the fields remaining within the segments are changed.
- When the length of the data that is received from the originating terminal is less than the length that is been defined for the field, the field is truncated to the length of the data.

Option 2 Format: Option 2 formats messages in the same way that option 1 does, unless the segment contains no input data from the terminal after IMS TM has removed the literals. If this is true, and if no additional segments in the message contain input data from the terminal, IMS TM ends the message. The last segment that the program receives is the last segment that contains input data from the terminal.

Sometimes a segment that does not have any input data from the terminal is followed by segments that do contain input data from the terminal. When this

happens, MFS gives the program the length field and the Z fields for the segment, followed by a 1-byte field containing X'3F' This indicates to the program that this is a null segment.

If the message segments shown in Figure 9 on page 23 are formatted by option 2, they appear in the format shown in Figure 12.

Segment 1

LL	Z	Z			
0027	XX	02	TRANCODE	blanks	MCROSSbbbb
2	1	1	8	5	10

Segment 2

0005	XX	02	3F
2	1	1	1

Segment 3

0016	XX	02	010650	0090000
2	1	1	6	6

Figure 12. Option 2 Message Format

Segment 2 in Figure 12 contains only a X'3F' because that segment is null, but Segment 3 contains data. This message does not contain a segment 4 because it is null.

Option 3 Format: When you use option 3, the program receives only those fields that have been received from the terminal. The program receives only segments that contain fields received from the originating terminal. Segments and fields can be of variable length if you have defined option 3 as having a null fill character.

A segment in an option 3 message is identified by its relative segment number—in other words, what position in the message it occupies. The fields within a segment are identified by their offset count within the segment.

Example: The NAME field in Segment 1 (MCROSSbbbb) has an offset value of 17. The value 17 is the sum of the lengths of the fields preceding the NAME field and includes an 8-byte transaction code and a 5-byte field of blanks. It does not include the 2-byte relative segment number field (field A in Figure 13 on page 27), the 2-byte length field (field B), or the 2-byte relative offset field (field C).

Option 3 messages do not contain literals defined in the MID. This means that the transaction code is removed from the message, except during a conversation. If the transaction that the program is processing is a conversational transaction, the transaction code is not removed from the message. The transaction code still appears in the Scratch Pad Area (SPA).

Each segment the program receives contains the relative number of this segment in the message (field A in Figure 13 on page 27). In addition, each data field within the segment is preceded by two fields:

- A 2-byte length field (B). This length includes the length field itself, the 2-byte relative field offset, and the data in the field.
- A 2-byte relative field offset (C), giving the field's position in the segment as defined in the MID.

These two fields are followed by the data field. MFS includes these fields for each field that is returned to the application program.

If the message segments shown in Figure 9 on page 23 are formatted by option 3, they appear in the format shown in Figure 13. The notes following the figure explain the letters A, B, C, and D above Segment 1 and Segment 3.

Segment 1

LL	Z	Z	A	B	C	D
0020	XX	03	0001	0014	0017	MCROSSbbbb
2	1	1	2	2	2	10

Segment 3

LL	Z	Z	A	B	C	D	B	C	D
0000	XX	03	0003	0010	0004	010650	0010	0010	009000
2	1	1	2	2	2	6	2	2	6

Figure 13. Option 3 Message Format

Notes to Figure 13:

- The fields marked A contain the relative segment number. This number gives the segment's position within the message.
- The fields marked B contain the field length. This length is the sum of the lengths of B field (2 bytes) + C field (2 bytes) + D field (the length of the data).
- The fields marked C contain the relative field offset. This gives each field's position within the segment.
- The fields marked D contain the data from the terminal. In this example, the fill character was defined as blank, so the data field is always its defined length. IMS TM does not truncate it. If you define the fill character as null, the lengths of the data fields can differ from the lengths defined for them in the segment. With a null fill character, if the length of the data from the terminal is less than the length defined for the field, IMS TM truncates the field to the length of the data. Using a null fill with option 3 reduces the space required for the message even further.

MFS Output Message Formats

For output messages, define to MFS what it is to receive from the application program. If using option 1 or option 2, the output message format is the same as it is for input messages. Present all fields and segments to MFS. You can present null segments. All fields in output messages are fixed length and fixed position. Output messages do not contain option numbers.

Option 3 output messages are similar to input messages, except that they do not contain option numbers. The program submits the fields as required in their segments with the position information.

Using LU 6.2 User Edit Exit (Optional)

This exit edits input and output messages from LU 6.2 devices when the implicit application program interface support is used. If it is not provided then messages are presented without modification. IMS does not invoke the exit for CPI-C driven transactions because IMS does not participate in the data flows when the application program uses the CPI directly.

The LU 6.2 User Edit Exit is called once for each message segment or inbound control flow. You can call the exit for data messages and use it to:

- Examine the contents of a message segment.
- Change the contents of a message segment.
- Expand or compact the contents of a message segment.
- Discard a message segment and process subsequent segments, if any.
- Use the Deallocate_Abend command to end the conversation.

For more information on LU 6.2 User Edit Exit, see *IMS/ESA Customization Guide* and *IMS/ESA Administration Guide: Transaction Manager*.

DB2 Considerations

For the most part, the message processing function of a dependent region that accesses DB2 databases is similar to that of a dependent region that accesses only DL/I databases. The method each program uses to retrieve and send messages and back out database changes is the same. Differences include the following:

- DL/I statements are coded differently from SQL (structured query language) statements.
- When an IMS TM application program receives control from IMS TM, IMS has already acquired the resources the program is able to access. IMS TM schedules the program, although some of the databases are not available. DB2 does not allocate resources for the program until the program issues its first SQL statement. If DB2 cannot allocate the resources your program needs, your program can optionally receive an initialization error when it issues its first SQL call.
- When an application issues a successful checkpoint call or a successful message `GU` call, DB2 closes any cursors that the program is using. This means that your program should issue its OPEN CURSOR statement **after** a checkpoint call or a message `GU`.

IMS TM and DB2 work together to keep data integrity in these ways:

- When your program reaches a commit point, IMS TM makes any changes that the program has made to DL/I databases permanent, releases output messages for their destinations, and notifies DB2 that the program has reached a commit point. DB2 then makes permanent any changes that the program has made to DB2 databases.
- When your program terminates abnormally or issues one of the IMS TM rollback calls (`ROLB`, `ROLS` without a token, or `ROLL`), IMS TM cancels any output messages your program has produced, backs out changes your program has made to DL/I databases since the last commit point, and notifies DB2. DB2 backs out the changes that the program has made to DB2 databases since the last commit point.

Through the Automated Operator Interface (AOI), IMS TM application programs can issue DB2 commands and IMS TM commands. To issue DB2 commands, the

program issues the IMS TM /SSR command followed by the DB2 command. The output of the /SSR command is routed to the master terminal operator (MTO).

DB2 Considerations

Chapter 2. Defining Application Program Elements

This chapter describes the elements of your application program that are used to communicate with IMS. Your application program must define these elements. The chapter also describes formatting DL/I calls for language interfaces and provides language calls information for assembler language, C language, COBOL, Pascal, and PL/I.

In this Chapter:

- "Formatting DL/I Calls for Language Interfaces"
- "Application Programming for Assembler Language" on page 32
- "Application Programming for C Language" on page 34
- "Application Programming for COBOL" on page 37
- "Application Programming for Pascal" on page 39
- "Application Programming for PL/I" on page 42
- "Relationship of Calls to PCB Types" on page 44
- "Specifying the I/O PCB Mask" on page 45
- "Specifying the Alternate PCB Mask" on page 48
- "Specifying the AIB Mask" on page 49
- "Specifying the I/O Areas" on page 51
- "Using the AIBTDLI Interface" on page 51
- "Specifying the Language-Specific Entry Point" on page 52
- "PCB Lists" on page 54
- "Using Language Environment" on page 55
- "Special DL/I Situations" on page 56

Related Reading: For detailed information on specific parameters for the DL/I calls see "Chapter 3. Writing DL/I Calls for Transaction Management" on page 59 and "Chapter 4. Writing DL/I Calls for System Services" on page 89.

Formatting DL/I Calls for Language Interfaces

When you use DL/I calls in a programming language supported by IMS (assembler language, C language, COBOL, Pascal, or PL/I), you must **call** the DL/I language interface to initiate the functions specified with the DL/I calls. IMS offers several interfaces for DL/I calls:

- A language-independent interface for any programs that are Language Environment conforming (CEETDLI)
- Language-specific interfaces for all supported languages (xxxTDLI)
- A non-language-specific interface for all supported languages (AIBTDLI)

Because the exact syntax for calling the language interfaces varies among the programming languages, the following sections describe the language-specific format.

Not every DL/I call uses all the parameters shown.

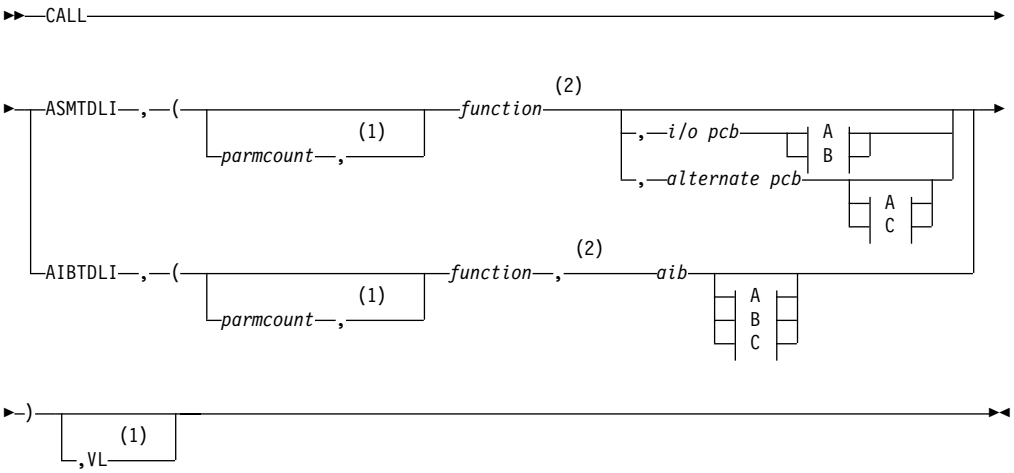
Related Reading: For descriptions of the call functions and the parameters they use, see "Chapter 3. Writing DL/I Calls for Transaction Management" on page 59 and "Chapter 4. Writing DL/I Calls for System Services" on page 89.

© Copyright IBM Corp. 1974, 2000

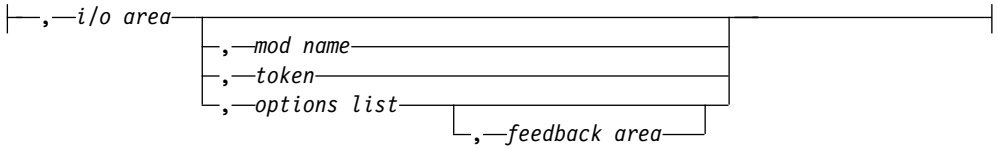
Application Programming for Assembler Language

This section contains the format, parameters, and sample DL/I call formats for IMS application programs in assembler language. In assembler language programs, all DL/I call parameters that are passed as addresses can be passed in a register, which, if used, must be enclosed in parentheses.

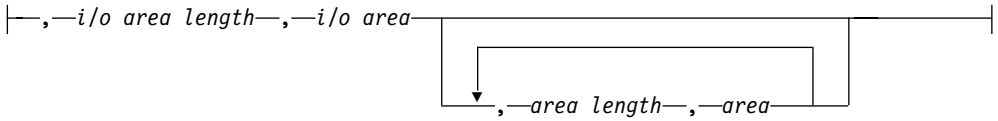
Format



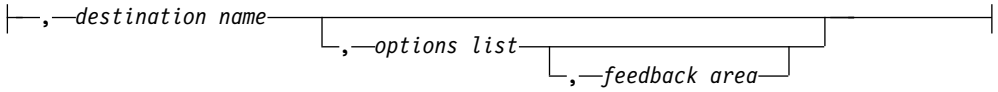
A:



B:



C:



Notes:

1 Assembler language must use either *parmcount* or **VL**.

2 See “Chapter 3. Writing DL/I Calls for Transaction Management” on page 59

and “Chapter 4. Writing DL/I Calls for System Services” on page 89 for descriptions of call functions and parameters.

Parameters

parmcount
Specifies the address of a 4-byte field in user-defined storage that contains the number of parameters in the parameter list that follows *parmcount*. Assembler language application programs must use either *parmcount* or **VL**.

function
Specifies the address of a 4-byte field in user-defined storage that contains the call function to be used. The call function must be left-justified and padded with blanks. An example would be (GUbb).

i/o pcb
Specifies the address of the PCB or alternate PCB to be used for the call. The PCB address must be one of the PCB addresses passed on entry to the application program in the PCB list.

alternate pcb
Specifies the address of the PCB or alternate PCB to be used for the call. The PCB address must be one of the PCB addresses passed on entry to the application program in the PCB list.

aib
Specifies the address of the application interface block (AIB) in user-defined storage. For more information on the contents of the AIB, see “Using the AIBTDLI Interface” on page 51.

i/o area
Specifies the address of the I/O area in user-defined storage used for the call. The I/O area must be large enough to contain the returned data.

i/o area length
Specifies the address of a 4-byte field in user-defined storage that contains the I/O area length (specified in binary).

area length
Specifies the address of a 4-byte field in user-defined storage that contains the length (specified in binary) of the area immediately following it in the parameter list. Up to seven area length/area pairs can be specified.

area
Specifies the address of the area in user-defined storage to be checkpointed. Up to seven area length/area pairs can be specified.

token
Specifies the address of a 4-byte field in user-defined storage that contains a user token.

options list
Specifies the address of the *options list* in user-defined storage that contains processing options used with the call.

feedback area
Specifies the address of the feedback area in user-defined storage that receives information about options list processing errors.

Assembler Language

mod name
Specifies the address of an 8-byte area in user-defined storage that contains the user-defined MOD name used with the call. The *mod name* parameter is used only with MFS.

destination name
Specifies the address of an 8-byte field in user-defined storage that contains the name of the logical terminal or transaction code to which messages resulting from the call are sent.

VL
Signifies the end of the parameter list. Assembler language programs must use either *parmcount* or **VL**.

Example DL/I Call Formats

DL/I AIBTDLI interface:

```
CALL  AIBTDLI,(function,aib,i/o area),VL
```

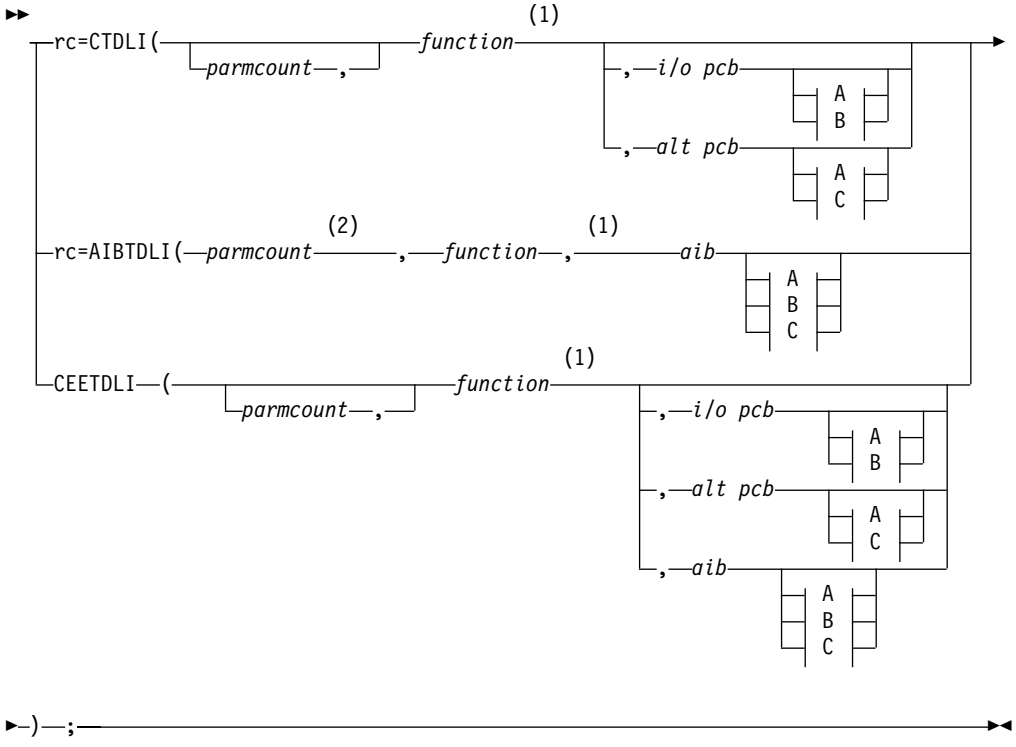
DL/I language-specific interface:

```
CALL  ASMTDLI,(function,i/o pcb,i/o area),VL
```

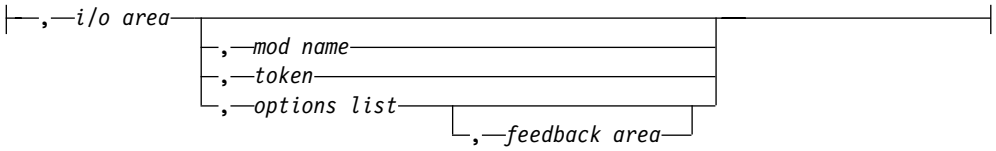
Application Programming for C Language

This section contains the format, parameters, and sample DL/I call formats for IMS application programs in C language.

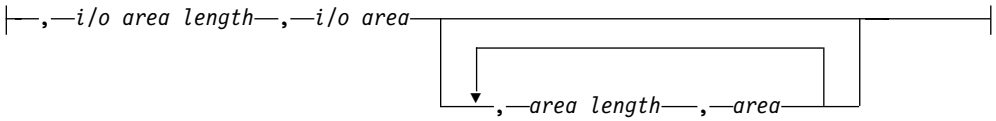
Format



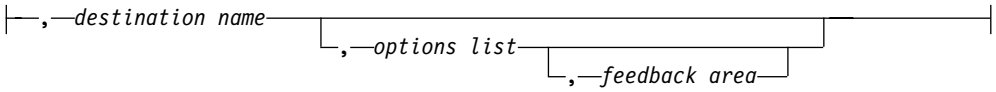
A:



B:



C:



Notes:

1 See “Chapter 3. Writing DL/I Calls for Transaction Management” on page 59 and “Chapter 4. Writing DL/I Calls for System Services” on page 89 for descriptions of call functions and parameters.

2 For AIBTDLI, *parmcount* is required for C language applications.

Parameters

rc Receives the DL/I status or return code. It is a 2-character field shifted into the 2 lower bytes of an integer variable (*int*). If the status or return code is two blanks, 0 is placed in the field. You can test the **rc** parameter with an **if** statement; for example, `if (rc == 'IX')`. You can also use **rc** in a **switch** statement. You can choose to ignore the value placed in **rc** and use the status code returned in the PCB instead.

parmcount
Specifies the name of a fixed-binary (31) variable in user-defined storage that is a pointer to the number of parameters in the parameter list that follows *parmcount*. The *parmcount* field is a pointer to long.

function
Specifies the name of a character (4) variable, left-justified, in user-defined storage, which contains the call function to be used. The call function must be padded with blanks. An example would be (GU␣␣).

i/o pcb
Specifies the name of a pointer variable that contains the address of the PCB or alternate PCB passed upon entry to the application program.

alternate pcb
Specifies the name of a pointer variable that contains the address of the PCB or alternate PCB passed upon entry to the application program.

aib
Specifies the name of the pointer variable that contains the address of the structure that defines the application interface block (AIB) in user-defined storage. For more information on the contents of the AIB, see “Using the AIBTDLI Interface” on page 51.

i/o area
Specifies the name of a pointer variable to a major structure, array, or character string that defines the I/O area in user-defined storage to be used for the call. The I/O area must be large enough to contain the returned data.

i/o area length
Specifies the name of a fixed-binary (31) variable in user-defined storage that contains the I/O area length.

area length
Specifies the name of a fixed-binary (31) variable in user-defined storage that contains the length of the area immediately following it in the parameter list. Up to seven area length/area pairs can be specified.

area
Specifies the name of the pointer variable that contains the address of the structure that defines the user-defined storage to be checkpointed. Up to seven area length/area pairs can be specified.

token
Specifies the name of a character (4) variable in user-defined storage that contains a user token.

options list
Specifies the name of the pointer variable that contains the address of the structure that defines the user-defined storage that contains processing options used with the call.

feedback area
Specifies the name of the pointer variable that contains the address of the structure that defines the user-defined storage that receives information about options list processing errors.

mod name
Specifies the name of a character (8) variable in user-defined storage that contains the user-defined MOD name used with the call. The *mod name* parameter is used only with MFS.

destination name
Specifies the name of a character (8) variable in user-defined storage that contains the name of the logical or terminal transaction code to which messages resulting from the call are sent.

I/O Area

In C language, the I/O area can be of any type, including structure or array. The **ceetdli** declarations in **leawi.h** and the **ctdli** declarations in **ims.h** do not have any prototype information, so no type checking of the parameters is done. The I/O area can be **auto**, **static**, or allocated (with **malloc** or **calloc**). Give special consideration to C-strings because DL/I does not recognize the C convention of terminating strings with nulls (`'\0'`). Instead of using the **strcpy** and **strcmp** functions, you might want to use the **memcpy** and **memcmp** functions.

Example DL/I Call Formats

DL/I CEEDTLI interface:

```
#include <leawi.h>
ceetdli(function,aib,i/o_area)
```

DL/I AIBTDLI interface:

```
int rc;
⋮
rc = aibtdli(parmcount,function,aib,i/o_area)
```

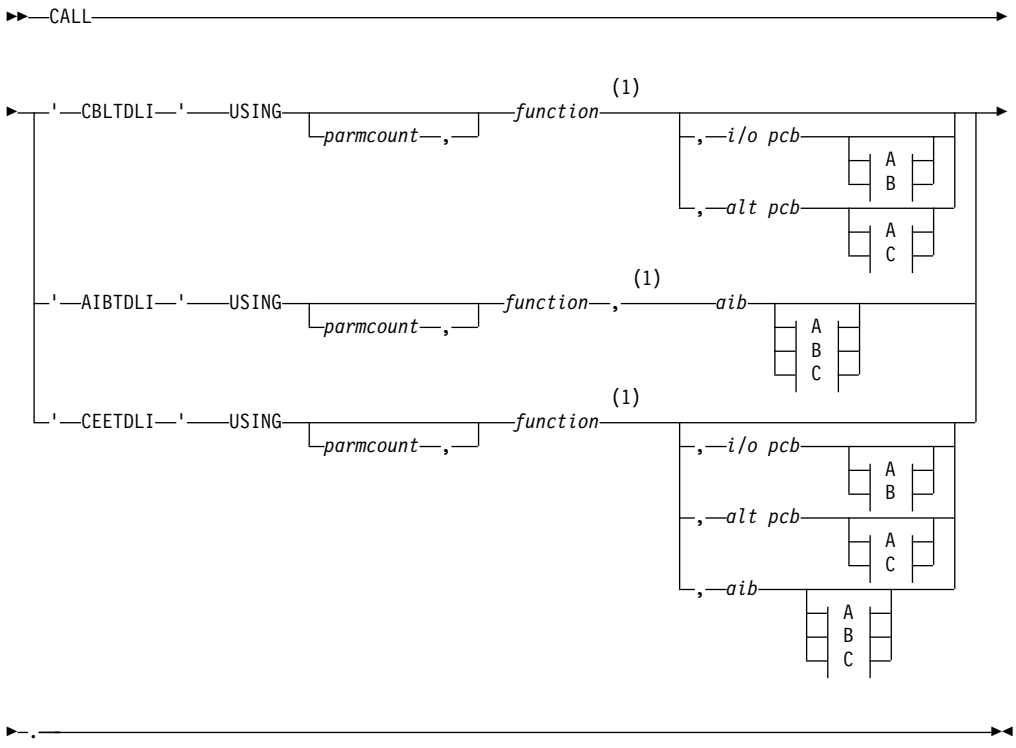
DL/I language-specific interface:

```
#include <ims.h>
int rc;
⋮
rc = ctdli(function,i/o_pcb,i/o_area)
```

Application Programming for COBOL

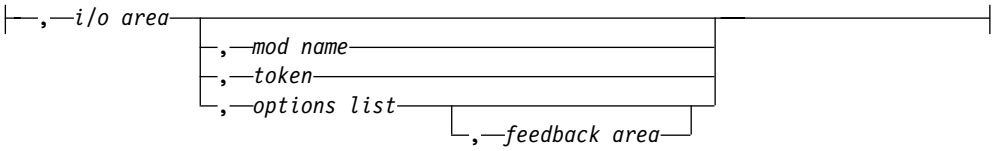
This section contains the format, parameters, and DL/I call sample formats for IMS application programs in COBOL.

Format

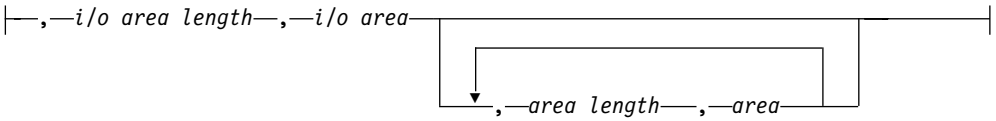


COBOL

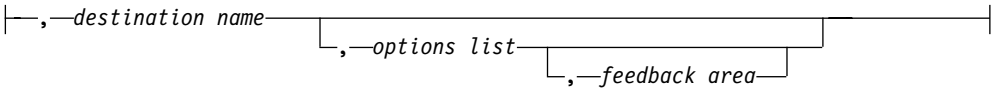
A:



B:



C:



Notes:

1 See “Chapter 3. Writing DL/I Calls for Transaction Management” on page 59 and “Chapter 4. Writing DL/I Calls for System Services” on page 89 for descriptions of call functions and parameters.

Parameters

parmcount
Specifies the identifier of a usage binary (4) byte data item in user-defined storage that contains the number of parameters in the parameter list that follows *parmcount*.

function
Specifies the identifier of a usage display (4) byte data item, left-justified, in user-defined storage, which contains the call function to be used. The call function must be padded with blanks. An example would be (GUbb).

i/o pcb
Specifies the identifier of the PCB group item from the PCB list passed to the application on entry.

alternate pcb
Specifies the identifier of the PCB group item from the PCB list passed to the application on entry.

aib
Specifies the identifier of the group item that defines the application interface block (AIB) in user-defined storage. For more information on the contents of the AIB, see “Using the AIBTDLI Interface” on page 51.

i/o area
Specifies the identifier of a group item, table, or usage display data item that defines the I/O area to be used for the call. The I/O area must be large enough to contain the returned data.

i/o area length
Specifies the identifier of a usage binary (4) byte data item in user-defined storage that contains the I/O area length.

area length
Specifies the identifier of a usage binary (4) byte data item in user-defined storage that contains the length of the area immediately following it in the parameter list. Up to seven area length/area pairs can be specified.

area
Specifies the identifier of the group item that defines the area to be checkpointed. Up to seven area length/area pairs can be specified.

token
Specifies the identifier of a usage display (4) byte data item that contains a user token.

options list
Specifies the identifier of the group item that defines the user-defined storage that contains processing options used with the call.

feedback area
Specifies the identifier of the group item that defines the user-defined storage that receives information about options list processing errors.

mod name
Specifies the identifier of a usage display (8) byte data item in user-defined storage that contains the user-defined MOD name used with the call.

destination name
Specifies the identifier of a usage display (8) byte data item that contains the name of the logical terminal or transaction code to which messages resulting from the call are sent.

Example DL/I Call Formats

DL/I CEETDLI interface:

```
CALL 'CEETDLI' USING function, aib,i/o area.
```

DL/I AIBTDLI interface:

```
CALL 'AIBTDLI' USING function, aib,i/o area.
```

DL/I language-specific interface:

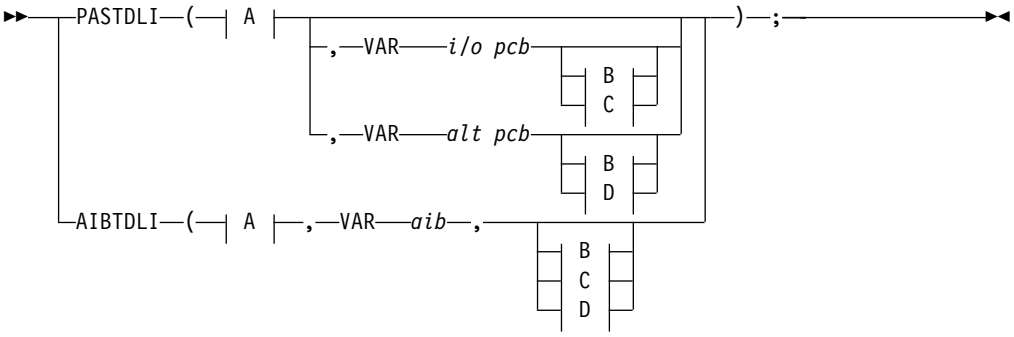
```
CALL 'CBLTDLI' USING function, i/o pcb, i/o area.
```

Application Programming for Pascal

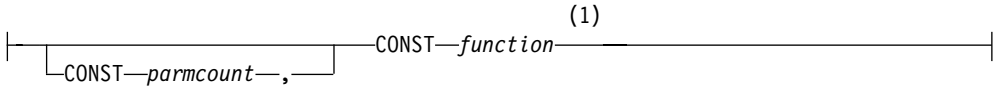
This section contains the format, parameters, and DL/I call sample formats for IMS application programs in Pascal.

Format

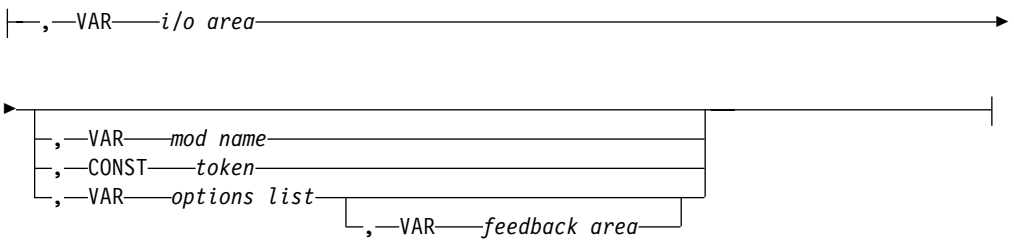
Pascal



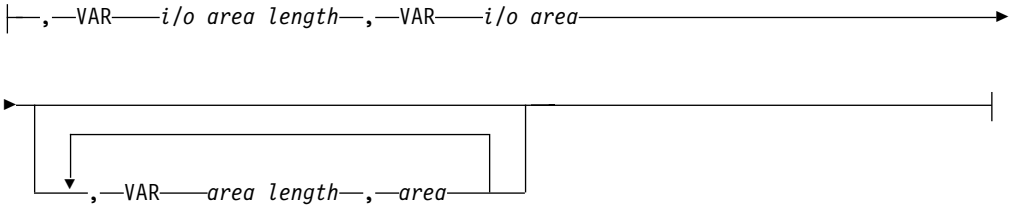
A:



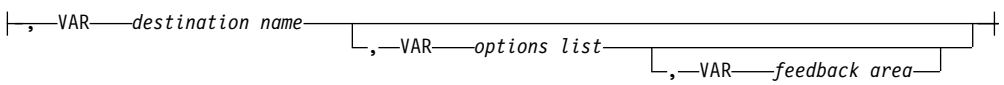
B:



C:



D:



Notes:

1 See “Chapter 3. Writing DL/I Calls for Transaction Management” on page 59 and “Chapter 4. Writing DL/I Calls for System Services” on page 89 for descriptions of call functions and parameters.

Parameters

parmcount
specifies the address of a fixed-binary (31) variable in user-defined storage that contains the number of parameters in the parameter list that follows *parmcount*.

function
Specifies the name of a character (4) variable, left-justified, in user-defined storage, which contains the call function to be used. The call function must be padded with blanks. An example would be (GUbb).

i/o pcb
Specifies the name of a pointer variable that contains the PCB defined in the procedure statement used for the call. This is the name used to declare the PCB in the procedure statement.

alternate pcb
Specifies the name of a pointer variable that contains the PCB defined in the procedure statement used for the call. This is the name used to declare the PCB in the procedure statement.

aib
Specifies the name of a pointer variable that contains the address of the structure that defines the application interface block (AIB) in user-defined storage. For more information on the contents of the AIB, see “Using the AIBTDLI Interface” on page 51.

i/o area
Specifies the name of a pointer variable to a major structure, array, or character string that defines the I/O area in user-defined storage to be used for the call. The I/O area must be large enough to contain the returned data.

i/o area length
Specifies the name of a fixed-binary (31) variable in user-defined storage that contains the I/O area length.

area length
Specifies the name of a fixed binary (31) variable in user-defined storage that contains the length (specified in binary) of the area immediately following it in the parameter list. Up to seven area length/area pairs can be specified.

area
Specifies the name of a pointer variable that contains the address of the structure that defines the area in user-defined storage to be checkpointed. Up to seven area length/area pairs can be specified.

token
Specifies the name of a character (4) variable in user-defined storage that contains a user token.

options list
Specifies the name of a pointer variable that contains the address of the structure that defines the user-defined storage that contains processing options used with the call.

feedback area
Specifies the name of the pointer variable that contains the address of the structure that defines the user-defined storage that receives information about options list processing errors.

Pascal

mod name
Specifies the name of a character (8) variable in user-defined storage that contains the user-defined MOD name used with the call.

destination name
Specifies the name of a character (8) variable in user-defined storage that contains the name of the logical terminal or transaction code to which messages resulting from the call are sent.

Example DL/I Call Formats

DL/I AIBTDLI interface:

```
AIBTDLI(CONST function,
        VAR aib,
        VAR I/O area);
```

DL/I language-specific interface:

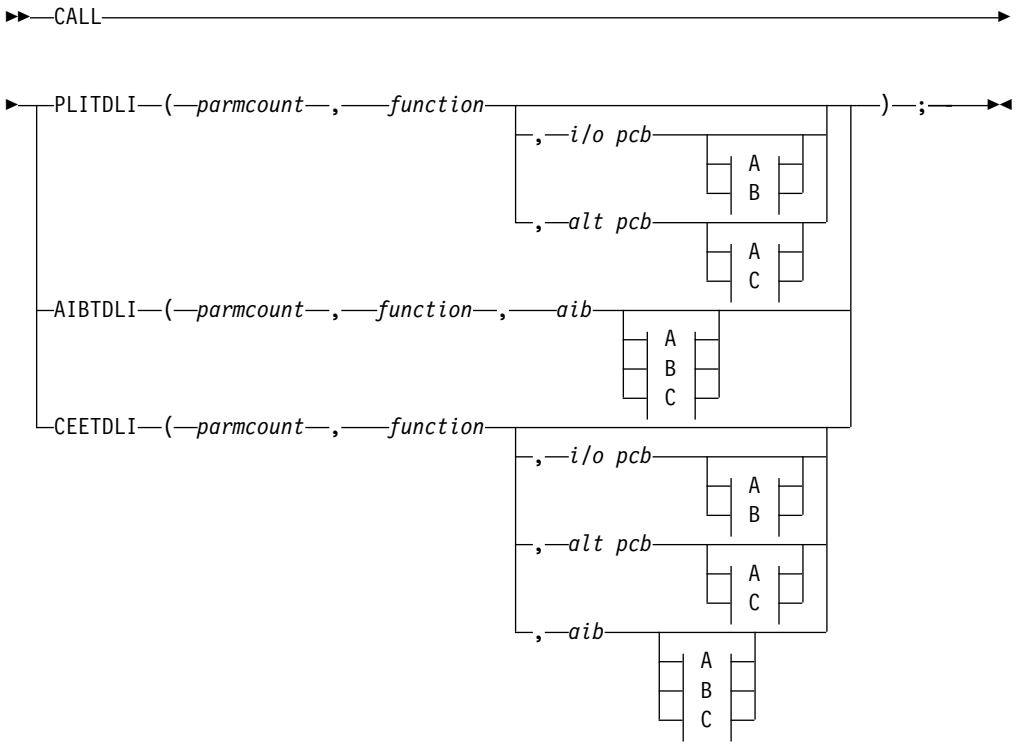
```
PASTDLI(CONST function,
area    VAR I/O PCB
        VAR I/O area);
```

Application Programming for PL/I

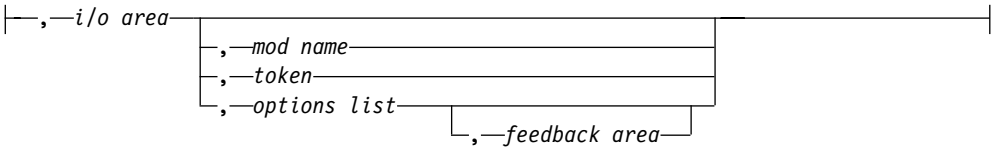
This section contains the format, parameters, and DL/I call sample formats for IMS application programs in PL/I.

For the PLITDLI interface all parameters except *parmcount* are indirect pointers; for the AIBTDLI interface, all parameters are direct pointers.

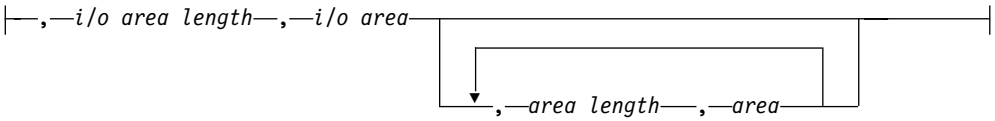
Format



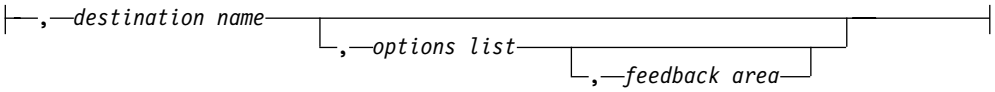
A:



B:



C:



Notes:

1 See “Chapter 3. Writing DL/I Calls for Transaction Management” on page 59 and “Chapter 4. Writing DL/I Calls for System Services” on page 89 for descriptions of call functions and parameters.

Parameters

parmcount
Specifies the name of a fixed-binary (31-byte) variable that contains the number of arguments that follow *parmcount*.

function
Specifies the name of a character (4-byte) variable, left justified, blank padded character string that contains the call function to be used. An example would be (GUbb).

i/o pcb
Specifies the name of the PCB to be used for the call.

alternate pcb
Specifies the name of the PCB to be used for the call.

aib
Specifies the name of the structure that defines the application interface block (AIB). For more information on the contents of the AIB, see “Using the AIBTDLI Interface” on page 51.

i/o area
Specifies the name of the I/O area used for the call. The I/O area must be large enough to contain the returned data.

i/o area length
Specifies the name of a fixed binary (31) variable in user-defined storage that contains the I/O area length (specified in binary).

area length
Specifies the name of a fixed binary (31) variable that contains the length (specified in binary) of the area immediately following it in the parameter list. Up to seven area length/area pairs can be specified.

area
Specifies the name of the area to be checkpointed. Up to seven area length/area pairs can be specified.

token
Specifies the name of a character (4) variable that contains a user token.

options list
Specifies the name of a structure that contains processing options used with the call.

feedback area
Specifies the name of a structure that receives information about options list processing errors.

mod name
Specifies the name of a character (8) variable character string containing the user-defined MOD name used with the call.

destination name
Specifies the name of a character (8) variable character string containing the logical terminal or transaction code to which messages resulting from the call are sent.

Example DL/I Call Formats

DL/I CEETDLI interface:

```
%INCLUDE CEEIBMAW;
CALL CEETDLI (function, i/o pcb, i/o area);
```

DL/I AIBTDLI interface:

```
CALL AIBTDLI (parmcount, function, aib, i/o area);
```

DL/I language-specific interface:

```
CALL PLITDLI (parmcount, function, i/o pcb, i/o area);
```

Relationship of Calls to PCB Types

Table 1 shows the relationship of DL/I calls to I/O PCBs and alternate PCBs. The PCB can be specified in one of two ways, depending on which xxxTDLI interface is used:

- As a parameter in the call list
- In the AIB

Table 1. Call Relationship to PCBs and AIBs

Call	I/O PCBs	ALT PCBs
APSB [1]		
AUTH	X	
CHKP (basic)	X	

Table 1. Call Relationship to PCBs and AIBs (continued)

Call	I/O PCBs	ALT PCBs
CHKP (symbolic)	X	
CHNG [2]		X
CMD	X	
DPSB [1]		
GCMD	X	
GN	X	
GSCD	X	
GU	X	
INIT	X	
INQY	X	X
ISRT	X	X
LOG	X	
PURG	X	X
ROLB	X	
ROLS	X	
ROLL [1]		
SETO	X	X
SETS	X	
SETU	X	
SYNC	X	
XRST	X	

Notes:

1. This call is not associated with a PCB.
2. The alternate PCB used by this call must be modifiable.

Specifying the I/O PCB Mask

After your program issues a call with the I/O Program Communications Block (I/O PCB), IMS returns information about the results of the call to the I/O PCB. To determine the results of the call, your program must check the information that IMS returns.

Issuing a system service call requires an I/O PCB. Because the I/O PCB resides outside your program, you must define a mask of the PCB in your program to check the results of IMS calls. The mask must contain the same fields, in the same order, as the I/O PCB. Your program can then refer to the fields in the PCB through the PCB mask.

An I/O PCB contains 10 fields. Table 2 shows these fields, their lengths, and the applicable environment for each field.

I/O PCB Mask

Table 2. I/O PCB Mask

Descriptor	Byte Length	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
Logical terminal name [1]	8	X		X		
Reserved for IMS [2]	2	X		X		
Status code [3]	2	X	X	X	X	X
Local date and time [4]	8	X		X		
Input message sequence number [5]	4	X		X		
Message output descriptor name [6]	8	X		X		
Userid [7]	8	X		X		
Group name [8]	8	X		X		
12-Byte Time Stamp [9]						
Date	4	X		X		
Time	6	X		X		
UTC Offset	2	X		X		

Notes:

1. **Logical Terminal Name**

   This field contains the name of the terminal that sent the message. When your program retrieves an input message, IMS places the name of the logical terminal that sent the message in this field. When you want to send a message back to this terminal, you refer to the I/O PCB when you issue the `ISRT` call, and IMS takes the name of the logical terminal from the I/O PCB as the destination.

2. **Reserved for IMS**

   This 2-byte field is reserved.

3. **Status Code**

   IMS places the status code describing the result of the DL/I call in this field. IMS updates the status code after each DL/I call that the program issues. Your program should always test the status code after issuing a DL/I call.

   The three status code categories are:

   - Successful status codes or status codes with exceptional but valid conditions. This category does not contain errors. If the call was completely successful, this field contains blanks. Many of the codes in this category are for information only. For example, a QC status code means that no more messages exist in the message queue for the program. When your program receives this status code, it should terminate.
   - Programming errors. The errors in this category are usually ones that you can correct. For example, an AD status code indicates an invalid function code.
   - I/O or system errors.

   For the second and third categories, your program should have an error routine that prints information about the last call that was issued before program termination. Most installations have a standard error routine that all application programs at the installation use.

4. **Local Date and Time**

   The current local date and time are in the prefix of all input messages except those originating from non-message-driven BMPs. The local date is a packed-decimal, right-aligned date, in the format `yyddd`. The local time is a packed-decimal time in the format `hhmmsst`. The current local date and time indicate when IMS received the entire message and enqueued it as input for the program, rather than the time that the application program received the message. To obtain the application processing time, you must use the time facility of the programming language you are using.

   For a conversation, for an input message originating from a program, or for a message received using Multiple System Coupling (MSC), the time and date indicate when the original message was received from the terminal.

5. **Input Message Sequence Number**

   The input message sequence number is in the prefix of all input messages except those originating from non-message-driven BMPs. This field contains the sequence number IMS assigned to the input message. The number is binary. IMS assigns sequence numbers by physical terminal, which are continuous since the time of the most recent IMS startup.

6. **Message Output Descriptor Name**

   You only use this field when you use MFS. When you issue a `GU` call with a message output descriptor (MOD), IMS places its name in this area. If your program encounters an error, it can change the format of the screen and send an error message to the terminal by using this field. To do this, the program must change the MOD name by including the MOD name parameter on an `ISRT` or `PURG` call.

   Although MFS does not support APPC, LU 6.2 programs can use an interface to emulate MFS. For example, the application program can use the MOD name to communicate with IMS to specify how an error message is to be formatted.

   Related Reading: For more information on the MOD name and the LTERM interface, see *IMS/ESA Administration Guide: Transaction Manager*.

7. **Userid**

   The use of this field is connected with RACF signon security. If signon is not active in the system, this field contains blanks.

   If signon is active in the system, the field contains one of the following:

   - The user's identification from the source terminal.
   - The LTERM name of the source terminal if signon is not active for that terminal.
   - The authorization ID. For batch-oriented BMPs, the authorization ID is dependent on the value specified for the BMPUSID= keyword in the DFSDCxxx PROCLIB member:
     - If BMPUSID=USERID is specified, the value from the USER= keyword on the JOB statement is used.
     - If USER= is not specified on the JOB statement, the program's PSB name is used.
     - If BMPUSID=PSBNAME is specified, or if BMPUSID= is not specified at all, the program's PSB name is used.

8. **Group Name**

   The group name, which is used by DB2 to provide security for SQL calls, is created through IMS transactions.

   Three instances that apply to the group name are:

- If you use RACF and SIGNON on your IMS system, the RACROUTE SAF (extract) call returns an eight-character group name.
- If you use your own security package on your IMS system, the RACROUTE SAF call returns any eight-character name from the package and treats it as a group name. If the RACROUTE SAF call returns a return code of 4 or 8, a group name was not returned, and IMS blanks out the group name field.
- If you use LU 6.2, the transaction header can contain a group name.

  Related Reading: See *IMS/ESA Administration Guide: Transaction Manager* for more information on LU 6.2.

9. **12-Byte Time Stamp**

   This field contains the current date and time fields, but in the IMS internal packed-decimal format. The time stamp has the following parts:

   Date `yyyydddf`

   This packed-decimal date contains the year (`yyyy`), day of the year (`ddd`), and a valid packed-decimal + sign such as (`f`).

   Time `hhmmssthmiju`

   This packed-decimal time consists of hours, minutes, and seconds (`hhmmss`) and fractions of the second to the microsecond (`thmiju`). **No** packed-decimal sign is affixed to this part of the timestamp.

   UTC Offset `aqq$`

   The packed-decimal UTC offset is prefixed by 4 bits of attributes (`a`). If the 4th bit of (`a`) is 0, the time stamp is UTC; otherwise, the timestamp is local time. The control region parameter, `TSR=(U/L)`, specified in the DFSPBxxx PROCLIB member, controls the representation of the time stamp with respect to local time versus UTC time.

   The offset value (`qq$`) is the number of quarter hours of offset to be added to UTC or local time to convert to local or UTC time respectively.

   The offset sign (`$`) follows the convention for a packed-decimal plus or minus sign.

   Field 4 on page 47 always contains the local date and time.

Related Reading: For a more detailed description of the internal packed-decimal time-format, see *IMS/ESA DBRC Guide and Reference*.

Specifying the Alternate PCB Mask

An alternate PCB mask contains three fields. Table 3 shows these fields, the field length, and in which environment the field applies.

Table 3. Alternate PCB Mask

Descriptor	Byte Length	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
Logical terminal name [1]	8 bytes	X		X		
Reserved for IMS [2]	2 bytes	X		X		
Status code [3]	2 bytes	X		X		

Notes:

1. **Logical Terminal Name**

   This field contains the name of the logical terminal, LU 6.2 descriptor or the transaction code to which you want to send the message.

   Related Reading: For more information on LU 6.2, see *IMS/ESA Administration Guide: Transaction Manager*.

2. **Reserved for IMS**

   This 2-byte field is reserved.

3. **Status Code**

   This field contains the 2-byte status code that describes the results of the call that used this PCB most recently.

For information on when to use an alternate PCB, see “Sending Messages to Other Terminals and Programs” on page 123.

Specifying the AIB Mask

The AIB is used by your program to communicate with IMS, when your application does not have a PCB address or the call function does not use a PCB. The AIB mask enables your program to interpret the control block defined. The AIB structure must be defined in working storage on a fullword boundary, and initialized according to the order and byte length of the fields as shown in Table 4. The notes below the figure describe the contents of each field.

Table 4. AIB Fields

Descriptor	Byte Length	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
AIB identifier [1]	8	X	X	X	X	X
DFSAIB allocated length [2]	4	X	X	X	X	X
Subfunction code [3]	8	X	X	X	X	X
Resource name [4]	8	X	X	X	X	X
Reserved [5]	16					
Maximum output area length [6]	4	X	X	X	X	X
Output area length used [7]	4	X	X	X	X	X
Reserved [8]	12					
Return code [9]	4	X	X	X	X	X
Reason code [10]	4	X	X	X	X	X
Error code extension [11]	4	X		X		
Resource address [12]	4	X	X	X	X	X
Reserved [13]	48					

Notes:

1. **AIB Identifier (AIBID)**

   This 8-byte field contains the AIB identifier. You must initialize AIBID in your application program to the value DFSAIBbb before you issue DL/I calls. This field is required. When the call is completed, the information returned in this field is unchanged.

2. **DFSAIB Allocated Length (AIBLEN)**

   This field contains the actual 4-byte length of the AIB as defined by your program. You must initialize AIBLEN in your application program before you issue DL/I calls. The minimum length required is 128 bytes. When the call is completed, the information returned in this field is unchanged. This field is required.

3. **Subfunction Code (AIBSFUNC)**

   This 8-byte field contains the subfunction code for those calls that use a subfunction. You must initialize AIBSFUNC in your application program before you issue DL/I calls. When the call is completed, the information returned in this field is unchanged.

4. **Resource Name (AIBRSNM1)**

   This 8-byte field contains the name of a resource. The resource varies depending on the call. You must initialize AIBRSNM1 in your application program before you issue DL/I calls. When the call is complete, the information returned in this field is unchanged. This field is required.

   For PCB related calls where the AIB is used to pass the PCB name instead of passing the PCB address in the call list, this field contains the PCB name. The PCB name for the I/O PCB is IOPCBbb. The PCB name for other types of PCBs is defined in the PCBNAME= parameter in PSBGEN.

5. **Reserved**

   This 16-byte field is reserved.

6. **Maximum Output Area Length (AIBOALEN)**

   This 4-byte field contains the length of the output area in bytes that was specified in the call list. You must initialize AIBOALEN in your application program for all calls that return data to the output area. When the call is completed, the information returned in this area is unchanged.

7. **Used Output Area Length (AIBOAUSE)**

   This 4-byte field contains the length of the data returned by IMS for all calls that return data to the output area. When the call is completed this field contains the length of the I/O area used for this call.

8. **Reserved**

   This 12-byte field is reserved.

9. **Return code (AIBRETRN)**

   When the call is completed, this 4-byte field contains the return code.

10. **Reason Code (AIBREASN)**

    When the call is completed, this 4-byte field contains the reason code.

11. **Error Code Extension (AIBERRXT)**

    This 4-byte field contains additional error information depending on the return code in AIBRETRN and the reason code in AIBREASN.

12. **Resource Address (AIBRSA1)**

    When the call is completed, this 4-byte field contains call-specific information. For PCB related calls where the AIB is used to pass the PCB name instead of passing the PCB address in the call list, this field returns the PCB address.

13. **Reserved**

    This 48-byte field is reserved.

The application program can use the returned PCB address, when available, to inspect the status code in the PCB and to obtain any other information needed by the application program.

Related Reading: See *IMS/ESA Application Programming: Transaction Manager* for more information about the return codes and reason codes.

Specifying the I/O Areas

Use an I/O area to pass segments between the application program and IMS TM. What the I/O area contains depends on the type of call you are issuing:

- When your program retrieves a segment, IMS TM places the segment your program requested in the I/O area.
- When your program adds a new segment, your program first builds the new segment in the I/O area.
- Before modifying a segment, your program must first retrieve the segment. When your program retrieves the segment, IMS TM places the segment in an I/O area.

The format of the record segments you pass between your program and IMS can be fixed length or variable length. Only one difference is important to the application program: a message segment contains a 2-byte length field (or 4 bytes for the PLITDLI interface) at the beginning of the data area of the segment.

The I/O area for IMS TM calls must be large enough to hold the largest message segment your program retrieves from or sends to IMS TM.

Using the AIBTDLI Interface

This section explains how to use the application interface block (AIB), an interface between your application program and IMS.

Restriction: No fields in the AIB can be used by the application program except as defined by IMS.

Overview

When you use the AIBTDLI interface, you specify the PCB requested for the call by placing the PCB name (as defined by PSBGEN) in the resource name field of the AIB. You do not specify the PCB address. Because the AIB contains the PCB name, your application program can refer to the PCB name rather than the PCB address. Your application program does not need to know the relative PCB position in the PCB list. At completion of the call, the AIB returns the PCB address that corresponds to the PCB name passed by the application program.

The names of DB PCBs and alternate PCBs are defined by the user during PSBGEN. All I/O PCBs are generated with the PCB name IOPCBbbb. For a generated program specification block (GPSB), the I/O PCB is generated with the PCB name IOPCBbbb, and the modifiable alternate PCB is generated with the PCB name TPPCB1bb.

The ability to pass the PCB name means that you do not need to know the relative PCB number in the PCB list. In addition, the AIBTDLI interface enables your application program to make calls on PCBs that do not reside in the PCB list. The LIST= keyword controls whether the PCB is included in the PCB list. The LIST= keyword is defined in the PCB macro during PSBGEN.

Related Reading: See *IMS/ESA Utilities Reference: System* for more information.

Defining Storage for the AIB

The AIB resides in user-defined storage that is passed to IMS for DL/I calls that use the AIBTDLI interface. Upon call completion, IMS updates the AIB. Allocate at least 128 bytes of storage for the AIB.

Specifying the Language-Specific Entry Point

IMS gives control to an application program through an entry point. The formats for coding entry statements in assembler language, C language, COBOL, Pascal, and PL/I are shown in this section. Your entry point must refer to the PCBs in the order in which they are defined in the PSB.

IMS passes the PCB pointers to a PL/I program differently than it passes them to an assembler language, C language, COBOL, or Pascal program. In addition, Pascal requires that IMS pass an integer before passing the PCB pointers. IMS uses the LANG keyword or the PSBGEN statement of PSBGEN to determine the type of program to which it is passing control. Therefore, you must be sure that the language specified during PSBGEN is consistent with the language of the program.

Application interfaces that use the AIB structure (AIBTDLI or CEETDLI) use the PCB name rather than the PCB structure and do not require the PCB list to be passed at entry to the application program.

When you code each DL/I call, you must provide the PCB you want to use for that call. For all IMS TM application programs, the list of PCBs the program can access is passed to the program at its entry point.

Assembler Language

You can use any name for the entry statement to an assembler language DL/I program. When IMS passes control to the application program, register 1 contains the address of a variable-length fullword parameter list. Each word in the list contains the address of a PCB. Save the parameter list address before you overwrite the contents of register 1. IMS sets the high-order byte of the last fullword in the list to X'80' to indicate the end of the list. Use standard MVS linkage conventions with forward and backward chaining.

C Language

When IMS passes control to your program, it passes the addresses, in the form of pointers, for each of the PCBs your program uses. The usual **argc** and **argv** arguments are not available to a program invoked by IMS. The IMS parameter list is made accessible by using the **__pcblist** macro. You can directly reference the PCBs by **__pcblist[0]**, **__pcblist[1]**, or you can define macros to give these more meaningful names. I/O PCBs must be cast to get the proper type:

```
(IO_PCB_TYPE *)(__pcblist[0])
```

The entry statement for a C langu age program is the **main** statement.

```
#pragma runopts(env(IMS),plist(IMS))
#include <ims.h>

main()
{
.
.
.
}
```

The **env** option specifies the operating environment in which your C language program is to run. For example, if your C language program is invoked under IMS and uses IMS facilities, specify **env(IMS)**. The **plist** option specifies the format of the invocation parameters received by your C language program when it is invoked. When your program is invoked by a system support services program such as IMS, the format of the parameters passed to your main program must be converted into the C language format: **argv**, **argc**, and **envp**. To do this conversion, you must specify the format of the parameter list received by your C language program. The **ims.h** include file contains declarations for PCB masks.

You can finish in three ways:

- End the main procedure without an explicit **return** statement.
- Execute a **return** statement from **main**.
- Execute an **exit** or an **abort** call from anywhere, or alternately issue a **longjmp** back to **main**, and then do a normal return.

One C language program can pass control to another by using the **system** function. The normal rules for passing parameters apply. For example, when using the **system** function, the **argc** and **argv** arguments can be used to pass information. The initial __**pcblist** is made available to the invoked program.

COBOL

The procedure statement must refer to the I/O PCB first, then to any alternate PCB it uses, and finally to the DB PCBs it uses. The alternate PCBs and DB PCBs must be listed in the order in which they are defined in the PSB.

```
Procedure division using the PCB-NAME-1 [,...,PCB-NAME-N]
```

On previous versions of IMS, **using** might be coded on the entry statement to reference PCBs. However, IMS continues to accept such coding on the entry statement.

Recommendation: Use the procedure statement rather than the entry statement to reference the PCBs.

Pascal

The entry point must be declared as a REENTRANT procedure. When IMS passes control to a Pascal procedure, the first address in the parameter list is reserved for Pascal's use and the other addresses are the PCBs the program uses. The PCB types must be defined before this entry statement. The IMS interface routine PASTDLI must be declared with the GENERIC directive.

```
procedure ANYNAME(var SAVE: INTEGER;
                  var pcb1-name: pcb1-name-type[;
                  ...
                  var pcbn-name: pcbn-name-type]); REENTRANT;
procedure ANYNAME;
(* Any local declarations *)
  procedure PASTDLI; GENERIC;
begin
  (* Code for ANYNAME *)
end;
```

PL/I

The entry statement can be any valid PL/I name and must appear as the first executable statement in the program. When IMS passes control to your program, it

passes the addresses of each of the PCBs your program uses in the form of pointers. When you code the entry statement, make sure you code the parameters of this statement as pointers to the PCBs, and not the PCB names.

```
anyname: PROCEDURE (pcb1_ptr [,..., pcbn_ptr]) OPTIONS (MAIN);
⋮
RETURN;
```

Interface Considerations

This section explains the interfaces: CEETDLI and AIBTDLI

CEETDLI

The considerations are:

- For PL/I programs, the CEETDLI entry point is defined in the CEEIBMAW include file. Alternatively, you can declare it yourself. But it must be declared as an assembler language entry (DCL CEETDLI OPTIONS(ASM);).
- For C language applications, you must specify **env(IMS)** and **plist(IMS)**; these specifications enable the application to accept the PCB list of arguments. The CEETDLI function is defined in <leawi.h>; the CTDLI function is defined in <ims.h>.

AIBTDLI

The considerations are:

- When using the AIBTDLI interface for C/MVS, COBOL, or PL/I language applications, the language run-time options for suppressing abend interception (that is, NOSPIE and NOSTAE) must be specified. However, for Language Environment-conforming applications, the NOSPIE and NOSTAE restriction is removed.
- The AIBTDLI entry point for PL/I programs must be declared as an assembler language entry (DCL AIBTDLI OPTIONS(ASM);).
- For C language applications, you must specify **env(IMS)** and **plist(IMS)**; these specifications enable the application to accept the PCB list of arguments.

PCB Lists

This section describes the formats of PCB lists and GPSB PCB lists and provides a a description of PCBs in various types of application programs.

Format of a PCB List

PSBs have the following format:

```
[IOPCB]
[Alternate PCB ... Alternate PCB]
[DB PCB ... DB PCB]
[GSAM PCB ... GSAM PCB]
```

Each PSB must contain at least one PCB. An I/O PCB or alternate PCB is required for transaction management calls, and an I/O PCB is required for most system service calls. DB PCBs for DL/I databases are used only with the IMS Database Manager, but can be present even though your program is running under DCCTL. (A DB PCB can be a full-function PCB, a DEDB PCB, or an MSDB PCB.) GSAM PCBs can be used with DCCTL.

Format of a GPSB PCB List

A generated program specification block (GPSB) has the following format:

```
[IOPCB]
[Alternate PCB]
```

A GPSB contains only an I/O PCB and one modifiable alternate PCB. It can be used by all transaction management application programs, and permits access to the PCBs specified without the need for PSBGEN.

The PCBs in a GPSB have predefined PCB names. The name of the I/O PCB is IOPCBbb. The name of the alternate PCB is TPPCB1bb.

PCB Summary

This section summarizes the information concerning I/O PCBs and alternate PCBs in various types of application programs.

TM Batch Programs

Alternate PCBs are always included in the list of PCBs supplied to the program by IMS TM. The I/O PCB is always present in the PCB list regardless of the CMPAT options specified in PSBGEN.

BMPs, MPPs, and IFPs

I/O PCBs and alternate PCBs are always passed to BMPs, MPPs, and IFPs.

The PCB list always contains the address of the I/O PCB followed by the addresses of any alternate PCBs, followed by the addresses of the DB PCBs.

Using Language Environment

IBM Language Environment for MVS & VM provides the strategic execution environment for running your application programs written in one or more high level languages. It provides not only language-specific run-time support, but also cross-language run-time services for your applications, such as support for initialization, termination, message handling, condition handling, storage management, and National Language Support. Many of Language Environment's services are accessible explicitly through a set of Language Environment interfaces that are common across programming languages; these services are accessible from any Language Environment-conforming program.

Language Environment-conforming programs can be compiled with the following compilers:

- IBM C/C++ for MVS/ESA
- IBM COBOL for MVS & VM
- IBM PL/I for MVS & VM

These programs can be produced by programs coded in Assembler. All of these programs can use CEETDLI, the Language Environment-provided language-independent interface to IMS, as well as older language-dependent interfaces to IMS, such as CTDLI, CBLTDLI, and PLITDLI.

Although they do not conform to Language Environment, programs compiled with the following older compilers can run under Language Environment:

- IBM C/370
- COBOL
- IBM OS PL/I

Restriction: These programs cannot use CEETDLI, but they can use the older language-dependent interfaces to IMS.

The CEETDLI interface to IMS

The language-independent CEETDLI interface to IMS is provided by Language Environment. It is the only IMS interface that supports the advanced error handling capabilities provided by Language Environment. The CEETDLI interface supports the same functionality as the other IMS application interfaces, and it has the following characteristics:

- The **parmcount** variable is optional.
- Length fields are 2 bytes long.
- Direct pointers are used.

Related Reading: For more information about Language Environment, see *IBM Language Environment for MVS & VMIBM Language Environment for MVS & VM Programming Guide* and *Language Environment for MVS & VM Installation and Programming*.

LANG= Option on PSBGEN for PL/I Compatibility with Language Environment

For IMS PL/I applications running in a compatibility mode that uses the PLICALLA entry point, you must specify LANG=PLI on the PSBGEN. Your other option is to change the entry point and add SYSTEM(IMS) to the EXEC PARM of the compile step so that you can specify LANG=blank or LANG=PLI on the PSBGEN. Table 5 summarizes when you can use LANG=blank and LANG=PLI.

Table 5. Using LANG= Option in a Language Environment for PL/I Compatibility

Compile exec statement is PARM=(...,SYSTEM(IMS)...	and entry point name is PLICALLA	Then LANG= is as stated below:
Yes	Yes	LANG=PLI
Yes	No	LANG=blank or LANG=PLI
No	No	**Note:** Not valid for IMS PL/I applications
No	Yes	LANG=PLI

PLICALLA is only valid for PL/I compatibility with Language Environment. If a PL/I application using PLICALLA entry at link-edit time is link-edited using Language Environment with the PLICALLA entry, the link-edit will work; however, you must specify LANG=PLI in the PSB. If the application is re-compiled using PL/I for MVS & VM Version 1 Release 1 or later, and then link-edited using Language Environment Version 1 Release 2 or later, the link-edit will fail. You must remove the PLICALLA entry statement from the link-edit.

Special DL/I Situations

This section contains information on mixed-language programming, using the extended addressing capabilities of MVS/ESA, COBOL compiler options for preloaded programs, and considerations for the DCCTL environment.

Mixed-Language Programming

When an application program uses the Language Environment language-independent interface, CEETDLI, IMS does not need to know the language of the calling program.

When the application program calls IMS in a language-dependent interface, IMS determines the language of the calling program according to the entry name specified in the CALL statement:

- CALL CBLTDLI indicates the program is in COBOL.
- CALL PLITDLI indicates the program is in PL/I.
- CALL PASTDLI indicates the program is in Pascal.
- **ctdli(...)** indicates the program is in C language.
- CALL ASMTDLI indicates the program is in assembler language.

If a PL/I program calls an assembler language subroutine and the assembler language subroutine makes DL/I calls by using CALL ASMTDLI, the assembler language subroutine should use the assembler language calling convention, not the PL/I convention.

In this situation, where the I/O area uses the LLZZ format, the LL is a halfword, not the fullword that is used for PLITDLI.

For more information on Language Environment, see “Using Language Environment” on page 55.

Using Language Environment Routine Retention

If you run programs in an IMS TM dependent region that requires Language Environment (such as an IMS message processing region), you can improve performance if you use Language Environment library routine retention along with the existing PREINIT feature of IMS TM. For more information on this, see *IBM Language Environment for MVS & VM Programming Guide* and *IBM Language Environment for MVS & VM Installation and Customization*.

Using the Extended Addressing Capabilities of MVS/ESA

The two modes in MVS/ESA with extended addressing capabilities are: the addressing mode (AMODE) and the residency mode (RMODE). For more detailed information about the AMODE and RMODE, see *MVS/ESA System Programming Library: 32-bit Addressing*. IMS places no constraints on the RMODE and AMODE of an application program. The program can reside in the extended virtual storage area. The parameters referenced in the call can also be in the extended virtual storage area.

Preloaded Programs

If you compile your COBOL program with the COBOL for MVS & VM compiler and preload it, you must use the COBOL compiler option RENT. Alternatively, if you compile your COBOL program with the VS COBOL II compiler and preload it, you must use the COBOL compiler options RES and RENT.

DCCTL

In a DCCTL environment, the application can only reference an I/O PCB, alternate PCB, or GSAM PCB. Entry statements for COBOL, PL/I, C, and Pascal must refer

to all PCBs that are to be referenced. This includes all PCBs prior to the last referenced PCB and can include DB PCBs. If you used a GSAM PCB, all PCBs ahead of it must be referenced.

Chapter 3. Writing DL/I Calls for Transaction Management

This chapter describes the format for DL/I calls you can use with IMS TM to perform transaction management functions in your application program. Calls within the section are in alphabetical order. Transaction management calls must use either *i/o pcb* or *aib* parameters.

Each call description contains:

- A syntax diagram
- A definition for each parameter that can be used in the call
- Details on how to use the call in your application program
- Restrictions on the use of the call

Each parameter is described as an input or output parameter. “Input” refers to input to IMS from the application program. “Output” refers to output from IMS to the application program.

The syntax diagrams for the following calls do not contain the complete call structure. Instead, the calls begin with the *function* parameter. The call, the call interface (xxxTDLI), and *parmcount* (if it is required) are not included in the following syntax diagrams. See language-specific information (for COBOL, C language, Pascal, PL/I, and assembler language) in “Chapter 2. Defining Application Program Elements” on page 31 for the complete structure.

In this Chapter:

- “AUTH Call”
- “CHNG Call” on page 64
- “CMD Call” on page 72
- “GCMD Call” on page 74
- “GN Call” on page 75
- “GU Call” on page 76
- “ISRT Call” on page 78
- “PURG Call” on page 80
- “SETO Call” on page 82

Related Reading: The DL/I calls used for database management are described in *IMS/ESA Application Programming: Database Manager*. EXEC DL/I commands used in CICS are described in *IMS/ESA Application Programming: EXEC DLI Commands for CICS and IMS*. DCCTL users can issue calls using GSAM database PCBs, which are described in *IMS/ESA Application Programming: Database Manager*.

AUTH Call

An Authorization (AUTH) call verifies each user’s security authorization. It determines whether a user is authorized to access the resources specified on the AUTH call.

Format

© Copyright IBM Corp. 1974, 2000

TM Message Call: AUTH

Call Name	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
AUTH	X		X		

Parameters

i/o pcb
Specifies the I/O PCB to use for this call. This parameter is an input and output parameter.

aib
Specifies the application interface block (AIB) that is used for the call. This parameter is an input and output parameter.

The following fields must be initialized in the AIB:

AIBID
Eyecatcher. This 8-byte field must contain DFSAIB␣␣.

AIBLEN
AIB lengths. This field must contain the actual length of the AIB that the application program obtained.

AIBRSNM1
Resource name. This 8-byte, left-justified field must contain the PCB name IOPCB␣␣␣␣.

AIBOALEN
I/O area length. This field must contain the length of the I/O area that is specified in the call list.

i/o area
Specifies the I/O area used for the call. This parameter is an input and output parameter.

I/O Area

Figure 14 on page 61 shows the format of the parameter list in the I/O area before the AUTH call is issued. Figure 15 on page 62 shows the I/O area after the AUTH call.

I/O area before the AUTH call

For the AIBTDLI, ASMTDLI, CBLTDLI, CEETDLI, CTDLI and PASTDLI interfaces:

LL	ZZ	CLASSNAME	RESOURCE	USERDATA
2	2	8	8	8

For the PLITDLI interface:

LLLL	ZZ	CLASSNAME	RESOURCE	USERDATA
4	2	8	8	8

Figure 14. I/O Area before the AUTH Call is issued

LL or LLLL
specifies a 2-byte field that contains the length of the parameter list, including two bytes for LL. For the PLITDLI interface, use the 4-byte field LLLL. However, if you use the AIBTDLI interface, PL/I programs require only a 2-byte field.

ZZ specifies a 2-byte field that contains binary zeros.

CLASSNAME
specifies an 8-byte field that contains one of the following values:

TRAN␣␣␣␣
DATABASE
SEGMENT␣
FIELD␣␣␣
OTHER␣␣␣

All parameters are 8 bytes in length, left-justified, and must be padded to the right with blanks.

The use of a generic class name in the call parameter list eliminates the need for the application to be sensitive to the actual Resource Access Control Facility (RACF) class names being used. Since transaction authorization must be active, only the RACF class associated with the generic class name identifier for the transaction class must be defined. The generic class name in the call parameter list causes the authorization function to select the proper RACF class and request access checking for that class.

RESOURCE
specifies the 8-byte field that contains the name of the resource to be checked. Except for the generic class TRAN, the resource name can be whatever the application designates because the name has no meaning for IMS TM.

IMS TM performs no validity checking of the resource name.

USERDATA
specifies the 8-byte keyword constant USERDATA is the only value supported.

Its presence in the parameter list means that the application program wants any RACF installation data that exists in the RACF Accessor Environment Element (ACEE).

I/O area after the AUTH call

For the AIBTDLI, ASMTDLI, CBLTDLI, CEETDLI, CTDLI and PASTDLI interfaces:

LL	ZZ	FEEDBACK	EXITRC	STATUS	RESERVED	UL	USERDATA
2	2	2	2	2	16	2	Variable

For the PLITDLI interface:

LLLL	ZZ	FEEDBACK	EXITRC	STATUS	RESERVED	UL	USERDATA
4	2	2	2	2	16	2	Variable

Figure 15. I/O Area after the AUTH Call is issued

LL or LLLL
A 2-byte field that contains the length of the character string, plus 2 bytes for LL. For the PLITDLI interface, use the 4-byte field LLLL. However, if you use the AIBTDLI interface, PL/I programs require only a 2-byte field.

ZZ specifies a 2-byte field that contains binary zeros.

FEEDBACK
specifies a 2-byte field that contains one of the following RACF return codes:

0000 User is authorized.

0004 Resource or class not defined.

0008 User is not authorized.

000C RACF is not active.

0010 Invalid installation exit return code.

EXITRC
specifies a 2-byte field that contains the return code from the user exits if they were used. The EXITRC field contains the return code from the last user exit that was entered. If none of the user exits are present or invoked, the field contains binary zeros. If installation data is returned from the exit, the EXITRC field is set to zero to indicate an authorized return code from the exit.

STATUS
specifies a 2-byte field that contains the hexadecimal status code indicating installation data status:

0000 RACF installation data is present in the I/O area.

0004 Security exit installation data present in then I/O area.

0008 User is not currently signed on.

000C User is not authorized, so installation data is not made available, or user is authorized, but no installation data has been defined.

0010 User was authorized, but installation data was not requested.

0014 USERDATA exceeds PSBWORK area length.

0018 RACF not active and TRN=N defined.

RESERVED
Binary zeros (reserved)

UL
specifies a 2-byte field that specifies the length of the installation data, including the length of the UL parameter.

USERDATA
specifies a variable-length field that contains installation data from ACEE or a user security exit. The length of the installation data is limited to 1026 bytes, including the length (UL) field. If a security exit returns a value greater than 1026, IMS truncates the installation data and adjusts the length field to represent the amount of installation data actually returned to the application program. If security exit installation data is returned, IMS passes it to the application program even if the parameter list did not contain the USERDATA parameter.

Any available installation data is returned if the return code from RACF indicates that the user is authorized to the resource named in the call parameter list. No installation data is returned if the user who originated the transaction is no longer signed on to the terminal associated with the transaction. Installation data might or might not be provided by the security exits when they are involved in the security decision. However, when either of the exits returns installation data, IMS passes it on to the application program.

If provided, installation data is returned from a security exit to the application even when the call parameter list does not specify the USERDATA parameter. In that case, the STATUS field of the I/O area contains the code X'0004' indicating the presence of the installation data.

Usage

The AUTH call determines whether a user is authorized to access the resources specified on the AUTH call. AUTH is issued with an I/O PCB and its function depends on the application program. Authorization checking depends on the dependent region type and whether a GU call has been issued. The call functions are as follows:

- In BMPs, AUTH uses the user ID of the IMS control region or installation specific user exits to determine the status of the call.
- For BMPs that have issued a successful GU call to the I/O PCB, AUTH functions as it does in an MPP.
- In MPPs, AUTH verifies user authorization with RACF for the specified resource classes of those resources used by the application program.

Because the call can request RACF user data to be passed back in the I/O area as installation data, the processing of the call always results in changes to the STATUS field in the I/O area. This STATUS field notifies the application of the status of installation data in the I/O area: available or not available. It might not be available because the installation data is not defined or the originating user is no longer signed on to the IMS system.

Either of the supported security exits for transaction authorization (DFSCTRN0 or DFSCTSE0) can present installation data upon return to IMS. If an exit returns installation data, the data is returned to the application even if the parameter list did

not contain the USERDATA parameter. The STATUS field is set to indicate the origination of the installation data. The STATUS field indicates the presence of either RACF installation data or security exit installation data.

The application program also receives notification of the actual RACF return code. This return code, presented as FEEDBACK in the I/O area, can be used by the application program to detect inconsistent operational modes and take alternate action. Examples of inconsistent operational modes are the proper RACF classes not being defined or the requested resource not properly defined to RACF.

By checking the FEEDBACK, EXITRC, and STATUS in the I/O area, the application program can be sensitive to issues such as the proper RACF definitions and resources not being defined. If RACF is being used, and the AUTH call references any resources that are not defined, the PCB status code is set to blanks and the FEEDBACK field of the I/O area is set to indicate that the resource is not protected.

Because the value for EXITRC is provided by a user security exit, use of this field must be made with an understanding of exit operation and the knowledge that any changes to the exit can result in application errors. If due to operational errors, the proper resources are not protected, the application can deal with the error in any way. This feedback can make operational control simpler and give the application more flexibility.

Related Reading: RACF terms and concepts are discussed in more detail in other books. For additional information, see *IMS/ESA Administration Guide: System* and *IMS/ESA Customization Guide*.

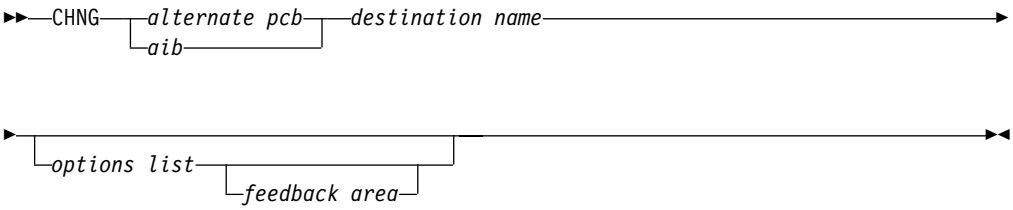
Restrictions

The AUTH call must not be issued before a successful GU call to the I/O PCB.

CHNG Call

The Change (CHNG) call sets the destination of a modifiable alternate PCB to the logical terminal, LU 6.2 descriptor, or transaction code that you specify. You can also use the CHNG call with the Spool Application Program Interface (Spool API) to specify print data set characteristics.

Format



Call Name	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
CHNG	X		X		

Parameters

alternate pcb
Specifies the modifiable alternate PCB to use for this call. This parameter is an input and output parameter.

aib
Specifies the application interface block (AIB) that is used for the call. This parameter is an input and output parameter.

The following fields must be initialized in the AIB:

AIBID
Eyecatcher. This 8-byte field must contain DFSAIBbb.

AIBLEN
AIB lengths. This field must contain the actual length of the AIB that the application program obtained.

AIBRSNM1
Resource name. This 8-byte, left-justified field must contain the name of a modifiable alternate PCB.

AIBOALEN
I/O area length. This field must contain the length of the I/O area that is specified in the call list.

destination name
Specifies an 8-byte field containing the destination name (the logical terminal or transaction code) to which you want messages sent. This parameter is an input parameter. The destination name can be up to 8 bytes. When you specify LU 6.2 options, IMS TM sets the destination name in the alternate PCB to DFSLU62b. If an LU 6.2 options list is specified the destination name parameter is ignored.

For more information on LU 6.2, see *IMS/ESA Administration Guide: Transaction Manager*.

Restriction: Some destination names are invalid. For more information on resource naming rules, see *IMS/ESA Installation Volume 2: System Definition and Tailoring*.

options list
Specifies one of several option keywords. This parameter is an input parameter. The options in the list are separated by commas and cannot contain embedded blanks. Processing for the options list terminates when the first blank in the list is reached or when the specified options list length has been processed. You can specify options for advanced print functions or for APPC (see “Advanced Print Function Options” on page 67 and “APPC Options” on page 69).

For more information on APPC, see *IMS/ESA Administration Guide: Transaction Manager*.

The format for the options list is as follows:

LL or LLLL [1, 2, 3]	**ZZ**	**keyword1=***variable1*
Halfword length of the options string, including the 4-byte length of LLZZ or LLLLZZ.	Halfword of zero.	CHNG options separated by commas.

LL or LLLL [1, 2, 3]	ZZ	keyword1=*variable1*

Notes:

1. For application programs that use the PLITDLI interface, the length field is a fullword (LLLL). However, the length of the LLLLZZ field is still considered four bytes.
2. If the length field is set to zero, the options list is ignored. IMS TM processes the CHNG call as if the *options list* parameter was not specified.
3. A keyword must be separated from the following variable by an equal sign (=). A keyword with no variable must be delimited by a comma or blank.

feedback area

Specifies an optional parameter used to return error information about the options list to the application program. This parameter is an output parameter. The amount of information that the application program receives is based on the size of the feedback area. If no feedback area is specified, the status code returned is the only indication of an options list error. If you specify a feedback area 1½ to 2 times the size of the specified options list (a minimum of eight words), IMS TM returns more specific information about errors in the options list.

The format for the feedback area passed to IMS in the call list is as follows:

LL or LLLL [1, 2]	ZZ
Halfword length of the feedback area, including the 4-byte length of the LLZZ fields.	Halfword of zero.

Notes:

1. For application programs that use the PLITDLI interface, the length field is a fullword (LLLL). However, the length of the LLLLZZ field is still considered 4 bytes.
2. If the length field is set to zero, the feedback area is ignored. IMS TM processes the CHNG call as if the *feedback area* parameter was not specified.

The output format returned to the application program from IMS for the feedback area is as follows:

LLZZ or LLLLZZ	LL	*feedback data*
The length field as specified in the input format for the feedback area.	Halfword length of the feedback data returned by IMS TM, including the 2-byte **LL** field.	Data returned by IMS TM. The feedback data generally includes the option keyword found to be in error and a 4-byte EBCDIC code in parentheses that indicates the reason for the error. Multiple errors are separated by commas.

Usage

Use the CHNG call to send an output message to an alternate destination in your system or in another system. When you issue the CHNG call, you supply the name of the destination to which you want to send the message. The alternate PCB you name then remains set to that destination until you do one of the following:

- Issue another CHNG call to reset the destination.

- Issue a Get Unique (GU) call to the message queue to start processing a new message. In this case, the name of the PCB you specify with the CHNG call still appears in the alternate PCB, even though it is no longer valid.
- Terminate the application program. When you terminate the application, IMS TM resets the destination to blanks.

For more information on sending messages to alternate terminals, see “ISRT Call” on page 78 and “PURG Call” on page 80.

You can use the CHNG call to perform Spool API functions.

For Spool API functions, each CHNG call to a nonexpress, alternate PCB, creates a separate JES spool data set. (PURG calls have no effect when issued against a nonexpress, alternate PCB.) If the destination of the PCB is the JES spool, it cannot be CHNGed to a non-JES spool destination until the data set(s) have been released by a sync point. Keywords that can be specified on the CHNG call are discussed in “Advanced Print Function Options” and “APPC Options” on page 69.

In the OTMA environment

If an IMS application program issues a CHNG call to an alternate PCB and specifies an options list, then the output destination cannot be an IMS Open Transaction Manager client.

An IMS application program that issues a CHNG call to an alternate PCB (specifying an options list) does not cause IMS to call the OTMA Prerouting and Destination Resolution exit routines to determine the destination. But an IMS application program that issues a CHNG call to an alternate PCB (specifying an APPC descriptor) does cause IMS to call the OTMA exit routines to determine the destination. For information on these exit routines, see *IMS/ESA Customization Guide*.

The application program can still issue ISRT calls to the I/O PCB to send data to an OTMA destination.

OTMA application programs can use CHNG and ISRT calls for APPC destinations. For more information, see *IMS/ESA Application Programming: Design Guide*.

Advanced Print Function Options

The IAFP keyword identifies the CHNG call as a request for Spool API functions. The parameters of the IAFP keyword are:

Keyword	Description
IAFP=*abc*	a — specifies carriage control options
	b — specifies integrity options
	c — specifies message processing options

The following options specify advanced print functions for the CHNG call.

Carriage Control Options: The 1-character carriage control options indicate the type of carriage control that is present in the message data when the ISRT or PURG call is issued. Your application program must insert the proper carriage control characters in the data stream. You can specify one of the following values for the IAFP keyword:

A The data stream contains ASA carriage control characters.

M The data stream contains machine carriage control characters.

N The data stream does not contain carriage control characters.

Integrity Options: The 1-character integrity options indicate the method IMS TM uses in allocating the IMS Spool data set that contains the IAFP message. You can specify one of the following options for the IAFP keyword:

0 IMS TM attempts no data set protection. Your application program must provide any disposition or hold status by using the appropriate OUTPUT descriptor options. IMS TM does attempt to prevent a partial message from printing and to deallocate data sets that contain messages that have already reached a sync point. To control whether error messages about the IMS Spool data set are issued, use the message processing options for the IAFP keyword.

1 The IMS Spool data set is placed on the SYSOUT HOLD queue when it is allocated. If IMS TM issues message DFS00121 or DFS00141, the operator must query the SYSOUT HOLD queue to locate the appropriate data sets. IMS TM releases the data set and deallocates it to be printed at syncpoint.

When you specify 1 for the integrity option, you must specify *M* for the message processing option of the IAFP keyword.

2 A remote destination is specified in the *destination name* parameter on the CHNG call. The IMS Spool data set, when allocated, is placed on a SYSOUT remote workstation, IMSTEMP. This destination must be included in the definitions as nonselectable so that the data set is not automatically selected to be printed. If IMS TM issues message DFS00121 or DFS00141, the operator must query IMSTEMP to locate the appropriate data sets. At sync point, IMS TM releases the data set and deallocates it to the remote workstation ID specified in the *destination name* parameter. The value 2 overrides any destination specified in the IAFP OUTPUT options.

Message Processing Options: The 1-character message processing options indicate whether IMS TM issues message DFS00141 during restart and message DFS00121 for dynamic allocation failures. You can specify one of the following options:

0 DFS00121 and DFS00141 are not issued. Your application program controls IAFP message integrity.

M DFS00121 and DFS00141 are issued if necessary. IMS TM controls IAFP message integrity.

The CHNG call can provide the data set characteristics in the following ways:

- Directly, using the PRTO= option
- Referencing prebuilt text units, using the TXTU= option
- Referencing an OUTPUT JCL statement in the dependent region's JCL, using the OUTN= option

When you use the IAFP keyword, you must also specify the PRTO, TXTU, or OUTN option. (The options PRTO, TXTU, and OUTN are mutually exclusive.) If you do not specify one of these additional options, or if you specify more than one of these options, or if you specify IAFP with an invalid value, IMS TM returns an AR status code to your application program.

Keyword	Description
PRTO=*outdes options*	Describes the data set processing options as they are specified on the TSO OUTDES statement.

The format for the PRTO= keyword is as follows:

LL	*outdes options*
Halfword length of the total OUTDES printer options, including the 2-byte length of **LL**.	Any valid combination of OUTDES printer options.

Note: For information on TSO OUTDES options, see *MVS/ESA Application Development Guide: Authorized Assembler Language Programs*. Some options depend on the release level of MVS/ESA.

TXTU=*address*

specifies the address of a list of text-unit pointers. The list (with the associated text units) can be created by a previous SETO call, or it can be created by your application program. The LLZZ or LLLLZZ prefix must be included on the buffer that contains the list. TXTU allows your application program to issue a SETO call to build the text units for the OUTDES options before the CHNG call is issued.

If your application program issues several CHNG calls with the same OUTDES printer options, the TXTU option means you do not need to build OUTDES options for each CHNG call.

OUTN=*name*

specifies a character string up to eight characters long that contains the name of an OUTPUT JCL statement that identifies the printer processing options to be used. If the specified OUTPUT DD statement is not included in the JCL for the region in which the application program runs, a dynamic allocation error occurs when the application attempts to insert data to the data set.

APPC Options

The following APPC options are available for the CHNG call:

Keyword **Description**

LU=*logical unit name*

Specifies the logical unit (LU) name of a partner for an LU 6.2 conversation with a partner application program. It is used in conjunction with the MODE and TPN options to establish the conversation. The LU name can be any alphanumeric string including national characters, but the first character cannot be a number. If the LU name is a network-qualified name, it can be up to seventeen characters long and consist of the network ID of the originating system, followed by '.', then the LU name. (for example, netwrkid.luname). The LU name and the network ID are both one to eight characters long. The default for this option is DFSLU.

Related Reading: For more information on LU 6.2, see *IMS/ESA Administration Guide: Transaction Manager*.

MODE=*mode name*

Specifies the mode of the partner for an LU 6.2 conversation with a partner application program. It is used in conjunction with the LU and TPN options to establish the conversation. The mode name can be any alphanumeric string up to eight characters long, including national characters, but the first character cannot be a number. If both MODE and SIDE options are specified, the mode name specified in the SIDE entry is ignored but is not changed. The default for this option is DFSMODE.

Related Reading: For more information on LU 6.2, see *IMS/ESA Administration Guide: Transaction Manager*.

TPN=*transaction program name*
Specifies the transaction program (TP) name of the partner application program in an LU 6.2 conversation. The option is used in conjunction with the LU and MODE keywords to establish the conversation.

Related Reading: For more information on LU 6.2, see *IMS/ESA Administration Guide: Transaction Manager*.

TP names can be up to 64 characters long and can contain any character from the 00640 character set except a blank. The 00640 character set includes the letters A-Z, the digits 0-9, and 20 special characters. The default for this option is DFSASYNC. For more information on the 00640 character set, see *Common Programming Interface Communications Reference*. The format for the TPN option is as follows:

LL	*tpn*
Halfword length of the TP name, including the 2-byte length of **LL**.	The TP name, which can be up to 64 characters long.

TP names that are processed with the IMS command processor must contain characters that are valid to IMS. For example, names that contain lower case letters cannot be processed and are rejected if they are used as operands for IMS commands.

SIDE=*side information entry name*
Specifies the side information entry name that can be used to establish an LU 6.2 conversation with a partner application program. For more information on LU 6.2, see *IMS/ESA Administration Guide: Transaction Manager*. The SIDE name can contain up to eight characters, including the uppercase alphabet (A-Z), and the digits 0-9. If the LU, MODE, or TPN keywords are specified, they override the SIDE keyword, but they do not change the side information entry name. This option has no default.

SYNC=N|C Overrides the APPC/IMS conversation synchronization level. N sets the synchronization level to NONE. C sets the synchronization level to CONFIRM. The default for this option is C.

TYPE=B|M Overrides the APPC/IMS conversation type. B sets the conversation type to BASIC. M sets the conversation type to MAPPED. The default for this option is M.

Related Reading: For more information on APPC and the default options, see *IMS/ESA Administration Guide: Transaction Manager*.

Options List Feedback Area

When errors are encountered in the options list, the options list feedback area is used to return error information to the application.

IMS attempts to parse the entire options list and return information on as many errors as possible. If the feedback area is not large enough to contain all the error information, only as much information is returned as space permits. The status code is the only indication of an option list error if you do not specify the area.

The feedback area must be initialized by the application with a length field indicating the length of the area. A feedback area approximately 1½ to 2 times the length of the options list or a minimum of 8 words should be sufficient.

Error Codes

This section contains information on error codes that your application can receive.

Error Code	Reason
(0002)	Unrecognized option keyword. Possible reasons for this error are: • The keyword is misspelled. • The keyword is spelled correctly but is followed by an invalid delimiter. • The length specified field representing the PRTO is shorter than the actual length of the options. • A keyword is not valid for the indicated call.
(0004)	Either too few or too many characters were specified in the option variable. An option variable following a keyword in the options list for the call is not within the length limits for the option.
(0006)	The length field (LL) in the option variable is too large to be contained in the options list. The options list length field (LL) indicates that the options list ends before the end of the specified option variable.
(0008)	The option variable contains an invalid character or does not begin with an alphabetic character.
(000A)	A required option keyword was not specified. Possible reasons for this error are: • One or more additional keywords are required because one or more keywords were specified in the options list. • The specified length of the options list is more than zero but the list does not contain any options.
(000C)	The specified combination of option keywords is invalid. Possible causes for this error are: • The keyword is not allowed because of other keywords specified in the options list. • The option keyword is specified more than once.
(000E)	IMS found an error in one or more operands while it was parsing the print data set descriptors. IMS usually uses MVS/ESA services (SJF) to validate the print descriptors (PRTO= option variable). When IMS calls SJF, it requests the same validation as for the TSO OUTDES command. Therefore, IMS is insensitive to changes in output descriptors. Valid descriptors for your system are a function of the MVS/ESA release level. For a list of valid descriptors and proper syntax, use the TSO HELP OUTDES command. IMS must first establish that the format of the PRTO options is in a format that allows the use of SJF services. If it is not, IMS returns the status code **AS**, the error code (000E), and a descriptive error message. If the error is detected during the SJF process, the error

message from SJF will include information of the form (R.C.=xxxx,REAS.=yyyyyyyy), and an error message indicating the error.

Related Reading: For more information on SJF return and reason codes, see *MVS/ESA Application Development Guide: Authorized Assembler Language Programs*.

The range of some variables is controlled by the initialization parameters. Values for the maximum number of copies, allowable remote destination, classes, and form names are examples of variables influenced by the initialization parameters.

Restrictions

Before you can use the CHNG call to set or alter the destination of an alternate PCB, you must issue the PURG call to indicate to IMS that the message that you have been building with that PCB is finished.

LU 6.2 architecture prohibits the use of the ALTRESP PCB on a CHNG call in an LU 6.2 conversation. The LU 6.2 conversation can only be associated with the IOPCB. The application sends a message on the existing LU 6.2 conversation (synchronous) or has IMS create a new conversation (asynchronous) using the IOPCB. Since there is no LTERM associated with an LU 6.2 conversation, only the IOPCB represents the original LU 6.2 conversation.

For Spool API functions, each CHNG call to a nonexpress, alternate PCB, creates a separate JES spool data set. (PURG calls have no effect when issued against a nonexpress, alternate PCB.) If the destination of the PCB is the JES spool, it cannot be CHNGed to a non-JES spool destination until the data set(s) have been released by a sync point.

CMD Call

The Command (CMD) call enables an application program to issue IMS commands.

Format

Call Name	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
CMD	X		X		

Parameters

i/o pcb
Specifies the I/O PCB to use for this call. This parameter is an input and output parameter.

aib
Specifies the application interface block (AIB) that is used for the call. This parameter is an input and output parameter.

The following fields must be initialized in the AIB:

AIBID
Eyecatcher. This 8-byte field must contain DFSAIBbb.

AIBLEN
AIB lengths. This field must contain the actual length of the AIB that the application program obtained.

AIBRSNM1
Resource name. This 8-byte, left-justified field must contain the PCB name IOPCBbbb.

AIBOALEN
I/O area length. This field must contain the length of the I/O area that is specified in the call list.

i/o area
Specifies the I/O area to use for this call. This parameter is an input and output parameter. The I/O area must be large enough to hold the largest segment passed between the program and IMS TM.

Usage

Use the CMD call with the GCMD call to send commands to and receive responses from IMS TM. After the CMD call issues the command to IMS TM, IMS TM processes the command and returns the first segment of the response message to the application program's I/O area, but only if a CC status code is returned on the CMD call. Your application program must then issue GCMD calls to retrieve all subsequent message segments one segment at a time. For more information, see "GCMD Call" on page 74. The CMD and GCMD command calls are typically used to perform functions that are usually handled by someone at a terminal. These programs are called automated operator (AO) applications.

Related Reading: For more information on the automated operator interface (AOI), see *IMS/ESA Customization Guide*.

Before you issue a CMD call, the IMS command that you want to execute must be in the I/O area that you refer to in the call. When you issue a CMD call, IMS TM passes the command from the I/O area to the IMS control region for processing. IMS TM places your application program in a wait state until the command is processed. The application program remains in a wait state until IMS TM returns a response. (*Response* means that IMS TM has received and processed the command.) For asynchronous commands, you receive a response when the command is processing, but not when it is complete.

You can also issue DB2 commands from your IMS TM application program. Issue the command call and use the /SSR command, followed by the DB2 command. IMS TM routes the command to DB2. DB2 issues a response to the command, and IMS TM routes the DB2 response to the master terminal operator (MTO).

Restrictions

The AIB must specify the I/O PCB for this call.

Any application program that uses this call must be authorized by the security administrator.

You cannot issue a CMD call from a CPI-C driven application program.

This call is not supported in an IFP or non-message-driven BMP.

GCMD Call

The Get Command (GCMD) call retrieves the response segments from IMS TM when your application program processes IMS commands using the CMD call.

Format

Call Name	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
GCMD	X		X		

Parameters

i/o pcb
Specifies the I/O PCB to use for this call. This parameter is an input and output parameter.

aib
Specifies the application interface block (AIB) that is used for the call. This parameter is an input and output parameter.

The following fields must be initialized in the AIB:

AIBID
Eyecatcher. This 8-byte field must contain DFSAIBbb.

AIBLEN
AIB lengths. This field must contain the actual length of the AIB that the application program obtained.

AIBRSNM1
Resource name. This 8-byte, left-justified field must contain the PCB name IOPCBbbb.

AIBOALEN
I/O area length. This field must contain the length of the I/O area that is specified in the call list.

i/o area
Specifies the I/O area to use for this call. This parameter is an output parameter. The I/O area must be large enough to hold the largest segment passed between the program and IMS TM.

Usage

When you issue a CMD call (see “CMD Call” on page 72), IMS TM returns the first command response segment to the application program's I/O area. If you are processing commands that return more than one command response segment, use the GCMD call to retrieve the second and subsequent command response segments. IMS TM returns one command response segment to the I/O area of your application program each time the application program issues a GCMD call. The I/O area must be large enough to hold the longest message segment expected by your application program. IMS allows a maximum segment size of 132 bytes (including the 4-byte LLZZ field).

The CMD and GCMD calls are typically used to perform functions that are usually performed by someone at a terminal. These programs are called automated operator (AO) applications.

Related Reading: For more information on the automated operator (AO) interface, see *IMS/ESA Customization Guide*.

PCB status codes indicate the results of a GCMD call. The status codes are similar to those that result from a message GN call. A QD status indicates that there are no more segments in the response. A QE status indicates that a GCMD call was issued after a CMD call that did not produce response segments. A blank status ('bb') indicates that a segment was retrieved successfully.

Restrictions

The AIB must specify the I/O PCB for this call.

Any AO application that uses this call must be authorized by the security administrator.

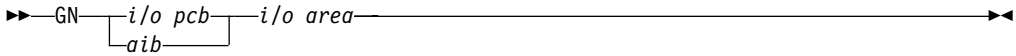
You cannot issue a GCMD call from a CPI-C driven application program.

This call is not supported in an IFP, or non-message driven BMP.

GN Call

If an input message contains more than one segment, a Get Unique (GU) call retrieves the first segment of the message and Get Next (GN) calls retrieve the remaining segments (see “GU Call” on page 76).

Format



Call Name	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
GN	X		X		

Parameters

i/o pcb
Specifies the I/O PCB to use for this call. This parameter is an input and output parameter.

aib
Specifies the application interface block (AIB) that is used for the call. This parameter is an input and output parameter.

The following fields must be initialized in the AIB:

AIBID
Eyecatcher. This 8-byte field must contain DFSAIBbb.

AIBLEN
AIB lengths. This field must contain the actual length of the AIB that the application program obtained.

AIBRSNM1
Resource name. This 8-byte, left-justified field must contain the PCB name IOPCBbbb.

AIBOALEN
I/O area length. This field must contain the length of the I/O area that is specified in the call list.

i/o area
Specifies the I/O area to use for this call. This parameter is an output parameter. The I/O area must be large enough to hold the largest segment passed between the program and IMS TM.

Usage

If you are processing messages that contain more than one segment, you use the GN call to retrieve the second and subsequent segments of the message. IMS TM returns one message segment to the I/O area of your application program each time the application program issues a GN call.

You can issue a GN call from a BMP program.

Restrictions

The AIB must specify the I/O PCB for this call.

You cannot issue a GN call from a CPI-C driven application program.

GU Call

The Get Unique (GU) call retrieves the first segment of a message.

Format

Call Name	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
GU	X		X		

Parameters

i/o pcb
Specifies the I/O PCB to use for this call. This parameter is an input and output parameter.

aib
Specifies the application interface block (AIB) that is used for the call. This parameter is an input and output parameter.

The following fields must be initialized in the AIB:

AIBID
Eyecatcher. This 8-byte field must contain DFSAIBbb.

AIBLEN
AIB lengths. This field must contain the actual length of the AIB that the application program obtained.

AIBRSNM1
Resource name. This 8-byte, left-justified field must contain the PCB name IOPCBbbb.

AIBOALEN
I/O area length. This field must contain the length of the I/O area that is specified in the call list.

i/o area
Specifies the I/O area to use for this call. This parameter is an output parameter. The I/O area must be large enough to hold the largest segment passed between the program and IMS TM.

Usage

An MPP or message-driven BMP uses two calls to retrieve input message from the host: GN and GU. A GU call retrieves the first segment of a message. The Get Next (GN) call retrieves subsequent segments. (See “GN Call” on page 75.)

When you issue a successful GU or GN, IMS TM returns the message segment to the I/O area that you specify in the call. Message segments are not all the same length. Because the segment length varies, your I/O area must be long enough to hold the longest segment that your program can receive. The first two bytes of the segment contain the length of the segment.

Your application program must issue a GU call to the message queue before issuing other DL/I calls. When IMS TM schedules an MPP, the Transaction Manager transfers the first segment of the first message to the message processing region. When the MPP issues the GU for the first message, IMS TM already has the message waiting. If the application program does not issue a GU message call as the first call of the program, IMS TM has to transfer the message again, and the efficiency provided by message priming is lost.

If an MPP responds to more than one transaction code, the MPP has to examine the text of the input message to determine what processing the message requires.

After a successful GU call, IMS TM places the following information in the I/O PCB mask:

- The name of the logical terminal that sent the message.
- The status code for this call. (See “System Service Call Summary” on page 440)
- The input prefix, giving the date, time, and sequence number of the message at the time it was first queued. IMS returns both an 8-byte local date containing a 2-digit year and a 12-byte timestamp (local or UTC time) containing a 4-digit year.
- The MOD name (if you are using MFS).
- The user ID of the person at the terminal, or if user IDs are not used in the system, the logical terminal name. If the message is from a BMP, IMS TM places the PSB name of the BMP in this field.
- Group name, used by DB2 to provide security for SQL calls.

Related Reading: For more information on the format of the I/O PCB mask, see "Specifying the I/O PCB Mask" on page 45.

Restrictions

The AIB must specify the I/O PCB for this call.

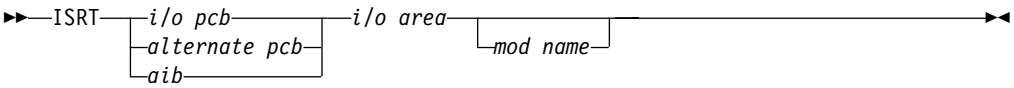
You cannot issue a GU call from a CPI-C driven application program.

ISRT Call

The Insert (ISRT) call sends one message segment to the destination that you specify in the call. The destination is represented by the I/O PCB, alternate PCB, or AIB you specify in the call parameters.

For Spool API functions, the ISRT call is also used to write data to the JES Spool.

Format



Call Name	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
ISRT	X		X		

Parameters

i/o pcb

alternate pcb
Specifies the PCB to use for this call. These parameters are input and output parameters.

aib
Specifies the application interface block (AIB) that is used for the call. This parameter is an input and output parameter.

The following fields must be initialized in the AIB:

AIBID
Eyecatcher. This 8-byte field must contain DFSAIB␣␣.

AIBLEN
AIB lengths. This field must contain the actual length of the AIB that the application program obtained.

AIBRSNM1
Resource name. This 8-byte, left-justified field must contain the PCB name IOPCB␣␣␣ (if the I/O PCB is used), or the name of an alternate PCB (if an alternate PCB is used).

AIBOALEN
I/O area length. This field must contain the length of the I/O area that is specified in the call list.

i/o area
Specifies the I/O area to be used for the call. This parameter is an input parameter. The I/O area must be large enough to hold the largest segment passed between the application program and IMS TM.

mod name
Specifies the MOD you want used for this output message. This parameter is an input parameter. The 8-byte MOD name must be left-justified and padded with blanks as necessary. If the terminal receiving the output does not use MFS, this parameter is ignored. If you specify a valid MOD name, IMS TM uses that MOD to format the screen for the output message you are sending.

Usage

To issue the ISRT call successfully, your application program must first build the message you want to send in the application program's I/O area. The ISRT uses the destination name in the I/O PCB or alternate PCB, and the I/O area that you specify in the call, to locate the message to be sent. The ISRT call then sends the output message from your application program to another terminal. ISRT sends one message segment per issue, so your application program must issue one ISRT call for each segment of the message in the I/O area.

You can also specify a MOD name if you want to change the screen format. For example, if the application program detects an error and must notify the person at the terminal, you can specify a MOD name that formats the screen to receive the error message. ISRT and PURG are the only DL/I calls that allow you to specify a MOD name on the first segment of an output message.

When your application program issues one or more ISRT calls, IMS TM groups the message segments to be sent in the message queue. IMS TM sends the message segments to the destination when the application program does one of the following:

- Issues a GU call to retrieve the first segment of the next message
- Reaches a commit point
- Issues a PURG call on an express alternate PCB

Your application must also use the ISRT call to issue replies to other terminals in conversational programs and to pass a conversation between application programs.

Related Reading: For more information on ISRT in conversational programs see "Sending Messages to Other Terminals and Programs" on page 123 and "Passing the Conversation to another Conversational Program" on page 137.

Spool API Functions

You can use the ISRT call to write data to the JES Spool. These writes are done using BSAM and, if possible, each BSAM "write" is done directly from the application program's buffer area.

Restriction: BSAM does not support the I/O area for sysout data sets above the 16-MB line. If IMS/ESA finds an I/O area above the 16-MB line, it moves the application data to a work area below the line before it performs the BSAM write. If the I/O area is already below the line, the write is done directly from the I/O area. Do not take unusual steps to place the I/O area below the line unless performance indicates a need to do so.

TM Message Call: ISRT

When you issue the ISRT call for an alternate PCB set up for IAFP processing, prefix the I/O area with a BSAM block descriptor word for variable length records.

Related Reading: For more information on BSAM block descriptor words, see *MVS/ESA Data Administration Guide for Data Facility Product*.

LL or LLLL[1, 2]	ZZ[2]	ll[3]	zz[3]
Halfword length of the I/O area or block, including the 4-byte length of the LLZZ fields.	Halfword of zero	Halfword length of the logical record or segment, including the 4-byte length of the llzz fields.	Halfword of zero

Notes:

1. For application programs that use the PLITDLI interface, the length field is a fullword (LLLL). However, the length of the LLLLZZ field is still considered 4 bytes.
2. LLZZ is the equivalent of the BSAM Block Descriptor Word (BDW).
3. llzz is the equivalent of the BSAM Record Descriptor Word (RDW).

For more information on Spool API, see *IMS/ESA Application Programming: Design Guide*.

Restrictions

A CPI-C driven application program can only issue the ISRT call to an alternate PCB.

If you want to send message segments before retrieving the next message or issuing a commit point, you must use the PURG call. For a description of the PURG call, see “PURG Call”.

MOD name can be specified only once per message, on the first ISRT or PURG call that begins the message.

BSAM does not support the I/O area for sysout data above the 16 MB line.

For more information on LU 6.2, see *IMS/ESA Administration Guide: Transaction Manager*.

PURG Call

The Purge (PURG) call allows your application program to send one or more output message segments (specified with the ISRT call) to the specified destination before the application program retrieves the next input message or issues a commit point.

For Spool API functions, the PURG call can also be used to release a print data set for immediate printing.

Parameters

i/o pcb

alternate pcb
Specifies the PCB to use for the call. These parameters are input and output parameters.

aib
Specifies the application interface block (AIB) that is used for the call. This parameter is an input and output parameter.

The following fields must be initialized in the AIB:

AIBID
Eyecatcher. This 8-byte field must contain DFSAIBbb.

AIBLEN
AIB lengths. This field must contain the actual length of the AIB that the application program obtained.

AIBRSNM1
Resource name. This 8-byte, left-justified field must contain the PCB name IOPCBbbb (if the I/O PCB is used), or the name of an alternate PCB (if an alternate PCB is used).

AIBOALEN
I/O area length. This field must contain the length of the I/O area that is specified in the call list.

i/o area
Specifies the I/O area to use for this call. This parameter is an input parameter. The I/O area must be large enough to hold the largest segment passed between the program and IMS TM.

mod name
Specifies the MOD you want used for this output message. This parameter is an input parameter. The 8-byte MOD name must be left justified and padded with blanks as necessary. PURG can specify the MOD name for the first message segment for an output message. If the terminal receiving the output does not use MFS, this parameter is ignored. If you specify a valid MOD name, IMS TM uses that MOD to format the screen for the output message you are sending.

Usage

Use the PURG call to send output messages to several different terminals. A PURG call tells IMS TM that the message built against the specified I/O PCB, or alternate PCB (with the ISRT call) is complete. IMS TM collects the message segments that have been inserted to one PCB as one message and sends the message to the destination specified by the destination name of the alternate PCB listed in the PURG call.

If you specify an I/O area in the PURG call parameters, PURG acts as an ISRT call to insert the first segment of the next message. When you identify the I/O area, you can also specify a MOD name to change the screen format.

Related Reading: For more information on sending messages to several terminals see “Sending Messages to Other Terminals and Programs” on page 123.

In the OTMA environment

An IMS application program that issues a PURG call causes IMS to call the Open Transaction Manager Access (OTMA) Prerouting and Destination Resolution exit routines to determine the destination. For information on these exit routines, see *IMS/ESA Customization Guide*.

Spool API Functions

You can use the PURG call with an express alternate PCB to release a print data set for immediate printing. When you issue the PURG call with an I/O area, IMS treats the call as two functions: the purge request, and the insertion of data provided by the I/O area.

If you issue the PURG call:

- Against an express alternate PCB, the data set is closed, unallocated, and released for printing. The destination is reset.
- With an I/O area against a nonexpress alternate PCB, the purge function is ignored and the data in the insert portion of the call is put into the print data set. This means that the call behaves like an ISRT call.
- With no I/O area against an express alternate PCB, the data set is closed, unallocated, and released for printing. IMS returns a status code of blanks.
- With no I/O area against a nonexpress alternate PCB, no action is taken.

Restrictions

CPI-C driven application programs can only issue the PURG call to alternate PCBs.

MOD name can be specified only once per message, in the first ISRT or PURG call that begins the message.

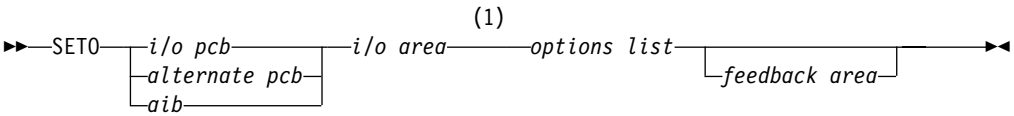
This call is not supported in an IFP.

For synchronized APPC/OTMA conversations, PURG calls on the I/O PCB are ignored. The next ISRT call is processed for the next segment of the current message.

SETO Call

The SET Options (SETO) call allows IMS application programs to set processing options. The SETO call can also be used to set processing options for Spool API functions.

Format



Notes:

1 The I/O area parameter is not used for calls that specify APPC options.

Call Name	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
SETO	X		X		

Parameters

i/o pcb

alternate pcb
Specifies the I/O or alternate PCB to be used for the call. These parameters are input and output parameters.

aib
Specifies the application interface block (AIB) that is used for the call. This parameter is an input and output parameter.

The following fields must be initialized in the AIB:

AIBID
Eyecatcher. This 8-byte field must contain DFSAIB␣␣.

AIBLEN
AIB lengths. This field must contain the actual length of the AIB that the application program obtained.

AIBRSNM1
Resource name. This 8-byte, left-justified field must contain the PCB name IOPCB␣␣␣ (if the I/O PCB is used), or the name of an alternate PCB (if an alternate PCB is used).

AIBOALEN
I/O area length. This field must contain the length of the I/O area that is specified in the call list.

i/o area
Specifies the I/O area to be used for the call. This parameter is an output parameter. If you specify an options list that contains advanced print functions, you must specify an I/O area. If you use APPC options, the I/O area parameter is optional.

For advanced print function options the I/O area must be at least 4 KB. If the I/O area including the LLZZ or LLLLZZ prefix is less than 4096 bytes in length, an AJ status code is returned. Once the text units area built in the I/O area, the area must not be copied to a new area. The I/O area passed on the SETO call must contain a LLZZ or, if PL/I, a LLLLZZ prefix.

LLLL applies only to DL/I call interface.

options list
Specifies several option keywords. This input parameter is required. The options in the list are separated by commas and cannot contain embedded blanks. Processing for the options list terminates when the first blank in the list is reached or when the specified options list length has been processed. You can specify options for advanced print functions or for APPC. The options you can specify are described in “Advanced Print Function Options” on page 85 and “APPC Options” on page 85.

The format for the options list is as follows:

LL or LLLL[1, 2]	**ZZ**	**keyword1=***variable1*
Halfword length of the options string, including the 4-byte length of LLZZ or LLLLZZ.	Halfword of zero.	SETO options separated by commas.

LL or LLLL[1, 2]	ZZ	keyword1=*variable1*

Note:

1. For application programs that use the PLITDLI interface, the length field is a fullword (LLLL). However, the length of the LLLLZZ field is still considered 4 bytes.
2. If the length field is set to zero, the options list is ignored. IMS TM processes the SETO call as if the *options list* parameter was not specified.

feedback area

Specifies an optional parameter used to return error information about the options list to the application program. This parameter is an output parameter. The amount of information that the application program receives is based on the size of the feedback area. If no feedback area is specified, the status code returned is the only indication of an options list area. If you specify a feedback area 1½ to 2 times the size of the specified options list (a minimum of eight words), IMS TM returns more specific information about errors in the options list.

The format for the feedback area passed to IMS TM in the call list is as follows:

LL or LLLL[1, 2]	ZZ
Halfword length of the feedback area, including the 4-byte length of the LLZZ fields.	Halfword of zero.

Note:

1. For application programs that use the PLITDLI interface, the length field is a fullword (LLLL). However, the length of the LLLLZZ field is still considered four bytes.
2. If the length field is set to zero, the feedback area is ignored. IMS TM processes the SETO call as if the *feedback area* parameter was not specified.

The output format returned to the application program from IMS TM for the feedback area is as follows:

LLZZ or LLLLZZ	LL	*feedback data*
The length field as specified in the input format for the feedback area.	Halfword length of the feedback data returned by IMS TM, including the 2-byte **LL** field.	Data returned by IMS TM. The feedback data generally includes the option keyword found to be in error and a 4-byte EBCDIC code in parentheses that indicates the reason for the error. Multiple errors are separated by commas.

Usage

The SETO call allows you to set processing options.

You can use the SETO call to reduce the overhead necessary to perform parsing and text construction of the OUTPUT descriptors for a data set. If your application program can use a set of descriptors more than once during an installation, the application can use the SETO call to provide print data set characteristics to the Spool API. When the SETO call is processed, it parses the OUTPUT options and constructs the dynamic OUTPUT text units in the work area provided by the application. After the application has received the prebuilt text units, you can use

the CHNG call and TXTU= option to provide the print characteristics for the data set without incurring the overhead of parsing and text unit build.

It is not necessary to use the SETO call to prebuild the text units if they can be prebuilt with another programming technique.

Keywords that can be specified on the SETO call are described in "Advanced Print Function Options" and "APPC Options".

Related Reading: For more information about Spool API, see *IMS/ESA Application Programming: Design Guide*.

In the OTMA environment

An IMS application program that issues a SETO call does not cause IMS to call the Open Transaction Manager Access (OTMA) Prerouting and Destination Resolution exit routines to determine the destination. For information on these exit routines, see *IMS/ESA Customization Guide*.

Existing IMS application programs that issue SETO calls might not run as expected because a return code is returned to the program if it is processing an OTMA-originated transaction. Also, APPC/IMS application programs that issue SETO calls might not need modification if they require implicit OTMA support.

A solution to this problem is to use an INQY call before issuing the SETO call. The application program can use the output from the INQY call to determine if a transaction is an OTMA-originated one, to bypass the SETO call.

Advanced Print Function Options

The PRTO= keyword identifies the SETO call as a Spool API request:

Keyword	Description
PRTO=*outdes options*	Describes the data set processing options as they are specified on the TSO OUTDES statement. The format for the **PRTO** keyword is as follows:

LL	*outdes options*
Halfword length of the total OUTDES printer options, including the 2-byte length of **LL**.	Any valid combination of OUTDES printer options, separated by commas.

Note: For information about TSO OUTDES options, see *MVS/ESA Application Development Guide: Authorized Assembler Language Programs*. Some options depend on the release level of MVS/ESA.

If MVS detects an error in the OUTDES printer options, an AS status code is returned to the application program.

APPC Options

The following options are available for the SETO call:

SEND_ERROR
causes the IMS LU Manager to issue SEND_ERROR on the conversation associated with the I/O or alternate PCB when a message is sent. Messages for express PCBs are sent during the PURG call or sync point processing, whichever comes first. Messages for nonexpress PCBs are sent during sync point processing.

This option is only used by LU 6.2 devices, and it is ignored if specified for a non-LU 6.2 device.

The option is mutually exclusive with the DEALLOCATE_ABEND option. If both options are coded in the options list, an AR status code is returned to the application.

DEALLOCATE_ABEND
deallocates a conversation by issuing a SEND_ERROR followed by a DEALLOCATE_ABEND at the time the message is sent. Once a `SETO` call with the DEALLOCATE_ABEND option is issued, any subsequent `ISRT` calls made to the PCB are rejected with a QH status code.

This option is applicable only to LU 6.2 devices. If specified for a non-LU 6.2 device, any subsequent `ISRT` calls made to the PCB are rejected with a QH status code.

When the `SETO` call is issued on an I/O PCB in an IFP region, the DEALLOCATE_ABEND option is not valid. If you attempt to use the option under these conditions, an AD status code is returned to the application.

The option is mutually exclusive with the SEND_ERROR option. If both options are coded in the options list, an AR status code is returned to the application.

Related Reading: For more information about APPC and LU 6.2, see *IMS/ESA Administration Guide: Transaction Manager*.

Options List Feedback Area

When errors are encountered in the options list, the options list feedback area is used to return error information to the application.

IMS attempts to parse the entire options list and return information on as many errors as possible. If the feedback area is not large enough to contain all the error information, only as much information is returned as space permits. The status code is the only indication of an option list error if you do not specify the area.

The feedback area must be initialized by the application with a length field indicating the length of the area. A feedback area approximately 1½ to 2 times the length of the options list or a minimum of 8 words should be sufficient.

Error Codes

This section contains information on error codes that your application can receive.

Error Code	Reason
(0002)	Unrecognized option keyword. Possible reasons for this error are: • The keyword is misspelled. • The keyword is spelled correctly but is followed by an invalid delimiter. • The length specified field representing the PRTO is shorter than the actual length of the options. • A keyword is not valid for the indicated call.
(0004)	Either too few or too many characters were specified in the option

variable. An option variable following a keyword in the options list for the call is not within the length limits for the option.

(0006) The length field (LL) in the option variable is too large to be contained in the options list. The options list length field (LL) indicates that the options list ends before the end of the specified option variable.

(0008) The option variable contains an invalid character or does not begin with an alphabetic character.

(000A) A required option keyword was not specified.

Possible reasons for this error are:

- One or more additional keywords are required because one or more keywords were specified in the options list.
- The specified length of the options list is more than zero but the list does not contain any options.

(000C) The specified combination of option keywords is invalid. Possible causes for this error are:

- The keyword is not allowed because of other keywords specified in the options list.
- The option keyword is specified more than once.

(000E) IMS found an error in one or more operands while it was parsing the print data set descriptors. IMS usually uses MVS/ESA services (SJF) to validate the print descriptors (PRTO= option variable). When IMS calls SJF, it requests the same validation as for the TSO OUTDES command. Therefore, IMS is insensitive to changes in output descriptors. Valid descriptors for your system are a function of the MVS/ESA release level. For a list of valid descriptors and proper syntax, use the TSO HELP OUTDES command.

IMS must first establish that the format of the PRTO options is in a format that allows the use of SJF services. If it is not, IMS returns the status code **AS**, the error code (000E), and a descriptive error message. If the error is detected during the SJF process, the error message from SJF will include information of the form (R.C.=xxxx,REAS.=yyyyyyyy), and an error message indicating the error. For more information on SJF return and reason codes, see *MVS/ESA Application Development Guide: Authorized Assembler Language Programs*.

The range of some variables is controlled by the initialization parameters. Values for the maximum number of copies, allowable remote destination, classes, and form names are examples of variables influenced by the initialization parameters.

Restrictions

A CPI-C driven application program can issue `SETO` calls only to an alternate PCB.

Chapter 4. Writing DL/I Calls for System Services

This chapter describes the system service calls you can use with IMS TM in each type of IMS application program and the parameters for each call. The calls are listed in alphabetical order.

Each call description contains:

- A syntax diagram
- A definition for each parameter that can be used in the call
- Details on how to use the call in your application program
- Restrictions on the use of the call

Each parameter is described as an input or output parameter. “Input” refers to input to IMS from the application program. “Output” refers to output from IMS to the application program.

System service calls must refer only to I/O PCBs. The system service calls are described only as they pertain to IMS TM functions.

Syntax diagrams for these calls begin with the *function* parameter. The call, the call interface, (xxxTDLI), and *parmcount* (if it is required) are not included in the following syntax diagrams. See specific information for assembler language, COBOL, Pascal, and PL/I in “Chapter 2. Defining Application Program Elements” on page 31 for the complete structure.

In this Chapter:

- “APSB Call” on page 90
- “CHKP (Basic) Call” on page 91
- “CHKP (Symbolic) Call” on page 92
- “DPSB Call” on page 93
- “GMSG Call” on page 94
- “GSCD Call” on page 96
- “ICMD Call” on page 97
- “INIT Call” on page 99
- “INQY Call” on page 102
- “LOG Call” on page 109
- “RCMD Call” on page 111
- “ROLB Call” on page 112
- “ROLL Call” on page 114
- “ROLS Call” on page 114
- “SETS/SETU Call” on page 116
- “SYNC Call” on page 118
- “XRST Call” on page 118

Related Reading: The DL/I calls used for database management are described in *IMS/ESA Application Programming: Database Manager*. EXEC DL/I commands used in CICS are described in *IMS/ESA Application Programming: EXEC DLI*

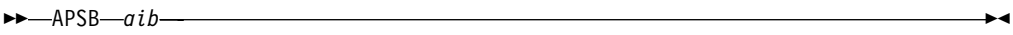
© Copyright IBM Corp. 1974, 2000

Commands for CICS and IMS. DCCTL users can issue calls using GSAM database PCBs. GSAM databases are described in *IMS/ESA Application Programming: Database Manager*.

APSB Call

The Allocate PSB (APSB) call is used to allocate a PSB for a CPI Communications driven application program. These types of application programs are used for conversations that include LU 6.2 devices.

Format



Call Name	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
APSB	X		X		

Parameters

aib
Specifies the application interface block (AIB) that is used for the call. This parameter is an input and output parameter.

The following fields must be initialized in the AIB:

AIBID
Eyecatcher. This 8-byte field must contain DFSAIBbb.

AIBLEN
AIB lengths. This field must contain the actual length of the AIB that the application program obtained.

AIBRSNM1
Resource name. This 8-byte, left-justified field must contain the PSB name.

Usage

CPI-C driven application programs must be link edited with the IMS language interface module and must indicate the PSB to be used before the application program can issue DL/I calls. The APSB call uses the AIB to allocate a PSB for these types of application programs.

When you issue the APSB call, IMS TM returns a list of PCB addresses contained in the specified PSB to the application program. The PCB list is returned in the AIBRSA1 field in the AIB.

IMS TM allows the APSB call to complete even if the databases that the PSB points to are not available. You can issue the INIT call to inform IMS TM of the application program's capabilities to accept additional status codes regarding data availability.

Related Reading: For more information on CPI Communications driven application programs, see *IMS/ESA Application Programming: Design Guide*.

Restrictions

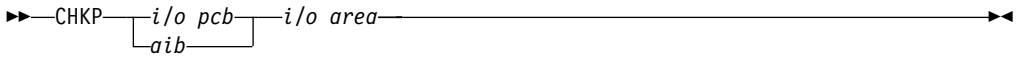
An application program that uses APSB can allocate only one PSB at a time. If your application requires more than one PSB, you must first release the PSB in use by issuing the deallocate PSB (DPSB) call.

CPI Communications driven application programs must issue the APSB call before issuing any other DL/I calls. If your application program attempts to issue DL/I calls before a PSB has been allocated with the APSB call, the application program receives error return and reason codes in the AIB.

CHKP (Basic) Call

A basic Checkpoint (CHKP) call is used for recovery purposes.

Format



Call Name	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
CHKP	X	X	X	X	X

Parameters

i/o pcb
Specifies the I/O PCB to use for this call. It is an input and output parameter.

aib
Specifies the application interface block (AIB) that is used for the call. This parameter is an input and output parameter.

The following fields must be initialized in the AIB:

AIBID
Eyecatcher. This 8-byte field must contain DFSAIB␣␣.

AIBLEN
AIB lengths. This field must contain the actual length of the AIB that the application program obtained.

AIBRSNM1
Resource name. This 8-byte, left-justified field must contain the PCB name IOPCB␣␣␣␣.

AIBOALEN
I/O area length. This field must contain the length of the I/O area that is specified in the call list.

i/o area
Specifies the I/O area to use for the call. This parameter is an input and output parameter. For the CHKP call, the I/O area that contains the 8-character checkpoint ID. If the program is an MPP or a message-driven BMP, the CHKP call implicitly returns the next input message into this I/O area. Therefore, the area must be long enough to hold the longest message that can be returned.

Usage

In transaction management application programs, the basic CHKP call can be used to retrieve the conversational SPA or the initial message segment that was queued before the application was scheduled. The CHKP call commits all changes made by the program and, if your application program abends, establishes the point at which the program can be restarted.

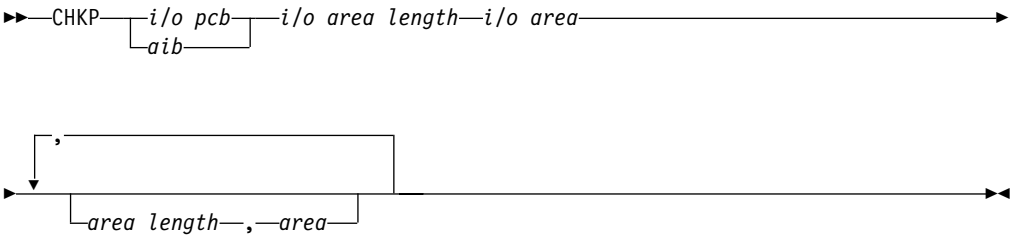
Restrictions

CPI Communications driven application programs cannot issue a basic CHKP call.

CHKP (Symbolic) Call

A symbolic Checkpoint (CHKP) call is used for recovery purposes.

Format



Call Name	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
CHKP	X	X	X	X	X

Parameters

i/o pcb
Specifies the I/O PCB to use for the call. This parameter is an input and output parameter.

aib
Specifies the application interface block (AIB) that is used for the call. This parameter is an input and output parameter.

The following fields must be initialized in the AIB:

AIBID
Eyecatcher. This 8-byte field must contain DFSAIBbb.

AIBLEN
AIB lengths. This field must contain the actual length of the AIB that the application program obtained.

AIBRSNM1
Resource name. This 8-byte, left-justified field must contain the PCB name IOPCBbbb.

AIBOALEN
I/O area length. This field must contain the length of the I/O area that is specified in the call list.

i/o area length
is no longer used by IMS. For compatibility reasons, this parameter must still be included in the call, and it must contain a valid address. You can get a valid address by specifying the name of any area in your program.

i/o area
Specifies the I/O area to be used for your call. This parameter is an input and output parameter. For the CHKP call, the I/O area contains the 8-character checkpoint ID. If the program is a message-driven BMP, the CHKP call implicitly returns the next input message into this I/O area. Therefore, the area must be long enough to hold the longest message that can be returned.

area length
Specifies a 4-byte field in your program that contains the length in binary of the first area to checkpoint. This parameter is an input parameter. Up to seven area lengths can be specified. For each area length, you must also specify an area parameter.

area
Specifies the area in your program that you want IMS to checkpoint. This parameter is an input parameter. You can specify up to seven areas in your program that you want IMS to checkpoint. Always specify the area length parameter first, followed by the area parameter. The number of areas you specify on a XRST call must be less than or equal to the number of areas you specify on the CHKP calls the program issues. When you restart the program, IMS restores only the areas you specified in the CHKP call.

Usage

In transaction management application programs, the symbolic CHKP call can be used to retrieve the conversational SPA or the initial message segment that was queued before the application was scheduled. The CHKP call commits all changes made by the program and, if your application program abends, establishes the point at which the program can be restarted. In addition, the symbolic CHKP call can:

- Work with the extended restart (XRST) call to restart your program if your program abends.
- Enables you to save as many as seven data areas in your program, which are restored when your program is restarted.

Restrictions

A CPI Communications driven application program cannot issue the symbolic CHKP call. The symbolic CHKP call is only allowed from batch and BMP applications.

You must issue an XRST call before the symbolic CHKP call.

DPSB Call

The Deallocate PSB (DPSB) call frees a PSB that was allocated with the APSB call.

Format

System Service Call: DPSB

Call Name	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
DPSB	X		X		

Parameters

aib
Specifies the application interface block (AIB) that is used for the call. This parameter is an input and output parameter.

The following fields must be initialized in the AIB:

AIBID
Eyecatcher. This 8-byte field must contain DFSAIBbb.

AIBLEN
AIB lengths. This field must contain the actual length of the AIB that the application program obtained.

AIBRSNM1
Resource name. This 8-byte, left-justified field must contain the PSB name.

Usage

The DPSB call must be used in a CPI Communications driven application program to release a PSB after a commit point occurs and before another PSB can be allocated. In a CPI Communications driven application program, the commit point is achieved with the COMMIT verb. For more information on CPI Communications driven application programs, see “CPI-C Driven Application Programs” on page 143.

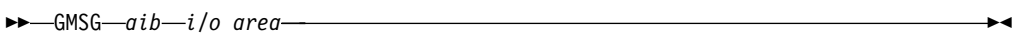
Restrictions

You can issue the DPSB call only after a commit point occurs, and it is valid only after a successful APSB call.

GMSG Call

A Get Message (GMSG) call is used in an automated operator (AO) application program to retrieve a message from AO exit routine DFSAOE00.

Format



Parameters

aib
Specifies the application interface block (AIB) to be used for this call. This parameter is an input and output parameter.

You must initialize the following fields in the AIB:

AIBID
Eyecatcher. This 8-byte field must contain DFSAIBbb.

AIBLEN
AIB lengths. This field must contain the actual length of the AIB that the application program obtained.

AIBSFUNC
Subfunction code. This field must contain one of the following 8-byte subfunction codes:

8-blanks (null)
When coded with an AOI token in the AIBRSNM1 field, indicates IMS is to return when no AOI message is available for the application.

WAITAOI
When coded with an AOI token in the AIBRSNM1 field, indicates IMS is to wait for an AOI message when none is currently available for the application. This subfunction value is invalid if an AOI token is not coded in AIBRSNM1. In this case, error return and reason codes are returned in the AIB.

The value WAITAOI must be left justified and padded with a blank character.

AIBRSNM1
Resource name. This field must contain the AOI token or blanks. The AOI token identifies the message the AO application is to retrieve. The token is supplied for the first segment of a message. If the message is a multisegment message, set this field to blanks to retrieve the second through the last segment. AIBRSNM1 is an 8-byte alphanumeric left-justified field padded with blanks.

AIBOALEN
I/O area length. This field must contain the length of the I/O area that is specified in the call list. This field is not changed by IMS.

AIBOAUSE
Length of the data returned in the I/O area. This parameter is an output parameter.

When partial data is returned because the I/O area is not large enough, AIBOAUSE contains the length required to receive all of the data, and AIBOALEN contains the actual length of the data.

i/o area
Specifies the I/O area to use for this call. This parameter is an output parameter. The I/O area should be large enough to hold the largest segment passed from IMS to the AO application. If the I/O area is not large enough to contain all of the data, IMS returns partial data.

Usage

GMSG is used in an AO application to retrieve a message associated with an AOI token. The AO application must pass an 8-byte AOI token to IMS to retrieve the first segment of the message. IMS uses the AOI token to associate messages from AO exit routine DFSAOE00 with the GMSG call from an AO application. IMS returns to the application only those messages associated with the AOI token. By using different AOI tokens, DFSAOE00 can direct messages to different AO applications. Note that your installation defines the AOI token.

Related Reading: For more information on the AOI exits, see *IMS/ESA Customization Guide*.

To retrieve the second through the last segments of a multisegment message, issue GMSG calls with no token specified (set the token to blanks). If you want to retrieve all segments of a message, you must issue GMSG calls until all segments are retrieved. IMS discards all nonretrieved segments of a multisegment message when a new GMSG call specifying an AOI token is issued.

Your AO application can specify a wait on the GMSG call. If no messages are currently available for the associated AOI token, your AO application waits until a message is available. The decision to wait is specified by the AO application, unlike a WFI transaction where the wait is specified in the transaction definition. The wait is done on a call basis; that is, within a single AO application some GMSG calls might specify waits while others do not.

Table 6 shows, by IMS environment, the types of application programs that can issue GMSG. GMSG is also supported from a CPI-C driven application program.

Table 6. GMSG Support by Application Region Type

Application Region Type	IMS Environment		
	DBCTL	**DB/DC**	**DCCTL**
DRA thread	Yes	Yes	N/A
BMP (nonmessage-driven)	Yes	Yes	Yes
BMP (message-driven)	N/A	Yes	Yes
MPP	N/A	Yes	Yes
IFP	N/A	Yes	Yes

Restrictions

A CPI-C driven program must issue an APSB (allocate PSB) call before issuing GMSG.

GSCD Call

This section contains programming interface information.

The Get System Contents Directory (GSCD) call retrieves the address of the IMS system contents directory (SCD) for batch programs.

Format

Call Name	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
GSCD				X	X

Parameters

i/o pcb
Specifies the PCB to use for this call. This parameter is an input and output parameter.

aib
Specifies the address of the application interface block (AIB) that is used for the call. This parameter is an input and output parameter.

The following fields must be initialized in the AIB:

AIBID
Eyecatcher. This 8-byte field must contain DFSAIB␣␣.

AIBLEN
AIB lengths. This field must contain the actual length of the AIB that the application program obtained.

AIBRSNM1
Resource name. This 8-byte, left-justified field must contain the PCB name IOPCB␣␣␣.

AIBOALEN
I/O area length. This field must contain the length of the I/O area that is specified in the call list.

i/o area
Specifies the I/O area to be used for the call. This parameter is an output parameter. For the GCSD call, the I/O area must be 8 bytes in length. IMS TM places the address of the SCD in the first 4 bytes and the address of the program specification table (PST) in the second 4 bytes.

Usage

IMS does not return a status code to a program after it issues a successful GSCD call. The status code from the previous call that used the same PCB remains unchanged in the PCB.

Restrictions

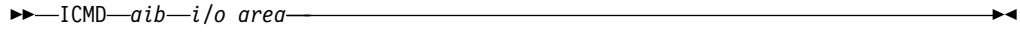
The GSCD call can be issued only from batch application programs.

For more information on GSCD, see *IMS/ESA Application Programming: Design Guide*.

ICMD Call

An Issue Command (ICMD) call lets an automated operator (AO) application program issue an IMS command and retrieve the first command response segment.

Format



Parameters

aib
Specifies the application interface block (AIB) used for this call. This parameter is an input and output parameter.

The following fields must be initialized in the AIB:

AIBID
Eyecatcher. This 8-byte field must contain DFSAIBbb.

AIBLEN
AIB lengths. This field must contain the actual length of the AIB that the application program obtained.

AIBOALEN
I/O area length. This field must contain the length of the I/O area that is specified in the call list. This field is not changed by IMS.

AIBOAUSE
Length of data returned in the I/O area. This parameter is an output parameter.

Your program must check this field to determine whether the ICMD call returned data to the I/O area. When the only response to the command is a DFS058 message indicating either COMMAND IN PROGRESS or COMMAND COMPLETE, the response is not returned.

When partial data is returned because the I/O area is not large enough, AIBOAUSE contains the length required to receive all of the data, and AIBOALEN contains the actual length of the data.

i/o area
Specifies the I/O area to use for this call. This parameter is an input and output parameter. The I/O area should be large enough to hold the largest command passed from the AO application to IMS, or command response segment passed from IMS to the AO application. If the I/O area is not large enough to contain all of the data, IMS returns partial data.

The general format of your I/O work area on an ICMD call is:

```
LLZZ/VERB KEYWORD1 P1 KEYWORD2 P2, P3.
```

LL	Two-byte field containing the length of the command text, including LLZZ.
ZZ	Two-byte field reserved for IMS.
***/* or CRC**	Indicates an IMS command follows. CRC (Command Recognition Character) rather than a slash (/) is used in the DBCTL environment.
VERB	The IMS command you are issuing.
KEYWORDX	Keywords that apply to the command being issued.
PX	Parameters for the keywords you are specifying.
. (Period)	End of the command.

The length of a command is limited by the size of the I/O area; the size is specified in the IOASIZE parameter in the PSBGEN macro during PCB generation. LL is the length of the command text. The size of the I/O area is the length of the actual command text, plus 4 bytes for LLZZ. The minimum size of the I/O work area is 132 bytes.

The fifth byte must be a "/" (or CRC for DBCTL), and the verb must follow immediately. The /BROADCAST and /LOOPTEST commands must have a period between the command segment and text segment, and must be preceded by

an LLZZ field that includes the size of the text. Comments can be added by placing a period (.) after the last parameter.

Restriction: When issuing the /SSR command, do not code an end-of-command indicator (period) as shown in the *IMS/ESA Operator's Reference*. If a period is used, it is considered part of the text.

Usage

ICMD enables an AO application to issue an IMS command and retrieve the first command response segment.

When using ICMD, put the IMS command that is to be issued in your application's I/O area. After IMS has processed the command, it returns the first segment of the response message to your AO application's I/O area to retrieve subsequent segments (one segment at a time), using the RCMD call.

Some IMS commands that complete successfully result in a DFS058 COMMAND COMPLETE message. Some IMS commands that are processed asynchronously result in a DFS058 COMMAND IN PROGRESS message. For a command entered on an ICMD call, neither DFS058 message is returned to the AO application. The AIBOAUSE field is set to zero to indicate no segment was returned. So, your AO application must check the AIBOAUSE field along with the return and reason codes to determine if a response was returned.

Related Reading: For more information on the AOI exits, see *IMS/ESA Customization Guide*.

Table 7 shows, by IMS environment, the types of application programs that can issue ICMD. ICMD is also supported from a CPI-C driven application.

Table 7. ICMD Support by Application Region Type

	IMS Environment		
Application Region Type	**DBCTL**	**DB/DC**	**DCCTL**
DRA thread	Yes	Yes	N/A
BMP (nonmessage-driven)	Yes	Yes	Yes
BMP (message-driven)	N/A	Yes	Yes
MPP	N/A	Yes	Yes
IFP	N/A	Yes	Yes

See the *IMS/ESA Operator's Reference* for a list of commands that can be issued using the ICMD call.

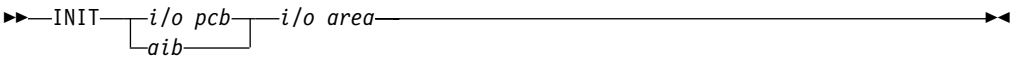
Restrictions

A CPI-C driven program must issue an APSB (allocate PSB) call before issuing ICMD.

INIT Call

An Initialize (INIT) call allows the application to receive data availability status codes by checking each DB PCB for data availability.

Format



Call Name	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
INIT	X	X	X	X	X

Parameters

i/o pcb
Specifies the I/O PCB to use for this call. This parameter is an input and output parameter.

aib
Specifies the address of the application interface block (AIB) that is used for the call. This parameter is an input and output parameter.

The following fields must be initialized in the AIB:

AIBID
Eyecatcher. This 8-byte field must contain DFSAIBbb.

AIBLEN
AIB lengths. This field must contain the actual length of the AIB that the application program obtained.

AIBRSNM1
Resource name. This 8-byte, left-justified field must contain the PCB name IOPCBbbb.

AIBOALEN
I/O area length. This field must contain the length of the I/O area that is specified in the call list.

i/o area
Specifies the I/O area to be used for the call. This parameter is an input parameter. For the INIT call, the I/O area contains the character string “DBQUERY”.

Usage

The INIT call is valid for all IMS TM application programs.

To specify the database query subfunction in your application program, specify the character string “DBQUERY” in the I/O area.

Determining Database Availability: INIT DBQUERY

When the INIT call is issued with the DBQUERY character string in the I/O area, the application program can obtain information regarding the availability of data for each PCB. Table 8 and Table 9 on page 101 contain sample I/O areas for the INIT call with DBQUERY.

Table 8. INIT I/O Area Examples for All xxxTDLI Interfaces Except PLITDLI

L	L	Z	Z	Character String
00	0B	00	00	DBQUERY

Table 8. INIT I/O Area Examples for All xxxTDLI Interfaces Except PLITDLI (continued)

L	L	Z	Z	Character String

Note: The LL and ZZ fields are binary. The LL value X'0B' is a hexadecimal representation of decimal 11.

Table 9. INIT I/O Area Examples for the PLITDLI Interface

L	L	L	L	Z	Z	Character String
00	00	00	0B	00	00	DBQUERY

Note: The LLLL and ZZ fields are binary. The L value X'0B' is a hexadecimal representation of decimal 11.

LL or LLLL
A 2-byte field that contains the length of the character string, plus 2 bytes for LL. For the PLITDLI interface, use the 4-byte field LLLL. When you use the AIBTDLI interface, PL/I programs require only a 2-byte field.

ZZ A 2-byte field of binary zeros.

One of the following status codes is returned for each database PCB:

NA At least one of the databases that can be accessed using this PCB is not available. A call made using this PCB probably results in a BA or BB status code if the `INIT STATUS GROUPA` call has been issued, or in a DFS3303I message and 3303 pseudoabend if it has not. An exception is when the database is not available because dynamic allocation failed. In this case, a call results in an AI (unable to open) status code.

In a DCCTL environment, the status code is always NA.

NU At least one of the databases that can be updated using this PCB is unavailable for update. An `ISRT`, `DLET`, or `REPL` call using this PCB might result in a BA status code if the `INIT STATUS GROUPA` call has been issued, or in a DFS3303I message and 3303 pseudoabend if it has not. The database that caused the NU status code might be required only for delete processing. In that case, `DLET` calls fail, but `ISRT` and `REPL` calls succeed.

ƀƀ The data that can be accessed with this PCB can be used for all functions the PCB allows. DEDBs and MSDBs always have the ƀƀ status code.

In addition to data availability status, the name of the database organization of the root segment is returned in the segment name field of the PCB. In DCCTL environments, the name of the database organization is UNKNOWN.

Automatic INIT DBQUERY

When the application program is entered initially, the status code in the database PCBs is initialized as if the `INIT DBQUERY` call was issued. This enables the application program to determine database availability without issuing the `INIT` call.

In DCCTL environments, the status code is NA.

Performance Considerations for the INIT Call (IMS Online Only)

For performance reasons, the `INIT` call should not be issued in online application programs before the first `GU` call to the I/O PCB. If the `INIT` call is issued first, the `GU` call to the I/O PCB is not processed as efficiently.

INQY Call

The Inquiry (INQY) call is used to request information regarding execution environment, destination type and status, and session status. INQY is valid only when using the AIBTDLI interface.

Format

Call Name	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
INQY	X	X	X	X	X

Parameters

aib

Specifies the address of the application interface block (AIB) that is used for the call. This parameter is an input and output parameter. The following fields must be initialized in the AIB:

AIBID
Eyecatcher. This 8-byte field must contain DFSAIBbb.

AIBLEN
AIB lengths. This field must contain the actual length of the AIB that the application program obtained.

AIBSFUNC
Subfunction code. This field must contain one of the 8-byte subfunction codes as follows:

bbbbbbbb (Null)
DBQUERYb
FINDbbbb
ENVIRONb
PROGRAMb

Use of the PCB and I/O area with the subfunctions is summarized in Table 13 on page 109.

AIBRSNM1
Resource name. This 8-byte, left-justified field must contain the PCB name of any PCB named in the PSB.

AIBOALEN
I/O area length. This field must contain the length of the I/O area that is specified in the call list. This field is not changed by IMS.

i/o area

Specifies the I/O area to be used for the INQY call. This parameter is an output parameter. An I/O area is required for INQY subfunctions ENVIRON, PROGRAM, and null. It is not required for subfunctions DBQUERY and FIND.

Usage

The INQY operates in both batch and online IMS TM environments. IMS TM application programs can use the INQY call to request information regarding output destination, session status, the current execution environment, the availability of databases, and the PCB address, which is based on the PCB name. Before you can issue an INQY call, you must initialize the fields of the AIB. See “Using the AIBTDLI Interface” on page 51 for more information.

When you use the INQY call, specify an 8-byte subfunction code, which is passed in the AIB. The INQY subfunction determines the information that the application receives. For a summary of PCB type and I/O area use for each subfunction, see Table 13 on page 109.

The INQY call returns information to the caller’s I/O area. The length of the data returned from the INQY call is passed back to the application in the AIB field AIBOAUSE.

You specify the size of the I/O area in the AIB field AIBOALEN. The INQY call returns only as much data as the area can hold in one call. If the area is not large enough for all the data, an AG status code is returned, and partial data is returned in the I/O area. In this case, the AIB field AIBOALEN contains the actual length of the data returned to the I/O area, and the AIBOAUSE field contains the output area length that would be required to receive all the data.

Querying Information from the PCB: INQY Null

When the INQY call is issued with the null subfunction, the application program obtains information related to the PCB, including output destination type and location, and session status. The INQY call can use the I/O PCB or the alternate PCB. The information you receive regarding destination location and session status is based on the destination type. The destination types are as follows: APPC, OTMA, TERMINAL, TRANSACT, and UNKNOWN.

Related Reading: For more information about APPC and LU 6.2, see *IMS/ESA Administration Guide: Transaction Manager*.

The INQY null subfunction returns character string data in the I/O area. The output that is returned for the destination types APPC, OTMA, TERMINAL, and TRANSACT is left justified and padded with blanks. The UNKNOWN destination type does not return any information. Table 10 lists the output returned from the INQY null call.

Table 10. INQY Null Data Output

Destination Type	Information Returned	Length in Bytes	Actual Value	Explanation
Terminal	Destination Type	8	TERMINAL	The destination of the I/O PCB or alternate PCB is a terminal.
	Terminal Location	8	LOCAL	The terminal is defined as local.
			REMOTE	The terminal is defined as remote.
	Queue Status	8	STARTED	The queue is started and can accept work.
			STOPPED	The queue is stopped and cannot accept work.
	Session Status	8	b	The status is not available.
			ACTIVE	The session is active.
			INACTIVE	The session is inactive.

Table 10. INQY Null Data Output (continued)

Destination Type	Information Returned	Length in Bytes	Actual Value	Explanation
Transaction	Destination Type	8	TRANSACT	The destination of the alternate PCB is a program.
	Transaction Location	8	LOCAL	The transaction is defined as local.
			REMOTE	The transaction is defined as remote.
			DYNAMIC	The transaction is defined as dynamic.[1]
			ƀ	The Program Routing exit routine has defined the destination as a transaction not on this system.
	Transaction Status	8	STARTED	The transaction can be scheduled.
			STOPPED	The transaction cannot be scheduled.
			ƀ	The Program Routing exit routine has defined the destination as a transaction not on this system. The transaction status is not available.
	Destination PSB Name	8		This field gives the name of the destination PSB.
			ƀ	The Program Routing exit routine has defined the destination as a transaction not on this system or the transaction is dynamic. The transaction destination is not available.
	Destination Program or Session Status	8	ƀ	The status is not available.
			ACTIVE	The session is active (remote transaction).
			INACTIVE	The session is inactive (remote transaction).
			STARTED	The program can be scheduled (local transaction).
			STOPPED	The program cannot be scheduled (local transaction).
APPC	Destination Type	8	APPC	The destination is an LU 6.2 device.
	APPC/MVS Side Information Entry Name[2]	8		This field provides the Side Name.
			ƀ	The Side Name is not available.
	Partner Logical Unit Name[3]	8		This field provides the partner LU name for the conversation.
			ƀ	The partner LU name is not available.
	Partner Mode Table Entry Name[4]	8		This field provides the Mode Name for the conversation.
			ƀ	The Mode Name is not available.
	User Identifier	8		This field provides the user ID.
			ƀ	The user ID is not available.
	Group Name	8		This field provides the Group Name.
			ƀ	The Group Name is not available.

Table 10. INQY Null Data Output (continued)

Destination Type	Information Returned	Length in Bytes	Actual Value	Explanation
	Synchronization Level[5]	1	C	The synchronization level is defined as CONFIRM.
			N	The synchronization level is defined as NONE.
	Conversation Type[6]	1	B	The conversation is defined as BASIC.
			M	The conversation is defined as MAPPED.
	Address of TPN[7]	4		This is the address of the LL field of the Transaction Program Name.[8]
			0	The address of the Transaction Program Name is not available.
OTMA	Destination Type	8	OTMA	The destination is an OTMA client.
	Tpipe Name	8		This field provides the OTMA transaction pipe name.
			ƀ	The Tpipe Name is not available.
	Member Name	16		This field provides the OTMA client's XCF member name.
			ƀ	The Member Name is not available.
	User Identifier	8		This field provides the User ID.
			ƀ	The User ID is not available.
	Group Name	8		This field provides the group name.
			ƀ	The Group Name is not available.
	Synchronization Level	1	S	The OTMA transaction pipe is synchronized.
			ƀ	The OTMA transaction pipe is not synchronized.
	Message Synchronization Level[5]	1	C	The synchronization level is defined as CONFIRM.
			N	The synchronization level is defined as NONE.
Unknown	Destination Type	8	UNKNOWN	Unable to find destination.

Note:

1. A dynamic transaction is only possible in a shared-queues environment. A transaction is dynamic when it is not defined to the IMS system that is sending the message, but rather to another IMS system that is sharing the queues. The dynamic transaction is created when the Output Creation exit routine, DFSINSX0, indicates a transaction whose destination is unknown to IMS. The output fields for the destination PSB name and destination program are set to blanks.
2. If the call is issued against an I/O PCB, the Side Name cannot be used and ƀ is returned. If the call is issued against an alternate modifiable PCB, the Side Name must be supplied in a `CHNG` call that is issued before `INQY`.
3. If the call is issued against an I/O PCB, the LU name must be coded. If the call is issued against a modifiable alternate PCB, the LU name must be supplied in a `CHNG` call that is issued before `INQY`.
4. If the call is issued against an I/O PCB, the Mode Name cannot be used and ƀ is returned. If the call is issued against an alternate modifiable PCB, the Mode Name must be supplied in a `CHNG` call that is issued before `INQY`.
5. When the synchronization level is not available, IMS uses the default value of CONFIRM.
6. When the conversation type is not available, IMS uses the default value of MAPPED.
7. The pointer identifies a length field (LL), which contains the length of the TPN in binary, including the 2 bytes required for LL.
8. The TPN can be up to 64 bytes long.

The contents of the output fields vary depending on the type of PCB used for the INQY call. Table 11 shows how INQY output for APPC destinations varies depending on the PCB type. The PCB can be an I/O PCB or an alternate PCB.

Table 11. INQY Output and PCB Type

Output Field	I/O PCB	Alternate PCB (Non-modifiable)	Alternate PCB (Modifiable)
Destination Type	APPC	APPC	APPC
Side Name	blanks	Side Name if available or blanks	Side Name if supplied on previous CHNG call or blanks
LU Name	Input LU Name	LU Name if available or blanks	LU Name if supplied on previous CHNG call or blanks
Mode Name	blanks	Mode Name if available or blanks	Mode Name if supplied on previous CHNG call or blanks
User Identifier	USERID if available or blanks	USERID if available or blanks	USERID if available or blanks
Group Name	Group Name if available or blanks	Group Name if available or blanks	Group Name if available or blanks
Sync Level	C or N	C or N	C or N
Conversation Type	B or M	B or M	B or M
TPN Address	Address of the TPN character string	Address of the TPN character string or zero	Address of the TPN character string or zero
TPN character string **Note:** If your TPN name is DFSASYNC, the destination represents an asynchronous conversation.	Inbound name of IMS Transaction that is executing.	Partner TPN, if available. If not available, address field is zero.	TP Name if it is supplied on the previous CHNG call. If not supplied, the address field is zero.

Related Reading: For more information on APPC and LU 6.2, see *IMS/ESA Administration Guide: Transaction Manager*.

Querying Data Availability: INQY DBQUERY

When the INQY call is issued with the DBQUERY subfunction, the application program obtains information regarding the data for each PCB. The only valid PCB name that can be passed in AIBRSNM1 is "IOPCBbbb". The INQY DBQUERY call is similar to the INIT DBQUERY call. It updates status codes in the database PCBs, but it does not return information in the I/O area.

In addition to the INIT DBQUERY status codes, the INQY DBQUERY call returns the following status codes in the I/O PCB:

Blanks The call is successful and all databases are available.

BJ None of the databases in the PSB are available, or no PCBs exist in the PSB. All database PCBs (excluding GSAM) contain an NA status code as the result of processing the INQY DBQUERY call.

BK At least one of the databases in the PSB is not available or availability is limited. At least one database PCB contains an NA or NU status code as the result of processing the INQY DBQUERY call.

The INQY call returns the following status codes in each DB PCB:

NA At least one of the databases that can be accessed using this PCB is not

available. A call made using this PCB probably results in a BA or BB status code if the INIT STATUS GROUPA call has been issued, or in a DFS3303I message and 3303 pseudoabend if it has not. An exception is when the database is not available because dynamic allocation failed. In this case, a call results in an AI (unable to open) status code.

In a DCCTL environment, the status code is always NA.

NU At least one of the databases that can be updated using this PCB is unavailable for update. An ISRT, DLET, or REPL call using this PCB might result in a BA status code if the INIT STATUS GROUPA call has been issued, or in a DFS3303I message and 3303 pseudoabend if it has not. The database that caused the NU status code might be required only for delete processing. In that case, DLET calls fail, but ISRT and REPL calls succeed.

ƀƀ The data that can be accessed with this PCB can be used for all functions the PCB allows. DEDBs and MSDBs always have the ƀƀ status code.

Querying the Environment: INQY ENVIRON

When the INQY call is issued with the ENVIRON subfunction, the application program obtains information regarding the current execution environment. The only valid PCB name that can be passed in AIBRSNM1 is “IOPCBƀƀƀ”. This includes the IMS identifier, release, region, and region type. The INQY ENVIRON call returns character string data in the I/O area. The output is left justified and padded with blanks on the right. Table 12 lists the output returned from the INQY ENVIRON call. To receive the following data and to account for expansion, the I/O area length should be larger than 140 bytes:

```
 88 bytes    INQY ENVIRON data
  2 bytes     Length field for Recovery Token section (18 bytes)
 16 bytes    Recovery Token
  2 bytes     Length field for APARM section (maximum of 34 bytes)
 32 bytes    APARM data
_____________________________________________
140 bytes    Total I/O area length
```

Table 12. INQY ENVIRON Data Output

Information Returned	Length in Bytes	Actual Value	Explanation
IMS Identifier	8		Provides the identifier from the execute parameters.
IMS Release Level	4		Provides the release level for IMS. For example, X'00000410'
IMS Control Region Type	8	BATCH	Indicates that an IMS Batch region is active.
		DB	Indicates that only the IMS Database Manager is active (DBCTL system).
		TM	Indicates that only the IMS Transaction Manager is active (DCCTL system).
		DB/DC	Indicates that both the IMS Database and Transaction managers are active (DB/DC system).
IMS Application Region Type	8	BATCH	Indicates that the IMS Batch region is active.
		BMP	Indicates that the Batch Message Processing region is active.
		DRA	Indicates that the Database Resource Adapter Thread region is active.
		IFP	Indicates that the IMS Fast Path region is active.
		MPP	Indicates that the Message Processing region is active.

Table 12. INQY ENVIRON Data Output (continued)

Information Returned	Length in Bytes	Actual Value	Explanation
Region Identifier	4		Provides the region identifier. For example, X'00000001'
Application Program Name	8		Provides the name of the application program being run.
PSB Name (currently allocated)	8		Provides the name of the PSB currently allocated.
Transaction Name	8		Provides the name of the transaction.
		ƀ	Indicates that there is no transaction.
User Identifier [1]	8		Provides the user ID.
		ƀ	Indicates that the user ID is unavailable.
Group Name	8		Provides the group name.
		ƀ	Indicates that the group name is unavailable.
Status Group Indicator	4	A	Indicates an INIT STATUS GROUPA call is issued.
	4	B	Indicates an INIT STATUS GROUPB call is issued.
		ƀ	Indicates that a status group is not initialized.
Address of Recovery Token [2]	4		Provides the address of the LL field, followed by the Recovery Token.
Address of the Application Parameter String [2]	4		Provides the address of the LL field, followed by the application parameter string.
		0	Indicates that the APARM=parameter is not coded in the EXEC (execute) parameters of the dependent region JCL.
Shared Queues Indicator	4		Indicates IMS is not using Shared Queues.
	4	SHRQ	Indicates IMS is using Shared Queues.

Note:

1. The user ID is derived from the PSTUSID field of the PST that represents the region making the `INQY ENVIRON` call. The PSTUSID field is one of the following:
   - For message-driven BMP regions that have not completed successful `GU` calls to the IMS message queue and for non-message-driven BMP regions, the PSTUSID field is derived from the name of the PSB currently scheduled into the BMP region.
   - For message-driven BMP regions that have completed successful a `GU` call and for any MPP region, the PSTUSID field is derived from the last message retrieved from the message queue, which is usually the input terminal's RACF ID. If the terminal has not signed onto RACF, the ID is the input terminal's LTERM.
2. The pointer identifies a length field (LL) that contains the length of the recovery token and user parameter in binary, including the 2 bytes required for LL.

Querying the PCB Address: INQY FIND

When the `INQY` call is issued with the FIND subfunction, the application program is returned with the PCB address of the requested PCB name. The valid PCB name that can be passed in AIBRSNM1 are "IOPCBƀƀƀ", or the name of the alternate PCB or database PCB as it is defined in the PSB.

On a FIND subfunction, the requested PCB remains unmodified, and no information is returned in an I/O area.

The FIND subfunction is used to get a PCB address following an `INQY DBQUERY` call. This process allows the application to analyze the PCB status code to determine if an NA or NU status code is set in the PCB.

Querying the Program Name: INQY PROGRAM

When you issue the INQY call with the PROGRAM subfunction, the application program name is returned in the first 8 bytes of the I/O area. The only valid PCB name that can be passed in AIBRSNM1 is "IOPCBbbb".

INQY Return Codes and Reason Codes

When you issue the INQY call, return and reason codes are returned to the AIB. Status codes can be returned to the PCB. See "Return and Reason Code Tables" on page 475 for the return and reason codes that apply to INQY. If return and reason codes other than those that apply to INQY are returned, your application should examine the PCB to see what status codes are found.

Map of INQY Subfunction to PCB Type

Table 13. Subfunction, PCB, and I/O Area Combinations for the INQY Call

Subfunction	I/O PCB	Alternate PCB	DB PCB	I/O Area Required
FIND	OK	OK	OK	NO
Null	OK	OK	NO	YES
ENVIRON	OK	NO	NO	YES
DBQUERY	OK	NO	NO	NO
PROGRAM	OK	NO	NO	YES

Restrictions

A CPI Communications driven application program cannot issue an INQY call with the null subfunction against an I/O PCB.

A batch program cannot issue an INQY call with a null subfunction.

LOG Call

The Log (LOG) call is used to send and write information to the IMS system log.

Format

Call Name	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
LOG	X	X	X	X	X

Parameters

i/o pcb
Specifies the address of the PCB to use for this call. This parameter is an input and output parameter.

aib
Specifies the application interface block (AIB) that is used for the call. This parameter is an input and output parameter.

The following fields must be initialized in the AIB:

AIBID
Eyecatcher. This 8-byte field must contain DFSAIBbb.

AIBLEN
AIB lengths. This field must contain the actual length of the AIB that the application program obtained.

AIBRSNM1
Resource name. This 8-byte, left-justified field must contain the PCB name IOPCBbbb.

AIBOALEN
I/O area length. This field must contain the length of the I/O area that is specified in the call list.

i/o area
Specifies the area in your program that contains the record that you want to write to the system log. This parameter is an input parameter. This record must be in the format shown in Figure 16.
The fields must be as follows:

For the AIBTDLI, ASMTDLI, CBLTDLI, CEETDLI, CTDLI and PASTDLI interfaces:

LL	ZZ	C	Text
2	2	1	Variable

For the PLITDLI interface:

LLLL	ZZ	C	Text
4	2	1	Variable

Figure 16. Log Record Formats for COBOL, PL/I, C Language, Pascal, and Assembler

LL or LLLL Specifies a 2-byte field that contains the length of the record. When you use the AIBTDLI interface, the length of the record is equal to LL + ZZ + C + text of the record. For the PLITDLI interface, the length of the record is equal to LLLL + ZZ + C + the text of the record. When you calculate the length of the log record, you must account for all of the fields. The total length you specify includes:

- 2 bytes for LL or LLLL. (For PL/I, include the length as 2, even though LLLL is a 4-byte field.)
- 2 bytes for the ZZ field.
- 1 byte for the C field.
- n bytes for the length of the record itself.

If you are using the PLITDLI interface, your program must define the length field as a binary fullword.

ZZ Specifies a 2-byte field of binary zeros.

C Specifies a 1-byte field containing a log code, which must be equal to or greater than X'A0'.

Text Specifies any data to be logged.

Usage

An application program can write a record to the system log by issuing the LOG call. When you issue the LOG call, you specify the I/O area that contains the record you want written to the system log. You can write any information to the log, and you can use log codes to distinguish among various types of information. You can issue the LOG:

- In the IMS DB/DC environment, and the record is written to the IMS log.
- In the DCCTL environment, and the record is written to the DCCTL log.

Restrictions

The length of the I/O area (including all fields) cannot be larger than the logical record length (LRECL) for the system log data set minus 4 bytes and the length of logrec prefix, (which is x'4A' bytes in length), or the I/O area specified in the IOASIZE keyword of the PSBGEN statement of the PSB.

RCMD Call

A Retrieve Command (RCMD) call lets an automated operator (AO) application program retrieve the second and subsequent command response segments after an ICMD call.

Format

Parameters

aib
Specifies the application interface block (AIB) used for this call. This parameter is an input and output parameter.

The following fields must be initialized in the AIB:

AIBID
Eyecatcher. This 8-byte field must contain DFSAIBbb.

AIBLEN
AIB lengths. This field must contain the actual length of the AIB that the application program obtained.

AIBOALEN
I/O area length. This field must contain the length of the I/O area that is specified in the call list. This field is not changed by IMS.

AIBOAUSE
Length of data returned in the I/O area. This parameter is an output parameter.

When partial data is returned because the I/O area is not large enough, AIBOAUSE contains the length required to receive all of the data and AIBOALEN contains the actual length of the data.

i/o area
Specifies the I/O area to use for this call. This parameter is an output parameter. The I/O area should be large enough to hold the largest command

response segment passed from IMS to the AO application. If the I/O area is not large enough for all of the information, partial data is returned in the I/O area.

Usage

RCMD lets an AO application retrieve the second and subsequent command response segments resulting from an ICMD call.

Related Reading: For more information on the AOI exits, see *IMS/ESA Customization Guide*.

Table 14 shows, by IMS environment, the types of application programs that can issue RCMD. RCMD is also supported from a CPI-C driven application program.

Table 14. RCMD Support by Application Region Type

Application Region Type	IMS Environment		
	DBCTL	DB/DC	DCCTL
DRA thread	Yes	Yes	N/A
BMP (nonmessage-driven)	Yes	Yes	Yes
BMP (message-driven)	N/A	Yes	Yes
MPP	N/A	Yes	Yes
IFP	N/A	Yes	Yes

RCMD retrieves only one response segment at a time. If you need additional response segments, you must issue RCMD once for each response segment issued by IMS.

Restrictions

An ICMD call must be issued before an RCMD call.

ROLB Call

The Rollback (ROLB) call backs out messages sent by the application program. For more information on the ROLB call, see “Backing out to a Prior Commit Point: ROLL, ROLB, and ROLS Calls” on page 145.

Format

Call Name	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
ROLB	X	X	X	X	X

Parameters

i/o pcb
Specifies the I/O PCB to use for the call. This parameter is an input and output parameter.

aib
Specifies the application interface block (AIB) that is used for the call. This parameter is an input and output parameter.

The following fields must be initialized in the AIB:

AIBID
Eyecatcher. This 8-byte field must contain DFSAIBbb.

AIBLEN
AIB lengths. This field must contain the actual length of the AIB that the application program obtained.

AIBRSNM1
Resource name. This 8-byte, left-justified field must contain the PCB name IOPCBbbb.

AIBOALEN
I/O area length. This field must contain the length of the I/O area that is specified in the call list.

i/o area
An output parameter that specifies the area in your program to which IMS TM returns the first message segment. For conversational transactions the SPA will be the first item returned to the application. Your next GN call will then return the first user segment of the message.

Usage

Issuing a ROLB in a conversational program causes IMS TM to back out the messages that the application program has sent. If the program issues a ROLB call and then reaches a commit point without sending the required response to the originating terminal, IMS TM terminates the conversation and sends the message DFS2171I NO RESPONSE CONVERSATION TERMINATED to the originating terminal.

If your application program has allocated resources that IMS TM cannot roll back, the resources are ignored. For example, if your application program issues CPI-C verbs to allocate resources (for modified DL/I or CPI-C driven programs), ROLB only affects those resources allocated by IMS. Your application must notify any CPI-C conversations that a ROLB call was issued.

For CPI-C driven application programs, all messages inserted to nonexpress alternate PCBs are discarded. Messages inserted to express alternate PCBs are discarded if the PURG call was not issued against the PCB before the ROLB call was issued.

Any application program that uses Spool API functions and creates print data sets can issue the ROLB call. This backs out any print data sets that have not been released to JES.

Restrictions

The AIB must specify the I/O PCB for this call.

ROLL Call

The Roll (ROLL) call backs out output messages sent by a conversational application program and terminates the conversation. For more information on the ROLL call, see “Backing out to a Prior Commit Point: ROLL, ROLB, and ROLS Calls” on page 145.

Format

Call Name	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
ROLL	X	X	X	X	X

Parameters

The only parameter required for the ROLL call is the call function.

Usage

IMS terminates the application with a U0778 abend.

If you issue a ROLL call during a conversation, IMS TM backs out the update and cancels output messages. IMS TM also terminates the conversation with a U0778 abend code.

For applications that use the CPI Communications interface, the original transaction is discarded if it is classified by IMS as a discardable transaction.

Any remote LU 6.2 conversation transactions generated by a modified DL/I or CPI-C driven application program are deallocated with TYPE (ABEND_SVC).

Any application program that uses Spool API functions and creates print data sets can issue the ROLL call. This backs out any print data sets that have not been released to JES.

Related Reading: For information on discardable and non-discardable transactions see *IMS/ESA Application Programming: Design Guide*. For more information on LU 6.2, see *IMS/ESA Administration Guide: Transaction Manager*.

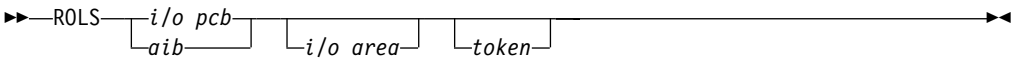
Restrictions

The ROLL call cannot use the AIBTDLI interface.

ROLS Call

The Roll Back to SETS/SETU (ROLS) call returns message queue positions to sync points established by the SETS/ SETU call. For more information on the ROLS and SETS/SETU calls, see “Backing out to a Prior Commit Point: ROLL, ROLB, and ROLS Calls” on page 145, and “SETS/SETU Call” on page 116).

Format



Call Name	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
ROLS	X	X	X	X	X

Parameters

i/o pcb
Specifies the I/O PCB to use for the call. This parameter is an input and output parameter.

aib
Specifies the application interface block (AIB) that is used for the call. This parameter is an input and output parameter.

The following fields must be initialized in the AIB:

AIBID
Eyecatcher. This 8-byte field must contain DFSAIBbb.

AIBLEN
AIB lengths. This field must contain the actual length of the AIB that the application program obtained.

AIBRSNM1
Resource name. This 8-byte, left-justified field must contain the PCB name IOPCBbbb.

AIBOALEN
I/O area length. This field must contain the length of the I/O area that is specified in the call list.

i/o area
Specifies the I/O area. It has the same format as the I/O area supplied on the SETS/SETU call. This parameter is an output parameter.

token
Specifies the name of the area in your program that contains a 4-byte identifier. This parameter is an input parameter.

Usage

Issuing a ROLS in a conversational program causes IMS TM to back out the messages that the application program has sent. For conversation transactions, this means that if the program issues a ROLS call and then reaches a commit point without sending the required response to the originating terminal, IMS TM terminates the conversation and sends the message DFS2171I NO RESPONSE, CONVERSATION TERMINATED to the originating terminal.

When you issue a ROLS call with a token and the messages to be rolled back include nonexpress messages that are processed by IMS TM, message queue repositioning might occur. The repositioning can include the initial message segment, and the original input transaction can be presented again to the IMS TM application program.

Input and output positioning is determined by the SETS/SETU call in standard and modified DL/I application programs. Input and output positioning does not apply to CPI-C driven application programs.

The application program must notify any remote transaction programs of the ROLS.

On a ROLS without a token, IMS issues the APPC/MVS verb, ATBCMTP TYPE(ABEND), specifying the transaction program instance (TPI). This causes all conversations associated with the application program to be DEALLOCATED TYPE(ABEND_SVC). If the original transaction was entered from an LU 6.2 device and IMS TM received the message from APPC/MVS, a discardable transaction is discarded. Nondiscardable transactions are placed on the suspend queue.

Related Reading: For more information on LU 6.2, see *IMS/ESA Administration Guide: Transaction Manager*.

Restrictions

When ROLS is issued during a conversational application program that includes resources outside of IMS TM (for example, a CPI-C driven application program), only the IMS TM resources are rolled back. The application program notifies the remote transactions of the ROLS call.

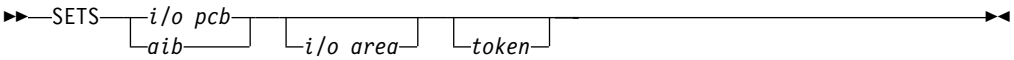
The Spool API functions do not restrict the use of the SETS/SETU and ROLS calls because these calls can be used by the application program outside the processing of print data sets. When these commands are issued, the Spool API takes no action because these commands cannot be used for the partial backout of print data sets. No special status codes are returned to the application program to indicate that the SETS/SETU or ROLS call was issued by an application that is using Spool API.

The ROLS call is not valid when the PSB contains a DEDB or MSDB PCB, or when the call is made to a DB2 database.

SETS/SETU Call

The Set Backout Point (SETS) call is used to set an intermediate backout point or to cancel all existing backout points. The Set Unconditional (SETU) call operates like the SETS call except that the SETU call isn't rejected if unsupported PCBs are in the PSB or if the program uses an external subsystem.

Format



Call Name	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
SETS/SETU	X	X	X	X	X

Parameters

i/o pcb
Specifies the I/O PCB to use for the call. This parameter is an input and output parameter.

aib
Specifies the application interface block (AIB) that is used for the call. This parameter is an input and output parameter.

The following fields must be initialized in the AIB:

AIBID
Eyecatcher. This 8-byte field must contain DFSAIB␣␣.

AIBLEN
AIB lengths. This field must contain the actual length of the AIB that the application program obtained.

AIBRSNM1
Resource name. This 8-byte, left-justified field must contain the PCB name IOPCB␣␣␣.

AIBOALEN
I/O area length. This field must contain the length of the I/O area that is specified in the call list.

i/o area
Specifies the area in your program that contains the data that is to be kept by IMS and returned on the corresponding ROLS call. This parameter is an input parameter.

token
Specifies the name of the area in your program that contains a 4-byte identifier. This parameter is an input parameter.

Usage

Except for the call names themselves, the SETS and SETU format and parameters are the same.

The SETS and SETU calls provide the backout points that IMS uses in the ROLS call. The ROLS call operates consistent with the SETS and SETU call backout points.

The meaning of the SC status code for SETS or SETU is as follows:

SETS The SETS call is rejected. The SC status code in the I/O PCB indicates that either the PSB contains unsupported options or the application program made calls to an external subsystem.

SETU The SETU call is not rejected. The SC status code indicates that unsupported PCBs exist in the PSB or the application made calls to an external subsystem.

Restrictions

The SETS call is not valid when the PSB contains a DEDB or MSDB PCB, or when the call is made to a DB2 database.

CPI-C driven transaction programs cannot issue the SETS/SETU call.

The Spool API functions do not restrict the use of the SETS/SETU and ROLS calls. This is so, because these calls can be used by the application outside the processing of print data sets. When these commands are issued, the Spool API takes no action because these commands cannot be used for the partial backout of print data sets.

SYNC Call

The Synchronization Point (SYNC) call is used to request commit point processing.

Format

Call Name	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
SYNC	X	X	X		

Parameters

i/o pcb
Specifies the I/O PCB to use for the call. This parameter is an input and output parameter.

aib
Specifies the application interface block (AIB) that is used for the call. This parameter is an input and output parameter.

The following fields must be initialized in the AIB:

AIBID
Eyecatcher. This 8-byte field must contain DFSAIB␣␣.

AIBLEN
AIB lengths. This field must contain the actual length of the AIB that the application program obtained.

AIBRSNM1
Resource name. This 8-byte, left-justified field must contain the PCB name IOPCB␣␣␣.

Usage

Issue the SYNC call to request that IMS TM process the application program with commit points for the application program.

Restrictions

The SYNC call is valid only in batch-oriented BMPs.

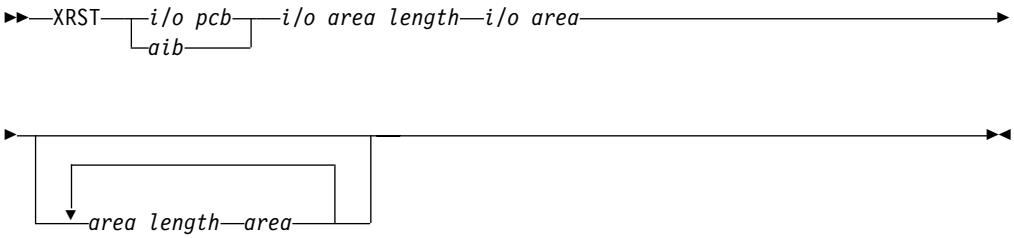
You cannot issue a SYNC call from a CPI Communications driven application program.

For important considerations about the use of the SYNC call, see *IMS/ESA Administration Guide: Database Manager*.

XRST Call

The Extended Restart (XRST) call is used to restart your program. If you use the symbolic Checkpoint call in your program, you must use the XRST call. For a description of the symbolic CHKP call see “CHKP (Symbolic) Call” on page 92.

Format



Call Name	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
XRST	X	X	X	X	X

Parameters

i/o pcb
Specifies the I/O PCB to use for this call. This parameter is an input and output parameter.

aib
Specifies the application interface block (AIB) that is used for the call. This parameter is an input and output parameter.

The following fields must be initialized in the AIB:

AIBID
Eyecatcher. This 8-byte field must contain DFSAIBbb.

AIBLEN
AIB lengths. This field must contain the actual length of the AIB that the application program obtained.

AIBRSNM1
Resource name. This 8-byte, left-justified field must contain the PCB name IOPCBbbb.

AIBOALEN
I/O area length. This field must contain the length of the I/O area that is specified in the call list. This parameter is not used during the XRST call. For compatibility reasons, this parameter must still be coded.

i/o area length
This parameter is no longer used by IMS. For compatibility reasons, this parameter must still be included in the call, and it must contain a valid address. You can get a valid address by specifying the name of any area in your program.

i/o area
Specifies a 30-byte area in your program. This area must be either set to blanks if starting your program normally or, if performing an extended restart, have a checkpoint ID.

area length
Specifies a 4-byte field in your program containing the length (in binary) of an area to restore. This input parameter is optional. You can specify up to seven area lengths. For each area length, you must also specify the area parameter.

The number of areas you specify on a XRST call must be less than or equal to the number of areas you specify on the CHKP calls the program issues. When you restart the program, IMS TM restores only the areas you specified in the CHKP call.

area
Specifies the area in your program that you want IMS TM to restore. You can specify up to seven areas. Each area specified must be preceded by an *area length* value. This parameter is an input parameter.

Usage

Programs that wish to issue Symbolic Checkpoint calls (CHKP) must also issue the Extended Restart call (XRST). The XRST call must be issued only once and should be issued early in the execution of the program. It does not need to be the first call in the program. However, it must precede any CHKP call. Any Database calls issued before the XRST call are not within the scope of a restart.

IMS determines whether to perform a normal start or a restart based on the I/O area provided by the XRST call or CKPTID= value in the PARM field on the EXEC statement in your program's JCL.

Starting Your Program Normally

When you are starting your program normally, the I/O area pointed to in the XRST call must contain blanks and the CKPTID= value in the PARM field must be nulls. This indicates to IMS that subsequent CHKP calls are symbolic checkpoints rather than basic checkpoints. Your program should test the I/O area after issuing the XRST call. IMS does not change the area when you are starting the program normally.

Restarting Your Program

You can restart the program from a symbolic checkpoint taken during a previous execution of the program. The checkpoint used to perform the restart can be identified by entering the checkpoint ID either in the I/O area pointed to by the XRST call (leftmost justified, with the rest of the area containing blanks) or by specifying the ID in the CKPTID= field of the PARM= parameter on the EXEC statement in your program's JCL. (If you supply both, IMS uses the CKPTID= value specified in the parm field of the EXEC statement.)

The ID specified can be:

- A 1 to 8-character extended checkpoint ID
- A 14-character ″time stamp″ ID from message DFS05401, where:
  IIII is the region ID
  DDD is the day of the year
  HHMMSST is the time in hours, minutes, seconds, and tenth of a second
- The 4-character constant ″LAST″. (BMPs only: this indicates to IMS that the last completed checkpoint issued by the BMP will be used for restarting the program)

The system message DFS05401 supplies the checkpoint ID and the time stamp.

The system message DFS6821 supplies the checkpoint ID of the last completed checkpoint which can be used to restart a batch program or batch message processing program (BMP) that was abnormally terminated.

If the program being restarted is in either a batch region or a BMP region, and the checkpoint log records no longer reside on the Online Log Data Set (OLDS) or System Log Data Set (SLDS), the //IMSLOGR DD defining the log data set must be

supplied in the JCL for the BATCH or BMP region. IMS reads these data sets and searches for the checkpoint records with the ID that was specified.

At the completion of the XRST call, the I/O area always contains the 8-character checkpoint ID used for the restart. An exception exists when the checkpoint ID is equal to 8 blank characters; the I/O area then contains a 14-character time stamp (IIIIDDDHHMMSST).

Also check the status code in the I/O PCB. The only successful status code for an XRST call are blanks.

Restrictions

If your program is being started normally, the first 5 bytes of the I/O area must be set to blanks.

If your program is restarted and the CKPTID= value in the PARM field of the EXEC statement is not used, then the rightmost bytes beyond the checkpoint ID being used in the I/O area must be set to blanks.

The XRST call is allowed only from Batch and BMP applications.

Chapter 5. More about Message Processing

This chapter explains additional message processing concepts and techniques that extend what IMS TM application programs can do. It also provides examples of message-driven program structure in assembler, C language, COBOL, Pascal, and PL/I.

In this Chapter:

- "Sending Messages to Other Terminals and Programs"
- "Communicating with Other IMS TM Systems Using MSC" on page 128
- "IMS Conversations" on page 131
- "Processing Conversations with APPC" on page 141
- "Processing Conversations with OTMA" on page 145
- "Backing out to a Prior Commit Point: ROLL, ROLB, and ROLS Calls" on page 145
- "Backing out to an Intermediate Backout Point: SETS/SETU and ROLS" on page 149
- "Writing a Message-Driven Program" on page 151
- "Coding DC Calls and Data Areas" on page 152

Sending Messages to Other Terminals and Programs

When an application program processes a message from a terminal, it usually sends the response to the terminal that sent the input message. But sometimes you might want to send output messages to a terminal other than the originating terminal, or to other terminals in addition to the originating terminal. You might also want to send messages to other application programs.

To send a message to a different terminal or to an application program, issue the `ISRT` call, but reference an alternate PCB instead of the I/O PCB. Alternate PCBs can be defined for a particular terminal or program, or they can be defined as modifiable. If the alternate PCB is not modifiable, only issue an `ISRT` call referencing the alternate PCB to send a message to the terminal or program that it represents. If the alternate PCB is modifiable, set the destination for the alternate PCB before issuing the `ISRT` call. To do this, use a `CHNG` call.

When you use an alternate PCB:

- If you want to send output messages to one alternate destination, define the alternate PCB for that destination.
- If you want to send output messages to more than one alternate destination, and you want to be able to change the destination of the alternate PCB, define the alternate PCB as modifiable during PSB generation. Then, before you issue the `ISRT` call, you issue a `CHNG` call to set the destination of the alternate modifiable PCB for the destination program or terminal.

  The *express alternate PCB* is a special kind of alternate PCB that is defined during PSB generation, by specifying EXPRESS=YES.

  When you use an express alternate PCB, messages you send using that PCB are sent to their final destinations immediately. Messages sent with other PCBs are sent to temporary destinations until the program reaches a commit point. Messages sent with express PCBs are sent if the program subsequently terminates abnormally, or issues one of the rollback calls: `ROLL`, `ROLB`, or `ROLS`.

© Copyright IBM Corp. 1974, 2000

Using an express alternate PCB in this kind of situation is a way to ensure that the program can notify the person at the terminal, even if abnormal termination occurs. For all PCBs, when a program abnormally terminates or issues a ROLL, ROLB, or ROLS call, messages inserted but not made available for transmission are cancelled, while messages made available for transmission are never cancelled.

For a nonexpress PCB, the message is not made available for transmission to its destination until the program reaches a commit point. The commit point occurs when the program terminates, issues a CHKP call, or requests the next input message and the transaction has been defined with MODE=SNGL.

For an express PCB, when IMS TM knows that it has the complete message, it makes the message available for transmission to the destination. In addition to occurring at a commit point, this also occurs when the application program issues a PURG call using that PCB or requests the next input message.

A PSBGEN can also specify an alternate PCB as an alternate response PCB defined during PSB generation.

- If you want to send a message to an LU 6.2 device, you can specify the LU 6.2 descriptor name that is associated with that device.

For more information on sending messages to alternate PCBs, see “Sending Messages to Other Terminals and Programs” on page 123.

Sending Messages to Other Terminals

To reply to a different terminal, also use the ISRT call, but use an alternate PCB instead of the I/O PCB.

Just as the I/O PCB represents the terminal that sent the message, an alternate PCB represents the terminal to which you want to send the message.

To One Alternate Terminal

If you are going to send messages to only one alternate terminal, you can define the alternate PCB for that terminal during PSB generation. When you define an alternate PCB for a particular destination, you cannot change that destination during program execution. Each time you issue an ISRT call that references that PCB, the message goes to the logical terminal whose name was specified for the alternate PCB. To send a message to that terminal, place one message segment at a time in the I/O area, and issue an ISRT call referring to the alternate PCB, instead of the I/O PCB.

To Several Alternate Terminals

To send messages to several terminals, you can define the alternate PCB as modifiable during PSB generation. Therefore, the alternate PCB represents more than one alternate terminal. You can change the destination while your program is running.

Before you can set or change the destination of an alternate PCB, you must indicate to IMS TM that the message you have been building so far with that PCB is finished. To do this, issue a PURG call.

PURG allows you to send multiple output messages while processing one input message. When you do not use PURG, IMS TM groups message segments into a message and sends them when the program issues a GU for a new message, terminates, or reaches a commit point. A PURG call tells IMS TM that the message built against this I/O PCB or alternate PCB (by issuing one ISRT call per message segment) is complete. IMS TM collects the message segments that you have

inserted to one PCB as one message and sends it to the destination represented by the alternate PCB you have referenced.

A PURG call that does not contain the address of an I/O area indicates to IMS TM that this message is complete. If you include an I/O area in the call, PURG acts as an ISRT call as well. IMS TM treats the data in the I/O area as the first segment of a new message. When you include an I/O area on a PURG call, you can also include a MOD name to change the format of the screen for this message. Although specifying the MOD name is optional, when you use it, you can specify it only once per message or in only the first ISRT or PURG that begins the message.

To set the destination of a modifiable alternate PCB during program execution, you use a CHNG call. When you issue the CHNG call you supply the name of the logical terminal to which you want to send the message. The alternate PCB you use then remains set with that destination until you do one of the following:

- Issue another CHNG call to reset the destination.
- Issue another GU to the message queue to start processing a new message. In this case, the name still appears in the alternate PCB, even though it is no longer valid.
- Terminate your program. When you do this, IMS TM resets the destination to blanks.

The first 8 bytes of the alternate PCB contain the name of the logical terminal to which you want to send the message.

When you issue a CHNG call, give IMS TM the address of the alternate PCB you are using and the destination name you want set for that alternate PCB.

When you use the PURG call, you give IMS TM only the address of the alternate PCB. IMS TM sends the message you have built using that PCB.

To indicate an error situation, you can send a message by issuing an ISRT call followed by a PURG call against an express PCB. These calls send the message to its final destination immediately.

Example: Suppose the program goes through these steps:

1. The program issues a GU call (and GN calls, if necessary) to retrieve an input message.
2. While processing the message, the program encounters an abnormal situation.
3. The program issues a PURG call to indicate to IMS TM the start of a new message.
4. The program issues a CHNG call to set the destination of an express PCB to the name of the originating logical terminal. The program can get this name from the first 8 bytes of the I/O PCB.
5. The program issues ISRT calls as necessary to send message segments. The ISRT calls reference the express PCB.
6. The program issues a PURG call referencing the express PCB. IMS TM then sends the message to its final destination.
7. The program can then terminate abnormally, or it can issue a ROLL, ROLB, or ROLS call to back out its database updates and cancel the output messages it has created since the last commit point.

If your output messages contained three segments, and you used the PURG call to indicate the end of a message (and not to send the next message segment), you could use this call sequence:

```
CHNG ALTPCB1, LTERMA
ISRT ALTPCB1, SEG1
ISRT ALTPCB1, SEG2
ISRT ALTPCB1, SEG3
PURG ALTPCB1
CHNG ALTPCB1, LTERMB
ISRT ALTPCB1, SEG4
ISRT ALTPCB1, SEG5
ISRT ALTPCB1, SEG6
```

Sending Messages to Other Application Programs

A program-to-program message switch occurs when one MPP sends a message to another online program (another MPP or a transaction-oriented BMP). To do this, use an alternate PCB and use some of the same options in an alternate PCB to send messages to alternate terminals. If you send messages to only one application program, then you can define the alternate PCB with the transaction code for that application program during PSB generation. If you send messages to more than one application program, you can define the alternate PCB as modifiable.

If you use an alternate modifiable PCB, IMS TM does some security checking when you issue the CHNG call to set the destination of the alternate modifiable PCB. The terminal that enters the transaction code that causes the message switch must be authorized to enter the transaction code that the CHNG call places in the alternate modifiable PCB. IMS TM does not do any security checking when you issue the ISRT call.

The security checking that is done in RACF when you issue a CHNG call for a program-to-program message switch is the same checking that is done in an environment that uses the Security Maintenance utility (SMU). When an IMS TM application program issues a CHNG call, that call invokes RACF, and a check is made to determine whether the originating terminal is authorized for the transaction code just issued. If, instead of using the CHNG call, the program issues an ISRT call against a preset alternate PCB, no security check is made, regardless of the environment.

What you consider when you do a program-to-program message switch is similar to what you consider when you communicate with a logical terminal. You have to remember to:

- Create an I/O area large enough to hold the largest segment that you are sending.
- Use an alternate PCB, not the I/O PCB, to send the message.
- Issue a CHNG call before the ISRT call to place the program's transaction code in the first field of the alternate PCB. If the alternate PCB was set to this transaction code in the PSBGEN, then you just issue the ISRT call.
- IMS TM must know the transaction code. Define it at system definition.
- A nonconversational program can do a program-to-program message switch to another nonconversational program, but not to a conversational program.
- A conversational program can do a program-to-program message switch to either another conversational program or a nonconversational program.

A message switch to another conversational program transfers the SPA and the responsibility to respond to the originating terminal to the new application program.

(See “Passing the Conversation to another Conversational Program” on page 137.) A message switch to a nonconversational program does not change the responsibilities of the conversational program. The conversational program must still return the SPA to IMS TM (if the SPA has been modified) and must respond to the originating terminal. Figure 17 shows the format for an output message to an application program.

For the AIBTDLI, ASMTDLI, CBLTDLI, CEETDLI, CTDLI and PASTDLI interfaces:

LL	Z1	Z2	Text
2	1	1	variable

For the PLITDLI interface:

LLLL	Z1	Z2	Text
4	1	1	variable

Figure 17. Message Format for Program-to-Program Message Switch

As you can see, the format is the same as for output messages to terminals. Z1 and Z2 are fields that must contain binary zeros. These fields are reserved for IMS. The text field contains the message segment that you want to send to the application program.

If the program that is processing the message expects the transaction code, include Program B’s transaction code as part of the message text of the message’s first segment, because IMS TM does not automatically include the transaction code in the first segment of a switched message. Including the transaction code in the first segment’s message text keeps the first segments of all messages in the same format, regardless of whether they are sent from terminals or other programs.

How the VTAM I/O Facility Affects Your VTAM Terminal

VTAM terminals can fail to respond to requests sent by IMS. The master terminal operator or an automated operator interface application program can optionally activate a “timeout” facility. This allows a message stating a specific amount of time has passed to be sent to the master terminal operator.

IMS TM can be set up to do one of the following:

- Do nothing, which means that your terminal remains inactive. This is the default.
- Send a message to the master terminal operator stating that the specified period of time has passed. The operator can then determine what action, if any, should be taken.
- Send a message to the master terminal operator stating that the specified period of time has passed.IMS TM then issues the VTAM `VARY NET,INACT` command followed by a VTAM `VARY NET,ACT` command. If the terminal is defined to IMS TM as non-shared and operable, and if IMS TM is not shutting down, IMS TM issues an OPNDST for the terminal.

  Restriction: This option does not apply to ISC terminals. If your installation chooses this option and an ISC terminal times out, a message is sent to the

master terminal stating that the specified period of time has passed. The operator can determine what action, if any, should be taken.

Communicating with Other IMS TM Systems Using MSC

In addition to communicating with programs and terminals in your IMS TM system, your program can communicate with terminals and programs in other IMS TM systems through Multiple Systems Coupling (MSC). MSC makes this possible by establishing links between two or more separate IMS TM systems. The terminals and transaction codes within each IMS TM system are defined as belonging to that system. Terminals and transaction codes within your system are called “local,” and terminals and transaction codes defined in other IMS TM systems connected by MSC links are called “remote.”

Related Reading: For an overview of MSC, see *IMS/ESA General Information*. For more detailed information, see *IMS/ESA Administration Guide: Transaction Manager*.

Implications of MSC for Program Coding

For the most part, communicating with a remote terminal or program does not affect how you code your program. MSC handles the message routing between systems.

Example: If you receive an input message from a remote terminal, and you want to reply to that terminal, you issue an `ISRT` call against the I/O PCB—just as you would reply to a terminal in your system.

In the following two situations, MSC might affect your programming:

- When your program needs to know whether an input message is from a remote terminal or a local terminal. For example, if two terminals in separate IMS TM systems had the same logical terminal name, your program’s processing might be affected by knowing which system sent the message.
- When you want to send a message to an alternate destination in another IMS TM system.

Restriction: If a transaction allocated by an LU 6.2 device is destined to a remote system through MSC links, IMS rejects the transaction with the message `TP_NOT_Avail_No_Retry`.

Related Reading: For more information on LU 6.2, see *IMS/ESA Administration Guide: Transaction Manager*.

Directed routing makes it possible for your program to find out whether an input message is from your system or from a remote system, and to set the destination of an output message for an alternate destination in another IMS TM system. With directed routing, you can send a message to an alternate destination in another IMS TM system, even if that destination is not defined in your system as remote.

Restriction: MSC directed routing does not support a program-to-program switch between conversational transactions.

Related Reading: For more information about MSC directed routing, see *IMS/ESA Administration Guide: System*.

Receiving Messages from Other IMS TM Systems

When an application program retrieves an input message, the program can determine whether the input message is from a terminal or program in its IMS TM system, or from a terminal or program in another IMS TM system. There might be situations in which the application program's processing is changed if the input message is from a remote terminal, rather than from a local terminal.

Example: Suppose that your IMS TM system is system A, and that it is linked to another IMS TM system called system B. MSC links are one-way links. The link from system A to system B is called LINK1, and the link from system B to system A is called LINK2. The application program named MPP1 runs in system A. The logical terminal name of the master terminals in both systems is MASTER. Figure 18 shows systems A and B.

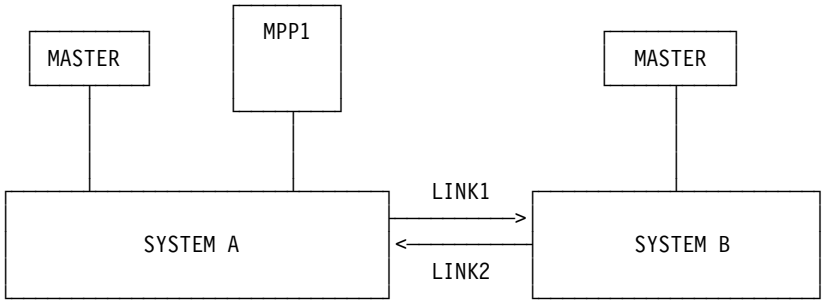


Figure 18. MSC Example

If the MASTER terminal in system B sends a message indicating that the system is shutting down to MPP1 in system A, MPP1 needs to know that the message is from MASTER in system B and not MASTER in system A.

If you have specified ROUTING=YES on the TRANSACT macro during IMS TM system definition, IMS TM does two things to indicate to the program that the message is from a terminal in another IMS TM system.

First, instead of placing the logical terminal name in the first field of the I/O PCB, IMS TM places the name of the MSC logical link in this field. In the example, this is LINK1. This is the logical link name that was specified on the MSNAME macro at system definition. However, if the message is subsequently sent back to the originating system, the originating LTERM name is reinstated in the first field of the I/O PCB.

Second, IMS TM turns on a bit in the field of the I/O PCB that is reserved for IMS. This is the second bit in the first byte of the 2-byte field. Figure 19 on page 130 shows the location of this bit within the reserved field.

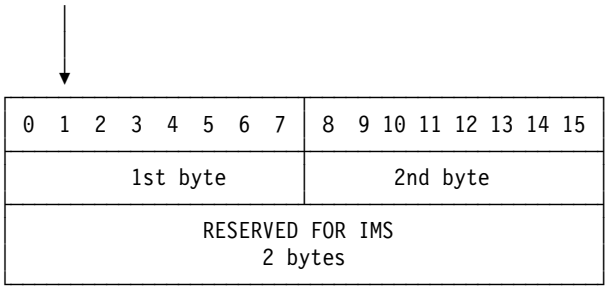


Figure 19. Directed Routing Bit in I/O PCB

MPP1 tests this bit to determine if the message is from MASTER in system A. If it is, MPP1 should terminate immediately. However, if the message is from MASTER in system B, MPP1 could perform some local processing and send transactions for system B to a message queue so that those transactions could be processed later on, when system B is up.

Sending Messages to Alternate Destinations in Other IMS TM Systems

To send an output message to an alternate terminal in another IMS TM system, your system must have an MSC link with the system to which you want to send the message. To do this, issue a `CHNG` call against an alternate PCB and supply the name of the MSC link (in the example this is LINK1) that connects the two IMS TM systems.

Example: If you were sending a message to TERMINAL 1 in system B after you received a message from some other terminal, you would first issue this `CHNG` call:

```
CHNG  altpcb, LINK1
```

Then issue an `ISRT` call (or calls) to send the message just as you would send a message to a local terminal.

Figure 20 shows the format of this type of output message.

For the AIBTDLI, ASMTDLI, CBLTDLI, CEETDLI, CTDLI and PASTDLI interfaces:

LL	ZZ	DESTNAME	b	TEXT
2	2	1 — 8	1	variable

For the PLITDLI interface:

LLLL	ZZ	DESTNAME	b	TEXT
4	2	1 — 8	1	variable

Figure 20. Directed Routing Output Message Format

The formats of the fields in a directed routing output message are:

- The LL and ZZ fields are 2 bytes each (For the PLITDLI interface, use the 4-byte field LLLL). LL (or LLLL) contains the total length of the message. This is the

sum of all of the fields in the message, including the LL field (and in PL/I, LLLL contains the total length minus 2). ZZ is reserved for IMS.

- The destination name, DESTNAME, is the name of the logical terminal to which you are sending the message. This field is from 1 to 8 bytes long and *it must be followed by a blank*.

  If the destination in the other system is a terminal, IMS TM removes the DESTNAME from the message. If the destination in the other system is a program, IMS TM does not remove the DESTNAME.
- The TEXT field contains the text of the message. Its length depends on the message you are sending.

If your message contains a security violation, MSC detects it in the receiving system (in this case, system B), and reports it to the person at the originating terminal (system A).

IMS Conversations

Definitions:

- A *conversational program* is an MPP that processes transactions made up of several steps. It does not process the entire transaction at the same time. A conversational program divides processing into a connected series of terminal-to-program-to-terminal interactions. You use conversational processing when one transaction contains several parts.
- A *nonconversational program* receives a message from a terminal, processes the request, and sends a message back to the terminal. A conversational program receives a message from a terminal, and replies to the terminal, but saves the data from the transaction in a scratchpad area (SPA). Then, when the person at the terminal enters more data, the program has the data it saved from the last message in the SPA, so it can continue processing the request without the person at the terminal having to enter the data again.

A Conversational Example

For this example, suppose that you want to find out if someone can qualify for a car loan. This inquiry contains two parts. First, you give the name and address of the person requesting the loan and the number of years for which the person wants the loan. After you give this information, IMS TM asks you for the information on the car: model, year, and cost. You enter this information, IMS TM invokes the program that processes this information, and the program tells you whether the loan can be granted.

If you use MFS, the process involves these steps:

1. Enter the format command (/FORMAT) and the MOD name. This tells IMS to format the screen in the way defined by this MOD.

   If the MOD name is CL, the command is:

   ```
   /FORMAT CL
   ```

   IMS TM then takes that MOD from the MFS library and formats your screen in the way defined by the MOD. When the MOD for the car loan application formats your screen, it looks like this:

   ```
   CARLOAN
   NAME:
   ADDRESS:
   YEARS:
   ```

The word “CARLOAN” is the transaction code for this application. Each transaction code is associated with an application program, so when IMS TM receives the transaction code “CARLOAN,”IMS TM knows what application program to schedule for this request.

2. Enter the customer’s name and address, and the length of the loan. When you enter this information, your screen looks like this:

```
CARLOAN
NAME:    JOHN EDWARDS
ADDRESS: 463 PINEWOOD
YEARS:   5
```

3. IMS TM reads the transaction code, CARLOAN, and invokes the program that handles that transaction code. MFS formats the information from the screen for the MPP’s I/O area by using the DIF and the MID.

   When the MPP issues its first call, which is usually a `GU` for the SPA, IMS TM clears the SPA to binary zeros and passes it to the application program.

4. Next, the MPP processes the input data from the terminal and does two things. It moves the data that it will need to save to the SPA, and it builds the output message for the terminal in the I/O area. The information that the MPP saves in the SPA is the information the MPP will need when the second part of the request comes in from the terminal. You do not save information in the SPA that you can get from the database. In this example, you save the name of the customer applying for the loan, because if the customer is granted the loan, the program uses the customer name to locate the information to be updated in the database.

   The program then issues an `ISRT` call to return the SPA to IMS, and another `ISRT` call to send the output message to the terminal.

   The response that the MPP sends to the terminal gives IMS TM the name of the MOD to format the screen for the next cycle of the conversation. In that cycle, you need to supply the model, year, and cost of the car that John Edwards wants to buy. Your screen looks like this:

```
MODEL:
YEAR:
COST:
```

5. IMS TM again uses the DIF and MID associated with the transaction code, and sends the information back to the MPP. The MPP has not been running all this time. when IMS TM receives the terminal input with the transaction code CARLOAN, IMS TM invokes the MPP that processes that transaction again for this cycle of the conversation.
6. IMS TM returns the updated SPA to the MPP when the MPP issues a `GU`, then returns the message to the MPP when the MPP issues a `GN`. The MPP does the required processing (in this case, determining whether the loan can be granted and updating the database if necessary), and is then ready to end the conversation. To do this, the MPP blanks out the transaction code in the SPA, inserts it back to IMS, then sends a message to the terminal saying whether the loan can be granted.

Conversational Structure

Structuring your conversational program depends on the interactions between your program and the person at the terminal. To understand what conversational processing involves, see “IMS Conversations” on page 131.

Before structuring your program, you need to know:

- ***What should the program do in an error situation?***

When a program in a conversation terminates abnormally, IMS TM backs out only the last cycle of the conversation. A cycle in a conversation is one terminal/program interaction. Because the conversation can terminate abnormally during any cycle, you should be aware of some things you can do to simplify recovery of the conversation:

- The ROLB or ROLS call can be used in conversational programs to back out database updates that the program has made since the last commit point. ROLL can also be used in conversational programs, but terminates the conversation. "Using ROLB, ROLL, and ROLS in Conversations" on page 137 explains how these calls work with conversational processing.
- If possible, updating the database should be part of the last cycle of the conversation so that you do not have different levels of database updates resulting from the conversation.
- If your program encounters an error situation and it has to terminate, it can use an express alternate PCB to send a message to the originating terminal, and, if desired, to the master terminal operator.

  To do this, the program issues a CHNG call against the express alternate PCB and supplies the name of the logical terminal from the I/O PCB, then an ISRT call that references that PCB and the I/O area that contains the message. The program can then issue another CHNG call to set the destination of the express alternate PCB for the master terminal, and another ISRT call that references that PCB, and the I/O area that contains the output message.

- ***Does your application program process each cycle of the conversation?***

  A conversation can be processed by one or several application programs. If your program processes each stage of the conversation (in other words, your program processes each input message from the terminal), the program has to know what stage of the conversation it is processing when it receives each input message.

  When the person at the terminal enters the transaction code that starts the conversation, IMS TM clears the SPA to binary zeros and passes the SPA to the program when the program issues a GU call. On subsequent passes, however, the program has to be able to tell which stage of the conversation it is on so that it can branch to the section of the program that handles that processing.

  One technique that the program can use to determine which cycle of the conversation it is processing is to keep a counter in the SPA. The program increments this counter at each stage of the conversation. Then, each time the program begins a new cycle of the conversation (by issuing a GU call to retrieve the SPA), the program can check the counter in the SPA to determine which cycle it is processing, then branch to the appropriate section.

- ***How can your program pass control of the conversation to another conversation program?***

  Sometimes it is more efficient to use several application programs to process a conversation. This does not affect the person at the terminal. It depends on the processing that is required.

  In the car loan example, one MPP could process the first part of the conversation (processing the name, address, and number of years), and another MPP could process the second part of the conversation (processing the data about the car and responding with the status of the loan).

  A program can:

  - Reply to the originating terminal using a *deferred program switch*.

– Pass the SPA (and, optionally, a message) to another conversational program without responding to the terminal using an *immediate program switch*. In this case, it is the next program's responsibility to respond to the originating terminal.

  Definitions:

  - A *deferred program switch* responds to the terminal but causes the next input from the terminal to go to another conversational program.
  - An *immediate program switch* passes the conversation directly to another conversational program.

A conversational program must:

1. Retrieve the SPA and the message using `GU` and `GN` calls.

   If your MPP is starting this conversation, test the variable area of the SPA for zeros to determine if this is the beginning of the conversation. If the SPA does not contain zeros, it means that you started the conversation earlier and that you are now at a later stage in the conversation. If this is true, you would branch to the part of your program that processes this stage of the conversation to continue the conversation.

   If another MPP has passed control to your MPP to continue the conversation, the SPA contains the data you need to process the message, so you do not have to test it for zeros. Start processing the message immediately.
2. Process the message, including handling any necessary database access.
3. Send the output message to the terminal by using an `ISRT` call against the I/O PCB. This step can follow step 4.
4. Store the data (that your program, or the program that you pass control to, needs to continue processing) in the SPA using an `ISRT` call to the I/O PCB. (This step can precede step 3.) IMS TM determines which segment is the SPA by examining the ZZZZ field of the segment shown in Figure 21 on page 135.

To end the conversation, move blanks to the area of the SPA that contains the transaction code, and then insert the SPA back to IMS TM by issuing an `ISRT` call and referencing the I/O PCB.

If your MPP passes the conversation to another conversational program, the steps after the program processes the message are somewhat different. "Passing the Conversation to another Conversational Program" on page 137 explains this.

Also, your program should be designed to handle the situation that occurs when the first `GU` call to the I/O PCB does not return a message to the application program. This can happen if the person at the terminal cancels the conversation by entering the `/EXIT` command before the program issues a `GU` call. (This happens if the message from this terminal was the only message in the message queue for the program.)

What the SPA Contains

The SPA that IMS TM gives your program when you issue a `GU` contains the four parts shown in Figure 21 on page 135.

For the AIBTDLI, ASMTDLI, CBLTDLI, CEETDLI, CTDLI and PASTDLI interfaces:

LL	ZZZZ	TRANCODE	User Work Area
2	4	8	variable

For the PLITDLI interface:

LLLL	ZZZZ	TRANCODE	User Work Area
4	4	8	variable

Figure 21. SPA Format

The SPA format fields are:

LL or LLLL
A length field that gives the total length of the SPA. This length includes 2 bytes for the LL field. (For the PLITDLI interface, use a 4-byte field. Its contents include 4 bytes for LLLL, minus 2.)

ZZZZ
A 4-byte field reserved for IMS TM that your program must not modify.

TRANCODE
The 8-byte transaction code for this conversation.

User Work Area
A work area that you use to save the information that you need to continue the conversation. The length of this area depends on the length of the data you want to save. This length is defined at system definition.

When your program retrieves the SPA with a GU to start the conversation, IMS TM removes the transaction code from the message. In your first message segment you receive only the data from the message that the person at the terminal entered.

The following list indicates the ways that an application program processes the SPA. The program must:

- Not modify the first 6 bytes of the SPA (LL and ZZZZ). IMS TM uses these fields to identify the SPA.

  If the program modifies the SPA, the program must return the SPA to IMS TM (or, for a program switch, to the other program).
- Not return the SPA to IMS TM more than once during one cycle of the conversation.
- Not insert the SPA to an alternate PCB that represents a nonconversational transaction code or a logical terminal. The program can use an alternate response PCB if it represents that same physical terminal as the originating logical terminal.

Restriction: If you are using MFS, the IMS TM does not always remove the transaction code.

What Messages Look Like in a Conversation

Because the first segment contains the SPA, conversational input messages are made up of at least two segments. The input message starts in the second message segment.

The input message segment in a conversation contains only the data from the terminal. During the first step in the conversation, IMS TM removes the transaction code from the input message and places it in the SPA. When the program issues the first `GU`, IMS TM returns the SPA. To retrieve the first message segment, the program must issue a `GN`.

The format for the output messages that you send to the terminal is no different than the format for output messages in nonconversational programs.

Saving Information in the SPA

After you have processed the message and are ready to reply to the terminal, you can save the necessary data in the SPA. The part of the SPA in which you save data is the work area portion. Use the `ISRT` call to save data to the work area. This is a special use of the `ISRT` call, because you are not sending the SPA to a terminal, but rather saving it for future use.

If your program processes each stage of the conversation, you just issue an `ISRT` call to the I/O PCB and give the name of the I/O area that contains the SPA. For example:

```
ISRT   I/O PCB, I/O AREA
```

This returns the updated SPA to IMS TM so that IMS TM can pass it to your program at the next cycle of the conversation.

If you do not modify the SPA, you do not need to return it to IMS. However, the SPA will be passed by IMS TM to your program at the next cycle of the conversation.

Replying to the Terminal

For a conversation to continue, the originating terminal must receive a response to each of its input messages. The person at the terminal cannot enter any more data to be processed (except IMS TM commands) until the response has been received at the terminal.

To continue the conversation, the program must respond to the originating terminal by issuing the required `ISRT` calls to send the output message to the terminal. To send a message to the originating terminal, the `ISRT` calls must reference either the I/O PCB or an alternate response PCB. Use an alternate response PCB in a conversation when the terminal you are responding to has two components—for example, a printer and a punch—and you want to send the output message to a component that is separate from the component that sent the input message. If the program references an alternate response PCB, the PCB must be defined for the same physical terminal as the logical terminal that sent the input message.

The program can send only one output message to the terminal for each input message. Output messages can contain multiple segments, but the program cannot use the `PURG` call to send multiple output messages. If a conversational program issues a `PURG` call, IMS TM returns an AZ status code to the application program and does not process the call.

Using ROLB, ROLL, and ROLS in Conversations

Issuing a ROLB or ROLS in a conversational program causes IMS TM to back out the messages that the application program has sent. This means that, if the program issues a ROLB or ROLS and then reaches a commit point without sending the required response to the originating terminal, IMS TM terminates the conversation and sends the message DFS2171I NO RESPONSE CONVERSATION TERMINATED. to the originating terminal.

If you issue ROLL during a conversation, IMS TM backs out the updates and cancels output messages, but it also terminates the conversation.

Passing the Conversation to another Conversational Program

A conversational program can pass the conversation to another conversational program in two ways:

- A deferred switch.

  The program can respond to the terminal but cause the next input from the terminal to go to another conversational program by:

  – Issuing an ISRT call against the I/O PCB to respond to the terminal
  – Placing the transaction code for the new conversational program in the SPA
  – Issuing an ISRT call referencing the I/O PCB and the SPA to return the SPA to IMS TM

  IMS TM then routes the next input message from the terminal to the program associated with the transaction code that was specified in the SPA. Other conversational programs can continue to make program switches by changing the transaction code in the SPA.

- An immediate switch.

  The program can pass the conversation directly to another conversational program by issuing an ISRT call against the alternate PCB that has its destination set to the other conversational program.

  The first ISRT call must send the SPA to the other program, but the program passing control can issue subsequent ISRT calls to send a message to the new program. If the program does this, in addition to routing the SPA to the other conversational program, IMS TM updates the SPA as if the program had returned the SPA to IMS. If the program does an immediate switch, the program cannot also return the SPA to IMS TM or respond to the original terminal.

Restrictions on Passing the Conversation

The following restrictions apply to passing the conversation to another conversational program:

- When an immediate program switch occurs and the MPP receives an XE status code, the program attempts to insert the SPA to an alternate express PCB. Remove the EXPRESS=YES option from the PCB or define and use another PCB that is not express. This restriction prevents the second transaction from continuing the conversation if the first transaction abends after inserting the SPA.

  The person at the terminal can issue the /SET CONV XX command, where XX is the program that is to be scheduled in order to process the next step of the conversation.

- The SPA size for a conversational program-to-program switch on a remote MSC system also has restrictions when the source system (where the inputting terminal resides) or an intermediate MSC system is IMS/ESA Version 5 or earlier:

- When the ISRT occurs in the local IMS/ESA Version 5 system, conversational program-to-program switches can occur to a transaction with a SPA that is larger than, smaller than, or equal to the SPA size of the current transaction.
- If the SPA ISRT is on a remote MSC system, and is going back to the inputting terminal on the source IMS system, the SPA must be smaller than or equal to the SPA size of the current transaction.
- If the SPA ISRT is on a remote MSC system, and the destination is a transaction, the SPA must be equal in size to the SPA of the current transaction.

Defining the SPA Size

Define the SPA size with the TRANSACT macro. An option to capture truncated data is also defined with the TRANSACT macro. The format is:

```
TRANSACT SPA=(size,STRUNC|RTRUNC)
```

The default is to support truncated data (STRUNC). When a conversation is initially started, and on each program switch, the truncated data option is checked and set or reset as specified. When the truncated data option is set, it remains set for the life of the conversation, or until a program switch occurs to a transaction that specifies that the option be reset.

Example: Assume you have three transactions defined as follows:

```
TRANA SPA=100
TRANB SPA=050
TRANC SPA=150
```

For TRANC to receive the truncated data (which is the second 50 bytes from TRANA that TRANB does not receive) from TRANA, one of the following sets of specifications can be used:

- TRANA - STRUNC or none, TRANB - STRUNC or none, TRANC - STRUNC or none
- TRANA - RTRUNC, TRANB - STRUNC, TRANC - STRUNC or none

Conversational Processing and MSC

If your installation has two or more IMS TM systems, and they are linked to each other through MSC, a program in one system can process a conversation that originated in another system.

- If a conversational program in system A issues an ISRT call that references a response alternate PCB in system B, system B does the necessary verification. This is because the destination is implicit in the input system. The verification that system B does includes determining whether the logical terminal that is represented by the response alternate PCB is assigned to the same physical terminal as the logical terminal that sent the input message. If it is not, system B (the originating system) terminates the conversation abnormally without issuing a status code to the application program.
- Suppose program A processes a conversation that originates from a terminal in system B. Program A passes the conversation to another conversational program by changing the transaction code in the SPA. If the transaction code that program A supplies is invalid, system B (the originating system) terminates the conversation abnormally without returning a status code to the application program.

- When the source system (where the inputting terminal resides) is IMS/ESA Version 5 or earlier, the SPA size for a conversational program-to-program switch has restrictions. For more information, see “Restrictions on Passing the Conversation” on page 137.

Ending the Conversation

To end the conversation, a program blanks out the transaction code in the SPA and returns it to IMS TM by issuing an `ISRT` call and referencing the I/O PCB and the SPA. This terminates the conversation as soon as the terminal has received the response.

The program can also end the conversation by placing a nonconversational transaction code in the transaction field of the SPA and returning the SPA to IMS. This causes the conversation to remain active until the person at the terminal has entered the next message. The transaction code will be inserted from the SPA into the first segment of the input message. IMS TM then routes this message from the terminal to the MPP or BMP that processes the transaction code that was specified in the SPA.

In addition to being ended by the program, a conversation can be ended by the person at the originating terminal, the master terminal operator, and IMS.

- The person at the originating terminal can end the conversation by issuing one of several commands:

  /EXIT The person at the terminal can enter the `/EXIT` command by itself, or the `/EXIT` command followed by the conversational identification number assigned by the IMS TM system.

  /HOLD The `/HOLD` command stops the conversation temporarily to allow the person at the terminal to enter other transactions while IMS TM holds the conversation. When IMS TM responds to the `/HOLD` command, it supplies an identifier that the person at the terminal can later use to reactivate the conversation. The `/RELEASE` command followed by this identifier reactivates the conversation.

- **/START LINE**. The master terminal operator can end the conversation by entering a `/START LINE` command (without specifying a PTERM) or `/START NODE` command for the terminal in the conversation or a `/START USER` command for a signed-off dynamic user in conversation.
- IMS TM ends a conversation if, after the program successfully issues a `GU` call or an `ISRT` call to return the SPA, the program does not send a response to the terminal. In this situation, IMS TM sends the message `DFS2171I NO RESPONSE, CONVERSATION TERMINATED` to the terminal. IMS TM then terminates the conversation and performs commit point processing for the application program.

Message Switching in APPC Conversations

With the system service DFSAPPC, you can transfer messages between separate LU 6.2 devices and between an LU 6.2 device and another terminal supported by IMS TM. Message delivery with DFSAPPC is asynchronous, so messages are held on the IMS TM message queue until they can be delivered.

To send a message with DFSAPPC, specify the logical terminal name of an IMS TM terminal or the Transaction Program (TP) name of an LU 6.2 device.

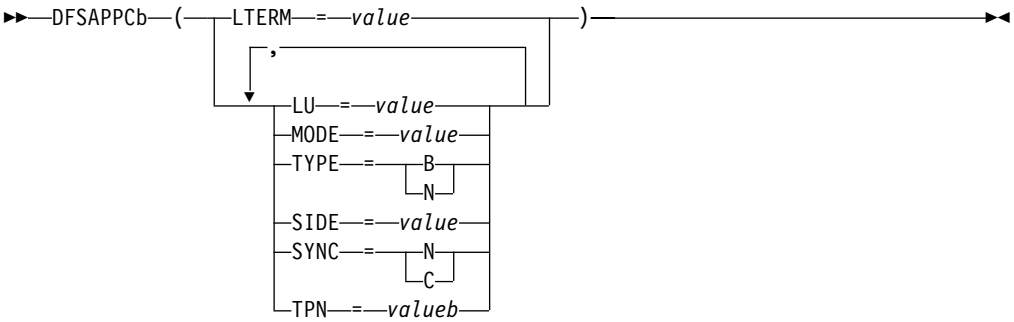
Related Reading: For more information on APPC and LU 6.2, see *IMS/ESA Administration Guide: Transaction Manager*.

DFSAPPC Format

The message format for DFSAPPC is as follows:

```
DFSAPPC (options)user_data
```

DFSAPPC can be coded as follows:



A blank (b) is required between DFSAPPC and the specified options.

Blanks are valid within the specified options except within keywords or values. Either commas or blanks can be used as delimiters between options, but because the use of commas is valid, the TP name must be followed by at least one blank.

If an LU 6.2 conversation has not been established from other sources (for example, during a CPI-C driven application program), DFSAPPC is used to establish the conversation with a partner LU 6.2 device. If no options are specified with DFSAPPC, IMS TM default options are used.

Option Keywords

LTERM=

Specifies the LTERM name of an IMS TM logical terminal. An LTERM name can contain up to eight alphanumeric or national (@, $, #) characters. If you specify LTERM, you cannot specify the other option keywords.

LU=

Specifies the LU name of the partner in an LU 6.2 conversation. The LU name can contain up to eight alphanumeric or national characters, but the first character must be a letter or a national character. If both LU and SIDE options are specified, LU overrides the LU name contained in the side information entry but does not change that LU name.

If the LU name is a network-qualified name, it can be up to 17 characters long and consist of the network ID of the originating system, followed by a '.', and the LU name (for example, netwrkid.luname). The LU name and the network ID can be up to eight characters long.

MODE=

Specifies the MODE name of the partner in an LU 6.2 conversation. The MODE name can contain up to eight alphanumeric or national characters, but the first character must be a letter or a national character. If both MODE and SIDE option keywords are specified, MODE overrides the MODE name contained in the side information entry but does not change that MODE name.

TPN=

Specifies the transaction program (TP) name of the partner in an LU 6.2

conversation. The TP name can contain up to 64 characters from the 00640 character set. Because the character set allows commas, at least one blank must follow the TP name. If both TPN and SIDE option keywords are specified, TPN overrides the TP name contained in the side information entry but does not change that name.

Related Reading: The *Common Programming Interface Communications Reference* describes the 00640 character set, which contains all alphanumeric and national characters and 20 special characters.

SIDE=
Specifies the name of the side information entry for the partner in an LU 6.2 conversation. The side information entry name can contain up to eight characters from the 01134 character set. If the SIDE option keyword is specified, it can be overridden with LU, MODE, and TPN option keywords.

Related Reading: The *Common Programming Interface Communications Reference* describes the 01134 character set, which contains the uppercase alphabet and the digits, 0-9.

SYNC=N|C
Specifies the synchronization level of the LU 6.2 conversation. N selects none as the synchronization level, and C selects confirm as the synchronization level.

TYPE=B|M
Specifies the conversation type for the LU 6.2 conversation. B selects a basic conversation type, and M selects a mapped conversation type.

Processing Conversations with APPC

APPC/IMS supports three different types of application programs:

- Standard: No explicit use of CPI Communications facilities.
- Modified: Uses the I/O PCB to communicate with the original input terminal. Uses CPI Communications calls to allocate new conversations and to send and receive data.
- CPI Communications driven: Uses CPI Communications calls to receive the incoming message and to send a reply on the same conversation. Uses the DL/I APSB call to allocate a PSB to access IMS databases and alternate PCBs.

In the modified or CPI Communications driven application programs, if an APPC conversation is allocated with SYNCLVL=SYNCPT, MVS manages the sync-point process for the APPC conversation participants: the application program and IMS. Transaction rollback and rescheduling is possible, because IMS issues the `SRRCMIT` or `SRRBACK` calls on behalf of the modified IMS APPC application program. If the CPI-C driven program is linked with the IMS stub code, DFSCPIR0, as required in previous releases, then IMS will also issue the SRRCMIT or SRRBACK calls. If the program is not linked with the stub code, then IMS is driven by the MVS sync point manager when the application issues these calls. With MVS as the sync point manager, failures can also be backed out.

You can schedule your standard and modified application programs locally and remotely using MSC or APPC/MVS. The logic flow for local scheduling differs from the logic flow for remote scheduling.

Scheduling programs remotely through MSC is not supported if an APPC/MVS conversation with SYNCLVL=SYNCPT is specified. In the following sections, the differences are described.

Related Reading: For both general information on LU 6.2 and APPC, and LU 6.2 flow diagrams, see *IMS/ESA Application Programming: Design Guide*.

Standard IMS Application Programs

Standard IMS application programs use the existing IMS call interface. Application programs that use the IMS standard API can take advantage of the LU 6.2 protocols. Standard IMS application programs use a DL/I `GU` call to get the incoming transaction. These standard IMS application programs also use DL/I `ISRT` calls to generate output messages to the same or different terminals, regardless of whether LU 6.2 is used.[2] The identical program can work correctly for both LU 6.2 and non-LU 6.2 terminal types. IMS generates the appropriate calls to APPC/MVS services.

Standard IMS Application Programs and MSC

When an APPC application program enters an IMS transaction that executes on a remote IMS, an LU 6.2 conversation is established between the APPC application program and the local IMS system. The local IMS is considered the partner LU of the LU 6.2 conversation. The transaction is then queued on the remote transaction queue of the local IMS system. From this point on, the transaction goes through normal MSC processing. After the remote IMS system executes the transaction, the output is returned to the local IMS system and is then delivered to the originating LU 6.2 application program.

Modified IMS Application Programs

Modified IMS application programs use a DL/I `GU` call to get the incoming transaction. These modified IMS application programs also use DL/I `ISRT` calls to generate output messages to the same or different terminals, regardless of whether LU 6.2 is used.[3] Unlike standard IMS application programs, modified IMS application programs use CPI Communications calls to allocate new conversations, and to send and receive data. IMS has no direct control of these CPI Communications conversations.

Modified IMS transactions are indistinguishable from standard IMS transactions until program execution. In fact, the same application program can be a standard IMS application on one execution, and a modified IMS application on a different execution. The distinction is simply whether the application program uses CPI Communications resources.

Modified IMS programs are scheduled by IMS TM, and the DL/I calls are processed by the DL/I language interface. The conversation, however, is maintained by APPC/MVS, and any failures that involve APPC/MVS are not backed out by IMS TM. The general format of a modified IMS application program is shown in Figure 22.

Related Reading: For more information on failure recovery and modified DL/I application program design, see *IMS/ESA Application Programming: Design Guide*.

2. A non-message-driven BMP is considered a standard IMS application program when it does not use the explicit API.

3. A non-message-driven BMP is considered a modified standard IMS application program when it uses the explicit API.

```
GU    IOPCB

        ALLOCATE
        SEND
        RECEIVE
        DEALLOCATE
      ISRT IOPCB
```

Figure 22. General Format of a Modified DL/I Application Program

Restriction: The APPC conversation cannot span sync points. If the conversation is not deallocated before a sync point is reached, IMS causes the conversation to be terminated by issuing a clean TP call (ATBCMTP). A new APPC conversation can be allocated after each sync point.

Modified IMS Application Programs and MSC

When an APPC program enters an IMS transaction that executes on a remote IMS system, an LU 6.2 conversation is established between the APPC program and the local IMS system. The local IMS system is considered the partner LU of the LU 6.2 conversation. The transaction is then queued on the local IMS system's remote transaction queue. From this point on, the transaction goes through normal MSC processing. After the remote IMS system executes the transaction, the output is returned to the local IMS and is then delivered to the originating LU 6.2 program.

CPI-C Driven Application Programs

CPI Communications driven application programs are defined only in the APPC/MVS TP_Profile data set; they are not defined to IMS. Their definition is dynamically built by IMS when a transaction is presented for scheduling by APPC/MVS, based on the APPC/MVS TP_Profile definition after IMS restart. The definition is keyed by TP name. APPC/MVS manages the TP_Profile information.

When a CPI Communications driven transaction program requests a PSB, the PSB must already be defined to IMS via the APPLCTN macro for sysgen and via PSBGEN or ACBGEN when APPLCTN PSB= is specified. When APPLCTN GPSB= is specified, a PSBGEN or ACBGEN is not required.

CPI-C driven application programs must begin with the CPI-C verbs, ACCEPT and RECEIVE, to initiate the LU 6.2 conversation. You can then issue the `APSB` call to allocate a PSB for use by the application program. After the `APSB` call is issued, you can issue additional DL/I calls using the PCBs that were allocated. You then issue the SRRCMIT verb to commit changes or the SRRBACK verb to back out changes. To use SRRCMIT and SRRBACK, your application program must be linked with DFSCPIR0.

Restriction: The I/O PCB cannot be used for message processing calls by CPI-C driven application programs. See the description of each call for specific CPI restrictions.

To deallocate the PSB in use, issue the `DPSB` call. You can then issue another `APSB` call, or use the CPI-C verb, DEALLOCATE, to end the conversation.

CPI-C driven application programs are considered discardable (unless they are allocated with a SYNCLVL=SYNCPT) by IMS TM and are therefore not recovered automatically at system failure. If they are allocated with a SYNCLVL=SYNCPT, a two-phase commit process is used to recover from any failures. The general format of a CPI-C driven application program is shown in Figure 23.

Related Reading: For more information on recovery procedures and CPI-C driven application program design, see *IMS/ESA Application Programming: Design Guide*.

```
  ACCEPT
  RECEIVE
APSB

GU DBPCB
REPL DBPCB
  SRRCMIT
DPSB

  DEALLOCATE
```

Figure 23. General Format of a CPI-C Driven Application Program

Restriction: The APPC conversation cannot span sync points. If the conversation is not deallocated before a sync point is reached, IMS causes the conversation to be terminated by issuing a clean TP call (ATBCMTP). A new APPC conversation can be allocated after each sync point.

Ending the APPC Conversation

The two ways to end a conversation using LU 6.2 devices are:

- Issuing the CPI-C verb, DEALLOCATE
- For IMS conversational transactions, inserting a blank transaction code into the SPA

Restriction: You cannot use the `/EXIT` command for LU 6.2 conversations.

Several error conditions can exist at the end of an LU 6.2 conversation:

- If your application program sends data to the LU 6.2 device just before deallocating conversation, IMS TM issues a SENDERROR and SENDDATA of the `DFS1966` error message. This indicates that the transaction ended, but that the last message could not be delivered. For SENDERROR to be activated, specify a synchronization level of CONFIRM.
- If IMS TM encounters an error sending output from an IMS TM conversational transaction to the LU 6.2 device, the output is discarded, and the conversation is terminated for both IMS TM and LU 6.2.
- If an IMS TM conversational application program abends during an LU 6.2 conversation, a DFS555 error message is sent to the originating LU 6.2 device, and the conversation is terminated for both IMS TM and LU 6.2.

Related Reading: For more information on LU 6.2, see *IMS/ESA Administration Guide: Transaction Manager*.

Coding a Conversational Program

Before coding a conversational program, obtain the following:

- The transaction code to use for a program to which you pass control
- The data that you should save in the SPA
- The maximum length of that data

A SPA contains four fields:

- The 2-byte length field.
- The 4-byte field that is reserved for IMS TM.
- The 8-byte transaction code.

- The work area where you store the conversation data. The length of this field is defined at system definition.

Processing Conversations with OTMA

You can run IMS conversational transactions through OTMA. Refer to *IMS/ESA Open Transaction Manager Access Guide*.

Backing out to a Prior Commit Point: ROLL, ROLB, and ROLS Calls

When a program determines that some of its processing is invalid, you can use the following calls to remove the effects of its incorrect processing: Roll Back calls ROLL, ROLS using a database PCB, ROLS with no I/O area or token, and ROLB. When you issue one of these calls, IMS does the following:

- Backs out the database updates that the program has made since the program's most recent commit point.
- Cancels the non-express output messages that the program has created since the program's most recent commit point.

The main difference among these calls is that ROLB returns control to the application program after backing out updates and canceling output messages, ROLS does not return control to the application program, and ROLL terminates the program with a user abend code of 0778. ROLB can return to the program the first message segment since the most recent commit point, but ROLL and ROLS cannot.

The ROLL and ROLB calls, and the ROLS call without a token specified, are valid when the PSB contains PCBs for GSAM data sets. However, segments inserted in the GSAM data sets since the last commit point are not backed out by these calls. An extended checkpoint-restart can be used to reposition the GSAM data sets when restarting.

You can use a ROLS call either to back out to the prior commit point or to back out to an intermediate backout point established by a prior SETS call. This topic refers only to the form of ROLS that backs out to the prior commit point. For information about the other form of ROLS, see "Backing out to an Intermediate Backout Point: SETS/SETU and ROLS" on page 149.

Table 15 summarizes the similarities and the differences among the ROLL, ROLS and ROLB calls.

Table 15. Comparison of ROLB, ROLL, and ROLS

Actions Taken:	ROLB	ROLL	ROLS
Back out database updates since the last commit point.	X	X	X
Cancel output messages created since the last commit point.	X[1]	X[1]	X[1]
Delete the message in process from the queue. Previous messages (if any) processed since the last commit point are returned to the queue to be reprocessed.		X	
Return the first segment of the first input message since the most recent commit point.	X[2]		
3303 abnormal termination and returns the processed input messages to the message queue.			X[3]

Table 15. Comparison of ROLB, ROLL, and ROLS (continued)

Actions Taken:	ROLB	ROLL	ROLS
778 abnormal termination, no dump.		X	
No abend; program continues processing.	X		

Notes:

1. ROLB, ROLL, or ROLS cancel output messages sent with an express PCB unless the program issued a PURG.

   Example: If the program issues the call sequence below, MSG1 would be sent to its destination because the PURG tells IMS that MSG1 is complete and the I/O area now contains the first segment of the next message (which in this example is MSG2). MSG2, however, would be canceled:

   ```
   ISRT    EXPRESS PCB, MSG1
   PURG    EXPRESS PCB, MSG2
   ROLB    I/O PCB
   ```

   Because IMS has the complete message (MSG1) and because an express PCB is being used, the message can be sent before a commit point.
2. Returned only if you supply the address of an I/O area as one of the call parameters.
3. The transaction is suspended and requeued for subsequent processing.

Using ROLL

A ROLL call backs out the database updates and cancels any non-express output messages the program has created since the last commit point. It also deletes the current input message. Any other input messages processed since the last commit point are returned to the queue to be reprocessed. IMS then terminates the program with a user abend code 0778. This type of abnormal termination terminates the program without a storage dump.

When you issue a ROLL call, the only parameter you supply is the call function, ROLL.

You can use the ROLL call in a batch program. If your system log is on direct access storage, and if dynamic backout has been specified through the use of the BKO execution parameter, database changes since the last commit point will be backed out. Otherwise they will not be backed out. One reason for issuing ROLL in a batch program is for compatibility.

After backout is complete, the original transaction is discarded if it is discardable, and it is not re-executed. IMS issues the APPC/MVS verb ATBCMTP TYPE(ABEND) specifying the TPI to notify remote transaction programs. Issuing the APPC/MVS verb causes all active conversations (including any spawned by the application program) to be DEALLOCATED TYP(ABEND_SVC).

Using ROLB

The advantage of using ROLB is that IMS returns control to the program after executing ROLB, so the program can continue processing. The parameters for ROL are:

- The call function ROLB
- The name of the I/O PCB or AIB

The total effect of the ROLB call depends on the type of IMS application that issued it.

- For current IMS application programs:

  After IMS backout is complete, the original transaction is represented to the IMS application program. Any resources that cannot be rolled back by IMS are ignored. For example, output sent to an express alternate PCB and a PURG call is issued before the ROLB.
- For modified IMS application programs:

  The same consideration for the current IMS application programs applies. It is the responsibility of the application program to notify any spawned conversations that a ROLB was issued.
- For CPI-C driven IMS application programs:

  Only IMS resources are affected. All database changes are backed out. Any messages inserted to non-express alternate PCBs are discarded. Also, any messages inserted to express PCBs that have not had a PURGE call are discarded. It is the responsibility of the application program to notify the originating remote program and any spawned conversations that a ROLB call was issued.

In MPPs and Transaction-Oriented BMPs

If the program supplies the address of an I/O area as one of the ROLB parameters, the ROLB call acts as a message retrieval call and returns the first segment of the first input message since the most recent commit point. This is true only if the program has issued a GU call to the message queue since the last commit point; it if has not, it was not processing a message when it issued the ROLB call.

If the program issues a GN to the message queue after issuing the ROLB, IMS returns the next segment of the message that was being processed when ROLB was issued. If there are no more segments for that message, IMS returns a QD status code.

If the program issues a GU to the message queue after the ROLB call, IMS returns the first segment of the next message to the application program. If there are no more messages on the message queue for the program to process, IMS returns a QC status code to the program.

If you include the I/O area parameter, but you have not issued a successful GU call to the message queue since the last commit point, IMS returns a QE status code to your program.

If you do not include the address of an I/O area in the ROLB call, IMS does the same things for you. If the program has issued a successful GU in the commit travel, and then issues a GN, IMS returns a QD status code. If the program issues a GU after the ROLB, IMS returns the first segment of the next message, or a QC status code if there are no more messages for the program.

If you have not issued a successful GU since the last commit point, and you do not include an I/O area parameter on the ROLB call, IMS backs out the database updates and cancels the output messages created since the last commit point.

In Batch Programs

If your system log is on direct access storage, and if dynamic backout has been specified through the use of the BKO execution parameter, you can use the ROLB call in a batch program. The ROLB call does not process messages as it does for MPPs; it backs out the database updates since the last commit point and returns control to your program. You cannot specify the address of an I/O area as one of the parameters on the call; if you do, an AD status code is returned to your

program. You must, however, have an I/O PCB for your program. Specify CMPAT=YES on the CMPAT keyword in the PSBGEN statement for your program's PSB.

Related Reading: For more information on using the CMPAT keyword, see *IMS/ESA Utilities Reference: System*. For information on coding the ROLB call, see "ROLB Call" on page 112.

Using ROLS

The two ways that you can use the ROLS call to back out to the prior commit point and return the processed input messages to IMS for later reprocessing are:

- Have your program issue the ROLS call using the I/O PCB but without an I/O area or token in the call. The parameters for this form of the ROLS call are:
  - The call function ROLS
  - The name of the I/O PCB or AIB
- Have your program issue the ROLS call using a database PCB that has received one of the data-unavailable status codes. This has the same result as if unavailable data were encountered, and the INIT call was not issued. ROLS must be the next call for that PCB. Intervening calls using other PCBs are permitted.

On a ROLS with a token, message queue repositioning can occur for all non-express messages including all messages processed by IMS. This processing using APPC/MVS calls and includes the initial message segments. The original input transaction can be represented to the IMS application program. Input and output positioning is determined by the SETS call. This positioning applies to current and modified IMS application programs but does not apply to CPI-C driven IMS programs. The IMS application program must notify all remote transaction programs of the ROLS.

On a ROLS without a token, IMS issues the APPC/MVS verb, ATBCMTP TYPE(ABEND), specifying the TPI. Issuing this verb causes all conversations associated with the application program to be DEALLOCATED TYPE(ABEND_SVC). If the original transaction was entered from an LU 6.2 device and IMS received the message from APPC/MVS, a discardable transaction is discarded rather than being placed on the suspend queue like a non-discardable transaction.

Related Reading: For more information on LU 6.2, see *IMS/ESA Administration Guide: Transaction Manager*.

The parameters for this form of the ROLS call are:

- The call function, ROLS
- The name of the DB PCB that received the BA or BB status code

In both of the above uses, the ROLS call causes a 3303 abnormal termination and does not return control to the application program. IMS keeps the input message for future processing.

Backing out to an Intermediate Backout Point: SETS/SETU and ROLS

You can use a ROLS call either to back out to an intermediate backout point established by a prior SETS or SETU call or to back out to the prior commit point. This topic refers only to the form of ROLS that backs out to the intermediate backout point. For information about the other form of ROLS, see “Backing out to a Prior Commit Point: ROLL, ROLB, and ROLS Calls” on page 145.

The ROLS call that backs out to an intermediate point backs out only DL/I changes. This version of the ROLS call does not affect CICS changes using CICS file control or CICS transient data.

The SETS and ROLS calls set intermediate backout points within the call processing of the application program and then backout database changes to any of these points. Up to nine intermediate backout points can be set. The SETS call specifies a token for each point. IMS then associates this token with the current processing point. A subsequent ROLS call, using the same token, backs out all database changes and discards all non-express messages that were performed following the SETS call with the same token. Figure 24 shows how the SETS and ROLS calls work together.

In addition, to assist the application program in reestablishing other variables following a ROLS call, user data can be included in the I/O area of the SETS call. This data is then returned when the ROLS call with the same token is issued.

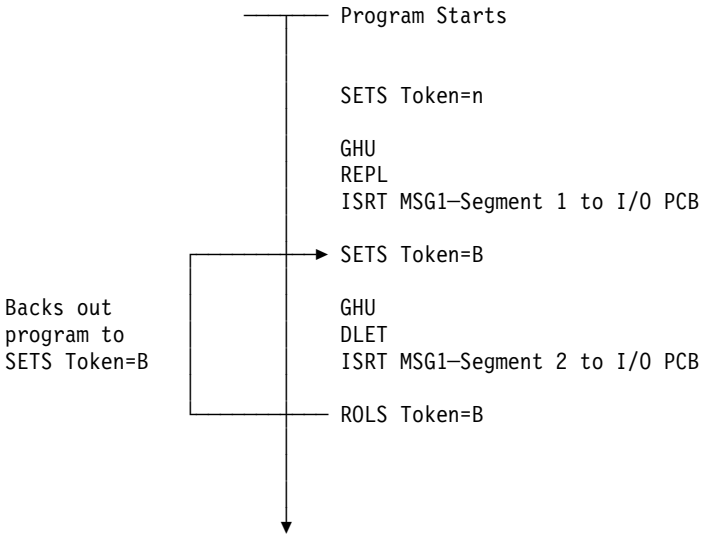


Figure 24. SETS and ROLS Calls Working Together

Using SETS/SETU

The SETS call sets up to nine intermediate backout points or cancels all existing backout points. By using the SETS call, you can back out pieces of work. If the necessary data to complete one piece of work is unavailable, you can complete a different piece of work and then return to the former piece.

To set an intermediate backout point, issue the call using the I/O PCB and include an I/O area and a token. The I/O area has the format LLZZuser-data, where LL is the length of the data in the I/O area including the length of the LLZZ portion. The ZZ field must contain binary zeros. The data in the I/O area is returned to the

application program on the related ROLS call. If you do not want to save some data to be returned on the ROLS call, you must set the LL that defines the length of the I/O area to 4.

For PLITDLI, you must define the LL field as a fullword rather than a halfword as it is for the other languages. The content of the LL field for PLITDLI is consistent with the I/O area for other calls using the LLZZ format; that is, the content is the total length of the area including the length of the 4-byte LL field minus 2.

A 4-byte token associated with the current processing point is also required. This token can be a new token for this program execution or match a token issued by a preceding SETS call. If the token is new, no preceding SETS calls are canceled. If the token matches the token of a preceding SETS call, the current SETS call assumes that position. In this case, all SETS calls that were issued subsequent to the SETS call with the matching token are canceled.

The parameters for this form of the SETS call are:

- The call function SETS
- The name of the I/O PCB or AIB
- The name of the I/O area containing the user data
- The name of an area containing the token

For the SETS call format, see “SETS/SETU Call” on page 116.

To cancel all previous backout points, the call is issued using the I/O PCB but does not include an I/O area or a token. When no I/O area is included in the call, all intermediate backout points set by prior SETS calls are canceled.

The parameters for this form of the SETS call are:

- The call function SETS
- The name of the I/O PCB or AIB

Because it is not possible to back out committed data, commit point processing causes all outstanding SETS to be canceled.

If PCBs for DEDB, MSDB, and GSAM organizations are in the PSB, or if the program accesses an attached subsystem, a partial backout is not possible. In that case, the SETS call is rejected with an SC status code. If the SETU call is used instead, it is not rejected because of unsupported PCBs, but returns an SC status code as a warning that the PSB contains unsupported PCBs and the function is not applicable to these unsupported PCBs.

For the status codes that are returned after the SETS call, see “Chapter 15. DL/I Status Codes” on page 443. For the explanations of those status codes and the response required, see “Status Code Explanations” on page 452.

Using ROLS

The ROLS call backs out database changes to a processing point set by a previous SETS or SETU call, or to the prior commit point and returns the processed input messages to the message queue.

To back out database changes and message activity that have occurred since a prior SETS call, you issue the ROLS call using the I/O PCB and specifying an I/O area and token in the call. If the token does not match a token set by a preceding SETS

call, an error status is returned. If the token does match the token of a preceding SETS call, the database updates made since this corresponding SETS call are backed out, and all non-express messages inserted since the corresponding SETS are discarded. The ROLS call returns blanks if the call is processed, and returns a status code if an error or warning occurs. If you are using SETU with ROLS and have an external subsystem, the ROLS call will not be rejected, but an RC status code will be returned as a warning. All SETS points that were issued as part of the processing that was backed out are then canceled, and the existing database position for all supported PCBs is reset. For the ROLS call format, see "ROLS Call" on page 114.

The parameters for this form of the ROLS call are:

- The call function ROLS
- The name of the I/O PCB or AIB
- The name of the I/O area to receive the user data
- The name of an area containing the 4-byte token

Related Reading: For the status codes that are returned after the ROLS call, see "Chapter 15. DL/I Status Codes" on page 443. For the explanations of those status codes and the response required, see "Status Code Explanations" on page 452.

Writing a Message-Driven Program

A message-driven program is similar to an MPP: it retrieves messages and processes them, and it can read and update MSDBs, DEDBs, and full-function databases.

Message-driven programs can send messages to the following destinations:

- The logical terminal that sent the input message, by issuing an ISRT call referencing the I/O PCB
- A different component of the physical terminal that sent the input message, by issuing an ISRT call referencing an alternate response PCB
- A different physical terminal from the one that sent the input message, by issuing an ISRT call referencing an alternate PCB

The message processing functions available to a message-driven program have some restrictions. These restrictions apply only to messages received or sent by the I/O PCB. The input message for a message-driven program must be a single segment message. Therefore, GU is the only call you can use to obtain the input message. The response message sent by the I/O PCB also must be a single segment message.

The transactions are in the response mode. This means that you must respond before the next message can be sent. You cannot use SPAs because a message-driven program cannot be a conversational program.

Not all of the system service calls are available. The following system service calls are valid in a message-driven region. However, other conditions might restrict their function in this environment:

CHKP (basic)

DEQ

INIT

LOG

SETS

ROLB
ROLS

The options or calls issued using alternate terminal PCBs have no constraints.

Coding DC Calls and Data Areas

The way you code DC calls and data areas depends on the application programming language you use.

Your Input

In addition to the information you need about the database processing that your program does, you need to know about message processing. Before you start to code, be sure you are not missing any of this information. Also, be aware of the standards at your installation that affect your program.

Information you need about your program's design:

- The names of the logical terminals that your program will communicate with
- The transaction codes, if any, for the application program's MPP skeleton to which your program will send messages
- The DC call structure for your program
- The destination for each output message that you send
- The names of any alternate destinations to which your program sends messages

Information you need about input messages:

- The size and layout of the input messages your program will receive (if possible)
- The format in which your program will receive the input messages
- The editing routine your program uses
- The range of valid data in input messages
- The type of data that input messages will contain
- The maximum and minimum length of input message segments
- The number of segments in a message

Information you need about output messages:

- The format in which IMS expects to receive output from your application program MPP skeleton
- The destination for the output messages
- The maximum and minimum length of output message segments

Skeleton MPP

For examples of skeleton MPPs, refer to:

Language	See
C	Figure 25 on page 154
COBOL	Figure 26 on page 156
Pascal	Figure 27 on page 158
PL/I	Figure 28 on page 160

These programs do not have all the processing logic that a typical MPP has. The purpose of providing these programs is to show you the basic MPP structure in COBOL, C language, Pascal, and PL/I. All the programs follow these steps:

1. The program retrieves an input message segment from a terminal by issuing a GU call to the I/O PCB. This retrieves the first segment of the message. Unless this message contains only one segment, your program issues GN calls to the I/O PCB to retrieve the remaining segments of the message. IMS places the input message segment in the I/O area that you specify in the call. In each of skeleton MPP examples, this is the MSG-SEG-IO-AREA.
2. The program retrieves a segment from the database by issuing a GU call to the DB PCB. This call specifies an SSA, SSA-NAME, to qualify the request. IMS places the database segment in the I/O area specified in the call. In this case, the I/O area is called DB-SEG-IO-AREA.
3. The program sends an output message to an alternate destination by issuing an ISRT call to the alternate PCB. Before issuing the ISRT call, the program must build the output message segment in an I/O area, and then the program specifies the I/O area in the ISRT call. The I/O area for this call is ALT-MSG-SEG-OUT.

The sample program is simplified for demonstration purposes; for example, the call to initiate sync point is not shown in the sample program. Include other IMS calls in a complete application program.

Coding Your Program in Assembler Language

The coding conventions of an assembler language MPP are the same as those for a DL/I assembler program. An assembler language MPP receives a PCB parameter list address in register 1 when it executes its entry statement. The first address in this list is a pointer to the I/O PCB; the addresses of any alternate PCBs that the program uses come after the I/O PCB address, and the addresses of the database PCBs that the program uses follow. Bit 0 of the last address parameter is set to 1.

Coding Your Program in C Language

The program shown in Figure 25 is a skeleton MPP written in C language. The numbers to the left of the program refer to the notes that follow the figure.

All storage areas that are referenced in the parameter list of your C language application program call to IMS can reside in the extended virtual storage area.

```
     #pragma runopts(env(IMS),plist(IMS))
1    #include <ims.h>
     #include <stdio.h>

     /* ------------------------------------------------------------ */
     /*                       ENTRY POINT                            */
     /* ------------------------------------------------------------ */

2    main() {

3      static const char func_GU[4] = "GU  ";
       static const char func_ISRT[4] = "ISRT";
     .
       char ssa_NAME[]...;
     .
4      char msg_seg_io_area[n];
       char db_seg_io_area[n];
       char alt_msg_seg_out[n];

5      int rc;
     .
6      #define io_pcb ((IO_PCB_TYPE *)(_pcblist[0])
       #define alt_pcb (_pcblist[1])
       #define db_pcb  (_pcblist[2])
     .
7      rc = ctdli(func_GU, io_pcb, msg_seg_io_area);
     .
8      rc = ctdli(func_GU, db_pcb, db_seg_io_area, ssa_name);
     .
9      rc = ctdli(func_ISRT, alt_pcb, alt_msg_seg_out);
     .
10   }
-----------------------------------------------------------------------
11   C language interface
```

Figure 25. C MPP Skeleton

Notes to Figure 25:

1. The env(IMS) establishes the correct operating environment and the plist(IMS) establishes the correct parameter list, when invoked under IMS. The ims.h header file contains declarations for PCB layouts, __pcblist, and the ctdli routine. The PCB layouts define masks for the DB PCBs that the program uses as structures. These definitions make it possible for the program to check fields in the DB PCBs.

   The stdio.h header file contains declarations for sprintf, which is useful for building SSAs.
2. After IMS has loaded the application program's PSB, IMS passes control to the application program through this entry point.
3. These are convenient definitions for the function codes and could be in one of your include files.
4. These could be structures, with no loss of efficiency.
5. The return code (status value) from DL/I calls can be returned and used separately.
6. The C language run-time sets up the __pcblist values. The order in which you refer to the PCBs must be the same order in which they have been defined in the PSB: first the I/O PCB, then any alternate PCBs that your program uses, and finally the database PCBs that your program uses.

7. The program issues a GU call to the I/O PCB to retrieve the first message segment. You can leave out the rc =, and check the status in some other way.
8. The program issues a GU call to the DB PCB to retrieve a database segment. The function codes for these two calls are identical; the way that IMS identifies them is by the PCB to which each call refers.
9. The program then sends an output message to an alternate destination by issuing an ISRT call to an alternate PCB.
10. When there are no more messages for the program to process, the program returns control to IMS by returning from main or by calling exit().
11. IMS provides a language interface module (DFSLI000) that gives a common interface to IMS. This module must be made available to the application program at link-edit time.

Coding Your Program in COBOL

The program in Figure 26 on page 156 is a skeleton MPP in COBOL that shows the main elements of an MPP. The numbers to the left of each part of the program refer to the notes that follow the program.

If you plan to preload your IBM COBOL for MVS & VM program, you must use the compiler option RENT. Alternatively, if you plan to preload your VS COBOL II program, you must use the compiler options RES and RENT.

If you want to use the IBM COBOL for MVS & VM compiler to compile a program that is to execute in AMODE(31) on MVS/ESA, you must use the compiler option RENT. Alternatively, if you want to use the VS COBOL II compiler to compile a program that is to execute in AMODE(31) on MVS/ESA, you must use the compiler options RES and RENT. All storage areas that are referenced in the parameter lists of your calls to IMS can optionally reside in the extended virtual storage area.

IBM COBOL for MVS & VM and VS COBOL II programs can coexist in the same application.

To compile the sample COBOL programs in this book, you must use the APOST option.

```
       ENVIRONMENT DIVISION.
           .
           .
           .
       DATA DIVISION.
       WORKING-STORAGE SECTION.
1          77 GU-CALL PICTURE XXXX VALUE 'GU  '.
           77 ISRT-CALL PICTURE XXXX VALUE 'ISRT'.
           77 CT PICTURE S9(5) COMPUTATIONAL VALUE +4.
           01 SSA-NAME.
           .
2          01 MSG-SEG-IO-AREA.
           .
           01 DB-SEG-IO-AREA.
           .
           01 ALT-MSG-SEG-OUT.
           .
       LINKAGE SECTION.
3          01 IO-PCB.
           .
           01 ALT-PCB.
           .
           01 DB-PCB.
           .
-------------------------------------------------------------
4      PROCEDURE DIVISION USING IO-PCB, ALT-PCB, DB-PCB

           ENTRY 'DLITCBL'
           .
5          CALL 'CBLTDLI' USING GU-CALL, IO-PCB,
               MSG-SEG-IO-AREA.
           .
6          CALL 'CBLTDLI' USING GU-CALL, DB-PCB,
               DB-SEG-IO-AREA, SSA-NAME.
           .
7          CALL 'CBLTDLI' USING ISRT-CALL, ALT-PCB,
               ALT-MSG-SEG-OUT.
           .
8          GOBACK.
-------------------------------------------------------------
9          COBOL LANGUAGE INTERFACE
-------------------------------------------------------------
```

Figure 26. COBOL MPP Skeleton

Notes to Figure 26:

1. To define each of the call functions that your program uses, use a 77 or 01 level working-storage statement. Assign the value to the call function in a picture clause defined as four alphanumeric characters.
2. Use a 01 level working-storage statement for each I/O area that you will use for message segments.
3. In the linkage section of the program, use a 01 level entry for each PCB that your program uses. You can list the PCBs in the order that you list them in the entry statement below, but this is not a requirement.
4. On the procedure statement, list the PCBs that your program uses in the order they are defined in the program's PSB: first the I/O PCB, then any alternate PCBs, and finally the database PCBs that your program uses.
5. The program issues a `GU` call to the I/O PCB to retrieve the first segment of an input message.

6. The program issues a GU call to the DB PCB to retrieve the segment that would be described in the SSA-NAME area.
7. The program sends an output message segment to an alternate destination by using an alternate PCB.
8. When there are no more messages for your MPP to process, you return control to IMS by issuing the GOBACK statement.

   If you compile all of your COBOL programs in the task with VS COBOL II, you can use STOP RUN, EXIT PROGRAM, and GOBACK, with their normal COBOL-defined semantics.
9. If the COBOL compiler option NODYNAM is specified, you must link edit the language interface module, DFSLI000, with your compiled COBOL application program. If the COBOL compiler option DYNAM is specified, do not link edit DFSLI000 with your compiled COBOL program.

Coding Your Program in Pascal

The program shown in Figure 27 on page 158 is a skeleton MPP written in Pascal. The numbers to the left of the program refer to the notes that follow the figure.

All storage areas that are referenced in the parameter list of your Pascal application program's call to IMS can reside in the extended virtual storage area.

```
1   segment PASCIMS;
    type
2     CHAR4 = packed array [1..4] of CHAR;
      CHARn = packed array [1..n] of CHAR;

3     IOPCBTYPE = record
                    (* Field declarations *)
                  end;

      ALTPCBTYPE = record
                     (* Field declarations *)
                   end;

      DBPCBTYPE = record
                    (* Field declarations *)
                  end;

4   procedure PASCIMS (var SAVE: INTEGER;
                       var IOPCB:  IOPCBTYPE;
                       var ALTPCB: ALTPCBTYPE;
                       var DBPCB:  DBPCBTYPE); REENTRANT;

    procedure PASCIMS;

5   type
      SSATYPE   = record
                    (* Field declarations *)
                  end;

      MSG_SEG_IO_AREA_TYPE = record
                               (* Field declarations *)
                             end;

      DB_SEG_IO_AREA_TYPE = record
                              (* Field declarations *)
                            end;

      ALT_MSG_SEG_OUT_TYPE = record
                               (* Field declarations *)
                             end;

6   var
      MSG_SEG_IO_AREA : MSG_SEG_IO_AREA_TYPE;
      DB_SEG_IO_AREA : DB_SEG_IO_AREA_TYPE;
      ALT_MSG_SEG_OUT : ALT_MSG_SEG_OUT_TYPE;

7   const
      GU   = 'GU  ';
      ISRT = 'ISRT';
      SSANAME  = SSATYPE(...);
```

Figure 27. Pascal MPP Skeleton (Part 1 of 2)

```
 8    procedure PASTDLI; GENERIC;

      begin

 9      PASTDLI(const GU,
                 var   IOPCB,
                 var   MSG_SEG_IO_AREA);

10      PASTDLI(const GU,
                 var   DBPCB,
                 var   DB_SEG_IO_AREA,
                 const SSANAME);

11      PASTDLI(const ISRT,
                 var   ALTPCB,
                 var   ALT_MSG_SEG_OUT);
12    end;
```

```
13      Pascal language interface
```

Figure 27. Pascal MPP Skeleton (Part 2 of 2)

Notes to Figure 27:

1. Define the name of the Pascal compile unit.
2. Define the data types needed for the PCBs used in your program.
3. Define the PCB data types used in your program.
4. Declare the procedure heading for the REENTRANT procedure called by IMS. The first word in the parameter list should be an INTEGER, which is reserved for VS Pascal's use, and the rest of the parameters will be the addresses of the PCBs received from IMS.
5. Define the data types needed for the SSAs and I/O areas.
6. Declare the variables used for the SSAs and I/O areas.
7. Define the constants (function codes, SSAs, and so forth) used in the PASTDLI DL/I calls.
8. Declare the IMS interface routine with the GENERIC Directive. GENERIC identifies external routines that allow multiple parameter list formats. A GENERIC routine's parameters are "declared" only when the routine is called.
9. The program issues a `GU` call to the I/O PCB to retrieve the first segment of an input message. The declaration of the parameters in your program might differ from this example.
10. The program issues a `GU` call to the DB PCB to retrieve a database segment. The function codes for these two calls are identical; the way that IMS identifies them is by the PCB that each call refers to. The declaration of the parameters in your program might differ from this example.
11. The program sends an output message segment to an alternate destination by issuing an `ISRT` call to an alternate PCB. The declaration of the parameters in your program might differ from this example.
12. When there are no more messages for your MPP to process, you return control to IMS by exiting the PASCIMS procedure. You can also code a RETURN statement to leave at another point.

13. You must link-edit your program to the IMS language interface module, DFSLI000, after you have compiled your program.

Coding Your Program in PL/I

The program shown in Figure 28 is a skeleton MPP written in PL/I. The numbers to the left of the program refer to the notes following the figure.

All storage areas that are referenced in the parameter list of your PL/I application program call to IMS can optionally reside in the extended virtual storage area.

If you plan to execute PL/I programs in 31-bit addressing mode, see *OS PL/I Version 2 Programming Guide*.

```
     /* ---------------------------------------------------- */
     /*                     ENTRY POINT                      */
     /* ---------------------------------------------------- */
1    UPDMAST:  PROCEDURE (IO_PTR, ALT_PTR, DB_PTR)
     OPTIONS (MAIN);

2    DCL   FUNC_GU     CHAR(4)   INIT('GU  ');
     DCL   FUNC_ISRT   CHAR(4)   INIT('ISRT');
     .
     DCL   SSA_NAME...;
     .
3    DCL   MSG_SEG_IO_AREA    CHAR(n);
     DCL   DB_SEG_IO_AREA     CHAR(n);
     DCL   ALT_MSG_SEG_OUT    CHAR(n);
     .
4    DCL   1 IO_PCB BASED (IO_PTR),...;
     DCL   1 ALT_PCB BASED (ALT_PTR),...;
     DCL   1 DB_PCB BASED (DB_PTR),...;
     .
5    DCL   THREE FIXED BINARY(31) INIT(3);
     DCL   FOUR FIXED BINARY(31)  INIT(4);
     DCL   PLITDLI ENTRY EXTERNAL;
     .
6    CALL PLITDLI (THREE, FUNC_GU, IO_PTR, MSG_SEG_IO_AREA);
     .
7    CALL PLITDLI (FOUR, FUNC_GU, DB_PTR, DB_SEG_IO_AREA,
          SSA_NAME);
     .
8    CALL PLITDLI (THREE, FUNC_ISRT, ALT_PTR, ALT_MSG_SEG_OUT);
     .
9    END UPDMAST;
-----------------------------------------------------------------
10   PL/I LANGUAGE INTERFACE
```

Figure 28. PL/I MPP Skeleton

Notes to Figure 28:

1. This is the standard entry point to a PL/I Optimizing Compiler MPP. This statement includes a pointer for each PCB that the MPP uses. You must refer to the PCBs in the same order as they are listed in the PSB: first the I/O PCB, then any alternate PCBs that your program uses, and finally the database PCBs that your program uses.
2. The program defines each call function that it uses in its data area. In PL/I you define the function codes as character strings and assign the appropriate values to them.

3. The most efficient way to define your I/O areas in PL/I is to define them as fixed-length character strings or through pointer variables.
4. To define your PCBs, use major structure declarations.
5. PL/I calls have a parameter that is not required in COBOL programs or assembler language programs. This is the parmcount, and it is always the first parameter. You define the values that your program will need for the parmcount in each of its calls. The parmcount gives the number of parameters that follow parmcount itself.
6. The program issues a `GU` call to the I/O PCB to retrieve the first message segment.
7. The program issues a `GU` call to the DB PCB to retrieve a database segment. The function codes for these two calls are identical; the way that IMS identifies them is by the PCB that each call refers to.
8. The program then sends an output message to an alternate destination by issuing an `ISRT` call to an alternate PCB.
9. When there are no more messages for the program to process, the program returns control to IMS by issuing the END statement or the RETURN statement.
10. You must link-edit your program to the IMS language interface module, DFSLI000, after you have compiled your program.

Part 2. Message Format Service

Chapter 6. Introduction to MFS 169
Advantages of Using MFS 169
Simplify Development and Maintenance 169
Improve Online Performance of a Terminal 170
MFS Control Blocks 171
MFS Examples 171
Looking at Payroll Records 172
Listing a Subset of Employees. 174
Relationship Between MFS Control Blocks and Screen Format. 175
Overview of MFS Components and Operation 177
MFS Language Utility 178
MFS Service Utility 179
MFS Device Characteristics Table Utility 179
MFS Message Editor 179
MFS Pool Manager. 179
MFSTEST Pool Manager. 180
Devices and Logical Units That Operate with MFS 180
Using Distributed Presentation Management (DPM) 182

Chapter 7. Message Formatting Functions 183
Input Message Formatting 183
How MFS Is Selected 183
274X, 3770, SLU 1, NTO, and SLU 4 183
3270 and SLU 2 184
Finance and SLU P Workstations 184
Intersystem Communication (ISC) Subsystems. 185
Formatting Messages from Terminals in Preset Destination Mode 185
Formatting of Messages Using Fast Path. 185
How MFS Formats Input Messages 186
Input Message Formatting Options 186
Examples 187
Cursor Position Input and FILL=NULL 191
Input Logical Page Selection 191
Input Message Field and Segment Edit Routines 192
Input Message Literal Fields 192
Input Message Field Attribute Data 193
IMS TM Password 193
Fill Characters for Input Message Fields 194
Input Modes (Devices Other Than 3270, SLU 2, or ISC Subsystems) 194
Input Field Tabs (Devices Other Than 3270 or SLU 2) 194
Optional Deletion of Null Characters for DPM-An 196
Examples of Optional Null Character Deletion for DPM-An 197
Multiple Physical Page Input Messages (3270 and SLU 2 Display Devices) 200
General Rules for Multiple DPAGE Input 200
3270 and SLU 2 Input Substitution Character 201
Input Format Control for ISC (DPM-Bn) Subsystems 201
Input Message Formatting 201
Input DPAGE Selection 201
Single Transmission Chain 201
Multiple Transmission Chains 202
Input Modes 202
Record Mode 202

© Copyright IBM Corp. 1974, 2000

Stream Mode . . . 203
Paging Requests . . . 203
Output Message Formatting . . . 203
How MFS Is Selected . . . 203
How MFS Formats Output Messages . . . 204
Output Message Formatting Options . . . 204
Logical Paging of Output Messages . . . 205
Operator Logical Paging of Output Messages . . . 207
Physical Paging of Output Messages . . . 207
Fill Characters for Output Device Fields . . . 208
System Control Area (SCA) and Default SCA (DSCA) . . . 209
Output Message Literal Fields . . . 210
Output Device Field Attributes . . . 210
Extended Field Attributes for Output Devices . . . 211
Extended Graphic Character Set (EGCS) . . . 212
Mixed DBCS/EBCDIC Fields . . . 214
Cursor Positioning . . . 220
Prompt Facility . . . 221
System Message Field (3270 or SLU 2 Display Devices) . . . 221
Printed Page Format Control . . . 221
Format Control for 3770, SLU 1, and SLU 4 Printers . . . 222
Output Format Control for 3270P Printers . . . 223
Output Format Control for SLU P DPM-An . . . 224
Output Format Control for ISC (DPM-Bn) Subsystems . . . 228
Format Control . . . 228
Function Management (FM) Headers . . . 228
Paged Output Messages . . . 228
Demand Paging . . . 228
Autopaged Output . . . 228
Output Modes . . . 229
Variable-Length Output Data Stream . . . 230
Output Field Tab Separator Character . . . 230
FILL=NULL Specification . . . 231
Trailing Blank Compression . . . 232
Specifying COMPR . . . 232
Saving Line Transmission Time . . . 232
Blank Compression on Variable-Length Output . . . 232
Data Structure Name . . . 235
Version Identification . . . 235
Your Control of MFS . . . 235
Operator Logical Paging . . . 236
Functions Provided . . . 236
Format Design Considerations . . . 236
Transaction Codes and Logical Page Requests . . . 236
Operator Control Tables . . . 237
3270 or SLU 2-Only Feature Definitions . . . 237
Paging Action at the Device . . . 238
Unprotected Screen Option . . . 242
The 3290 in Partitioned Format Mode . . . 243
Partition Initialization Options and Paging . . . 244
Clearing the Display . . . 245
The JUMP PARTITION Key . . . 245
Scrolling Operations . . . 245
The 3180 in Partitioned Format Mode . . . 245
Partition Option and Paging . . . 246
MFS Format Sets Supplied by IMS . . . 246

System Message Format 246
Multisegment System Message Format 246
Output Message Default Format 247
Block Error Message Format 247
/DISPLAY Command Format 247
Multisegment Format 247
MFS 3270 or SLU 2 Master Terminal Format 247
MFS Sign-On Device Formats 247
MFS Formatting for the 3270 or SLU 2 Master Terminal 247
MFS Device Characteristics Table 249
Version Identification Function for DPM Formats 250

Chapter 8. MFS Application Program Design 251
Relationships Between MFS Control Blocks 251
Device Considerations Relative to Control Block Linkages 256
3270 or SLU 2 Display Devices 256
3290 Information Panel in Partitioned Format Mode 257
274X, Finance, 3770, SLU 1, NTO, SLU 4, or SLU P 257
Finance or SLU P Workstations 258
ISC Subsystem (DPM-Bn) 258
Format Library Member Selection 258
3270 or SLU 2 Screen Formatting 261
3290 Screen Formatting 263
Screen Division 263
Terminal States and Modes 264
Partition Set Initialization, Paging, and Activation 264
3180 Screen Formatting 265
Performance Factors 265
All MFS-Supported Devices 265
3270 or SLU 2 Display Devices 266
3270 or SLU 2 Devices with Large Screens 267
SLU P and ISC Subsystems with DPM 267
Loading Programmed Symbol Buffers 268
Using an Application Program to Determine Whether Programmed Symbol Buffers Are Loaded 268
How to Load the Programmed Symbol Buffers 269
Solving Programmed Symbol Load Problems 269

Chapter 9. Application Programming Using MFS 273
Input Message Formats 273
Logical Pages 273
Device-Dependent Input Information (3270 or SLU 2) 273
Cursor Location 273
Selector Pen 274
Magnetic Stripe Reading Devices 275
Program Function Keys 275
Program Access Keys 275
Output Message Formats 275
Logical Pages 275
Segment Format 276
Example 277
Field Format (Options 1 and 2) 277
Field Format (Option 3) 278
Device-Dependent Output Information 278
System Control Area (SCA) 278
Cursor Location 280

Dynamic Attribute Modification 281
Dynamic Modification of Extended Field Attributes 283
Types 283
Values 283
Dynamic Modification of EGCS Data 289
Dynamic Modification of DBCS/EBCDIC Mixed Data 290
Specification of Message Output Descriptor Name 291
MFS Bypass for the 3270 or SLU 2 292
Specifying Input Forms for MFS Bypass 292
MFS Bypass for the SLU 2 (3290) with Partitioning 294
DIV Statement 294
DPAGE Statement 304

Chapter 10. MFS Language Utility 311
Utility Control Statements 311
Control Statement Syntax 311
Five Special Rules 312
Syntax Errors 313
Invalid Sequence of Statements 313
Summary of Control Statements 314
EXEC Statement Parameters 316
Message Definition Statements 317
MSG Statement 317
LPAGE Statement 319
PASSWORD Statement 320
SEG Statement 321
DO Statement 322
Printing Generated MFLD Statements 323
MFLD Statement 323
ENDDO Statement 329
MSGEND Statement 330
Format Definition Statements 330
FMT Statement 330
DEV Statement 330
DIV Statement 351
DPAGE Statement 360
PPAGE Statement 366
DO Statement 367
RCD Statement 369
DFLD Statement 370
ENDDO Statement 385
FMTEND Statement 385
Partition Set Definition Statements 386
PDB Statement 386
PD Statement 387
PDBEND Statement 389
Table Definition Statements 389
TABLE Statement 389
IF Statement 389
TABLEEND Statement 391
Compilation Statements 391
ALPHA Statement 391
COPY Statement 391
EQU Statement 392
Concatenated EQU Statements 393
RESCAN Statement 393

STACK Statement 394
UNSTACK Statement 394
TITLE Statement. 395
PRINT Statement 395
SPACE Statement 395
EJECT Statement 396
END Statement 396

Chapter 6. Introduction to MFS

The IMS message format service (MFS) is a facility of the IMS Transaction Manager environment that formats messages to and from terminal devices, so that IMS application programs do not deal with device-specific characteristics in input or output messages. In addition, MFS formats messages to and from user-written programs in remote controllers and subsystems, so that application programs do not deal with transmission-specific characteristics of the remote controller.

MFS uses control blocks you specify to indicate to IMS how input and output messages are arranged.

For input messages, MFS control blocks define how the message sent by the device to the application program is arranged in the program's I/O area.

For output messages, MFS control blocks define how the message sent by the application program to the device is arranged on the screen or at the printer. Data that appears on the screen but not in the program's I/O area, such as a literal, can also be defined.

In IMS Transaction Manager systems, data passing between the application program and terminals or remote programs can be edited by MFS or basic edit. Whether an application program uses MFS depends on the type of terminals or secondary logical units (SLUs) your network uses.

Restriction: MFS does not support message formatting for LU 6.2 devices.

In this Chapter:

- "Advantages of Using MFS"
- "MFS Control Blocks" on page 171
- "Overview of MFS Components and Operation" on page 177
- "Devices and Logical Units That Operate with MFS" on page 180

Advantages of Using MFS

The advantages of using MFS are as follows:

- MFS simplifies developing and maintaining terminal-oriented applications by performing common application functions and providing independence from specific devices or remote programs.
- MFS improves online performance by using control blocks for online processing.

Simplify Development and Maintenance

To simplify IMS application development and maintenance, MFS performs many common application program functions and gives application programs a high degree of independence from specific devices or remote programs.

With the device independence offered by MFS, one application program can process data to and from multiple device types while still using their different capabilities. Thus, MFS can minimize the number of required changes in application programs when new terminal types are added.

MFS makes it possible for an application program to communicate with different types of terminals without having to change the way it reads and builds messages. When the application receives a message from a terminal, how the message

© Copyright IBM Corp. 1974, 2000

appears in the program's I/O area is independent of what kind of terminal sent it; it depends on the MFS options specified for the program. If the next message the application receives is from a different type of terminal, you do not need to do anything to the application. MFS shields the application from the physical device that is sending the message in the same way that a DB program control block (PCB) shields a program from what the data in the database actually looks like and how it is stored.

Other common functions performed by MFS include left or right justification of data, padding, exits for validity checking, time and date stamping, page and message numbering, and data sequencing and segmenting. When MFS assumes these functions, the application program handles only the actual processing of the message data.

Figure 29 shows how MFS can make an application program device-independent by formatting input data from the device or remote program for presentation to IMS, and formatting the application program data for presentation to the output device or remote program.

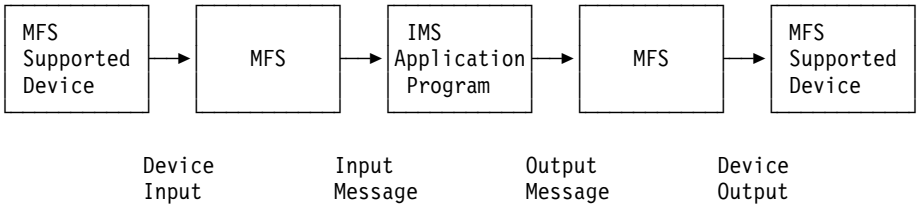


Figure 29. Message Formatting Using MFS

Improve Online Performance of a Terminal

MFS also improves online performance of a terminal-oriented IMS by using control blocks designed for online processing. The MFS control blocks are compiled offline, when the IMS Transaction Manager system is not being executed, from source language definitions. MFS can check their validity and make many decisions offline to reduce online processing. In addition, during online processing, MFS uses look-aside buffering of the MFS control blocks to reduce CPU and channel costs of input/output activity.

Because MFS control blocks are reentrant and can be used for multiple applications, online storage requirements are reduced. Optional real storage indexing and anticipatory fetching of the control blocks can also reduce response time. Further performance improvements can be gained when IMS is generated for MVS/ESA, since multiple I/O operations can execute concurrently to load the format blocks from the MFS format library.

In addition, MFS uses MVS paging services; this helps to reduce page faults by the IMS control region task.

MFS can reduce use of communication lines by compressing data and transmitting only required data. This reduces line load and improves both response time and device performance.

MFS Control Blocks

There are four types of MFS control blocks that you specify to format input and output for the application program and the terminal or remote program:

- *Message Output Descriptors (MODs)* define the layout of messages MFS receives from the application program.
- *Device Output Formats (DOFs)* describe how MFS formats messages for each of the devices the program communicates with.
- *Device Input Formats (DIFs)* describe the formats of messages MFS receives from each of the devices the program communicates with.
- *Message Input Descriptors (MIDs)* describe how MFS further formats messages so that the application program can process them.

Throughout this book, the term “message descriptors” refers to both MIDs and MODs. The term “device formats” refers to both DIFs and DOFs.

Each MOD, DOF, DIF and MID deals with a specific message. There must be a MOD and DOF for each unique message a program sends, and a DIF and MID for each unique message a program receives.

MFS Examples

One way to understand the relationship between the MFS control blocks is to look at a message from the time a user enters it at the terminal to the time the application program processes the message and sends a reply back to the terminal. Though MFS can be used with both display terminals and printer devices, for clarity in this example, a display terminal is being used.

Figure 30 shows the relationships between the MFS control blocks.

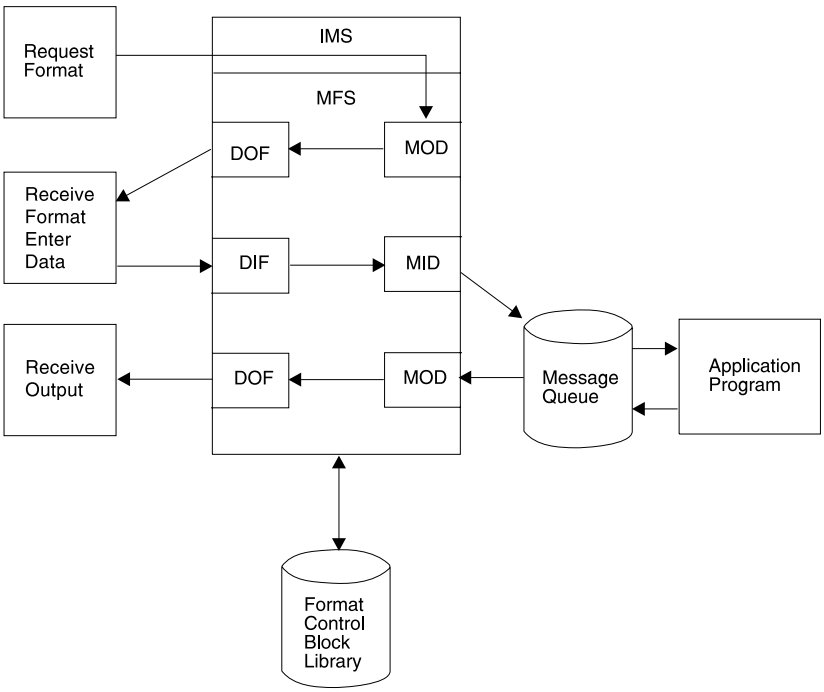


Figure 30. MFS Control Block Relationships

Looking at Payroll Records

Suppose your installation has a message processing program used to view employee payroll records. From a display terminal, issue the IMS format command (/FORMAT), and the MOD name. This formats the screen in the way defined by the MOD written by the MFS programmer. When you enter the MOD name, the screen contains only literals and no output data from the application program. At this stage, no application program is involved. (For more information about /FORMAT, see *IMS/ESA Operator's Reference*.)

In this example, suppose the name of the MOD that formats the screen for this application is PAYDAY. Enter this command:

```
/FORMAT PAYDAY
```

IMS locates the MFS MOD control block with the name PAYDAY and arranges the screen in the format defined by the DOF. Figure 34 on page 177 shows an example of the MFS control statements that define a MID, MOD, DIF, and DOF.

Figure 31 shows how this screen looks.

```
                        *EMPLOYEE PAYROLL*
                        ******************


          FIRST NAME:                          LAST NAME:
          EMPLOYEE NO:



          INPUT:
```

Figure 31. PAYDAY Screen, Formatted by DOF

The DOF defines a terminal format that asks you to give the employee's name and employee number. PAYUP is the transaction code associated with the application that processes this information. When you enter the MOD name, the transaction code is included in the first screen format displayed. This means that you do not need to know the name of the program that processes the data; you only need the name of the MOD that formats the screen.

After the screen format is displayed, you can enter the information. There are four stages to sending a message to the program and receiving the reply:

1. Enter the information at the terminal. For this example, enter the prompted information.

   Figure 32 shows how this screen looks after information is entered.

```
                          *EMPLOYEE PAYROLL*
                          ******************


          FIRST NAME: Joe                        LAST NAME: Blutzen
          EMPLOYEE NO: 60249


          INPUT:
```

Figure 32. PAYDAY Screen, with Filled Input Fields

2. When IMS receives this data, MFS uses the DIF and the MID control blocks to translate the data from the way it was entered on the terminal screen to the way that the application program is expecting to receive it. The DIF control block tells MFS the format of the data to come in from the terminal. The MID control block tells MFS how the application program expects to receive the data. When the application program issues a message call, IMS places the “translated” message in the program’s I/O area.

   When the application receives the message in its I/O area, the message looks like this:

   ```
   PAYUP JOE BLUTZEN 60249
   ```

   “PAYUP” is the transaction code. The name of the logical terminal does not appear in the message itself; IMS places it in the first field of the I/O PCB.

3. The application program processes the message, including any required database access, and builds the output message in the application program’s I/O area. After retrieving the information from the database, the program builds the output message segment for the employee, with social security and rate of pay information. The application program’s I/O area contains:

   ```
   LLZZJOE BLUTZEN 60249532596381150.00
   ```

   The LL is a 2-byte field in MFS messages that indicates the length of the field. How the LL field is defined depends on what programming language used to write the application program. For the AIBTDLI, ASMTDLI, CEETDLI, or PASTDLI interfaces, the LL field must be defined as a binary half word. For the PLITDLI interface, the LL field must be defined as a binary fullword. The value provided in the PLITDLI interface must represent the actual segment length minus 2 bytes.

   The ZZ is a 2-byte length field in MFS messages that contains the MFS formatting option that is being used to format the messages to and from the application program. MFS options are discussed in further detail in “Input Message Formatting Options” on page 186.

4. When the application program sends the message back to the terminal, MFS translates the message again, this time from the application program format to the format in which the terminal expects the data.

   The MOD tells MFS the format that the message will be in when it comes from the application program’s I/O area. The DOF tells MFS how the message is supposed to look on the terminal screen. MFS translates the message and IMS displays the translated message on the terminal screen.

Figure 33 shows how the screen looks.

```
                          *EMPLOYEE PAYROLL*
                          ******************


          FIRST NAME: Joe                        LAST NAME: Blutzen
          EMPLOYEE NO: 60249
          SOC SEC NO: 532-59-6381
          RATE OF PAY: $150.00


          INPUT:
```

Figure 33. PAYDAY Screen, Output Formatted by DOF and Displayed

Listing a Subset of Employees

Suppose you have an MPP that answers this request:

> List the employees who have the skill “ENGINEER” with a skill level of “3.” List only those employees who have been with the firm for at least 4 years.

To enter the request from a display terminal, issue the format command (/FORMAT) and the MOD name. This formats the screen in the way defined by the MOD you supply. When you enter the MOD name, the screen contains only literals and no output data from an application program. At this stage, an MPP is not involved. Suppose the name of the MOD that formats the screen for this request is LE, for “locate employee.” Enter this:

```
/FORMAT LE
```

IMS locates the MFS MOD control block with the name LE and arranges your screen in the format defined by the DOF. Your screen then looks like this:

```
SKILL
LEVEL
YEARS
      LOCEMP
```

The DOF defines a terminal format that asks you to qualify your request for an employee by giving the skill, level, and number of years of service of the employee you want. LOCEMP is the transaction code that is associated with the MPP that can process this request. When you enter the MOD name, the transaction code is included in the first screen format that is displayed for you. This means that you do not need the name of the program that processes your request; you only need the name of the MOD that formats the screen.

After the screen format is displayed, you can enter your request. There are four stages in sending a message to the program and receiving the reply.

1. Enter the information at the terminal. In this example, enter the values of the qualifications that IMS has given you on the screen: the skill is “eng” (engineer), the skill level is “3,” and the number of years with the firm is “4”.

   After you enter your request, your screen contains this data:

   ```
   SKILL ENG
   LEVEL 3
   YEARS 4
         LOCEMP
   ```

2. When IMS receives this data, MFS uses the DIF and the MID control blocks to translate the data from the way you entered it on the terminal screen to the way that the application program is expecting to receive it. The DIF control block tells MFS how the data is going to come in from the terminal. The MID control block tells MFS how the application program is expecting to receive the data. When the application program issues a GU call to the I/O PCB, IMS places the "translated" message in the program's I/O area.

   When the MPP receives the message in its I/O area, the message looks like this:

   ```
   LOCEMP    ENG0304
   ```

   "LOCEMP" is the transaction code. The name of the logical terminal does not appear in the message itself; IMS places it in the first field of the I/O PCB.

3. The MPP processes the message, including any required database access, and builds the output message in the MPP's I/O area.

   Suppose more than one employee meets these qualifications. The MPP can use one message segment for each employee. After retrieving the information from the database, the program builds the output message segment for the first employee. The program's I/O area contains:

   ```
   LLZZJONES,CE   3294
   ```

   When the program sends the second segment, the I/O area contains:

   ```
   LLZZBAKER,KT   4105
   ```

4. When the application program sends the message back to the terminal, MFS translates the message again, this time from the application program format to the format in which the terminal expects the data.

   The MOD tells MFS the format that the message will be in when it comes from the application program's I/O area. The DOF tells MFS how the message is supposed to look on the terminal screen. MFS translates the message and IMS displays the translated message on the terminal screen. The screen then contains the following data:

   ```
   SKILL    ENG
   NAME     NO
   JONES,CE 3294
   BAKER,KT 4105
   ```

Relationship Between MFS Control Blocks and Screen Format

This section discusses the relationship between MFS source language definitions and formats you see at the device. The sample code is designed for a 3270 display.

The standard way for an end-user or operator to receive an initial format is to request it with a /FORMAT command, specifying the name of a MOD. In Figure 34 on page 177, the label on the MOD is PAYDAY. This MOD contains the parameter SOR=PAYF, which points to a device output format, or DOF, with the same label.

The initial DOF also becomes the format for device input. Therefore, if you specify DIV TYPE=INOUT in the DOF, a device input format (DIF) is also generated. In the sample code, PAYF is both a DOF and a DIF, since it also describes the format of the next input. The final output message can be displayed with a format that is specified for output only and no DIF is generated.

Both the MOD and the MID point to the same DOF, thus establishing the relationship between device-related and message-related control blocks.

For output, MFS moves fields defined in a MOD to fields on the screen defined by a DOF. When a field definition is coded (MFLD) in a MOD, it is given a label. The same label is used in the coding of the device field (DFLD) in the DOF, defining where the field appears on the screen.

MFS moves data fields from output messages to screen fields; this is referred to as *mapping*. For input, MFS moves modified screen fields to data fields in the input message for the program by mapping identically labeled fields in the DIF and MID. For more detailed information on specifying these control blocks, see “Chapter 10. MFS Language Utility” on page 311.

The MFS control blocks are generated from the source statements like those in Figure 34 during execution of the MFS language utility program. The control blocks are stored in the various MFS libraries.

DOF/DIF

```
PAYF     FMT
         DEV       TYPE=(3270,2),FEAT=IGNORE,DSCA=X'00A0'
         DIV       TYPE=INOUT
         DPAGE     CURSOR=((5,15))
         DFLD      '**********************',POS=(1,21)
         DFLD      '*  EMPLOYEE PAYROLL  *',POS=(2,21)
         DFLD      '**********************',POS=(3,21)
         DFLD      'FIRST NAME:',POS=(5,2)
FNAME    DFLD      POS=(5,15),LTH=16
         DFLD      'LAST NAME:',POS=(5,36)
LNAME    DFLD      POS=(5,48),LTH=16
         DFLD      'EMPLOYEE NO:',POS=(7,2)
EMPNO    DFLD      POS=(7,16),LTH=6
         DFLD      'SOC SEC NO:',POS=(9,2)
SSN      DFLD      POS=(9,15),LTH=11
         DFLD      'RATE OF PAY: $',POS=(11,2)
RATE     DFLD      POS=(11,17),LTH=9
         DFLD      'INPUT:',POS=(16,2)
INPUT    DFLD      POS=(16,10),LTH=30
         FMTEND
```

MID

```
PAYIN    MSG       TYPE:INPUT,SOR=(PAYF,IGNORE)
         SEG
         MFLD      'PAYUP '     SUPPLIES TRANCODE
         MFLD      LNAME,LTH=16
         MFLD      FNAME,LTH=16
         MFLD      EMPNO,LTH=6
         MFLD      SSN,LTH=11
         MFLD      RATE,LTH=9
         MFLD      INPUT,LTH=30,JUST=R,FILL=C'0'
         MSGEND
```

MOD

```
PAYDAY   MSG       TYPE:OUTPUT,SOR=(PAYF,IGNORE)
         SEG
         MFLD      LNAME,LTH=16
         MFLD      FNAME,LTH=16
         MFLD      EMPNO,LTH=6
         MFLD      SSN,LTH=11
         MFLD      RATE,LTH=9
         MFLD      INPUT,LTH=30,JUST=R,FILL=C'0'
         MSGEND
```

Figure 34. Sample MFS Control Block Coding

Overview of MFS Components and Operation

MFS has the following components:

- The **MFS language utility**, which generates control blocks from user-written control statements and places them in a library called IMS.FORMAT.
- The **MFS service utility**, which is used for maintaining the control blocks in IMS.FORMAT.
- The **MFS device characteristics table utility**, which is used to add new screen sizes in the device characteristics table (DCT) and generate new MFS default formats for the screen size without system generation.
- The **MFS message editor**, which formats messages according to the control block specifications generated by the language utility.

- The **MFS pool manager**, which keeps the MFS control blocks required by the message editor in the real storage MFS buffer pool.
- The **MFSTEST pool manager**, which replaces the MFS pool manager when the language utility is being used in test mode.

The **IMS online change utility** also plays an important part in updating the MFS libraries, even though it is not an MFS utility. The online change utility allows the control block libraries to be modified while the IMS control region is executing.

Related Reading: For a more complete description of online change, see *IMS/ESA Utilities Reference: System*.

MFS Language Utility

The MFS language utility processes user-written control statements. The primary function of this utility is to create MFS control blocks used in online execution. Definition control statements define the MFS control blocks.

Additional functions of the MFS language utility include:

- SYSPRINT listing control
- SYSIN/SYSLIB record stacking and unstacking
- Repetitive generation of message and device fields
- Equate processing
- Alphabetic character generation
- Copying SYSLIB members into the utility input stream
- Printing statistics of counters maintained by the utility

A number of parameters on the JCL EXEC statement used during compilation can be varied to control printed output, compress the partitioned data set libraries IMS.FORMAT and IMS.REFERAL, and prevent definitions with a specified level of error from being written in IMS.REFERAL.

The language utility can operate in three modes: standard, test, and batch. All produce the same control blocks. They differ in their ability to operate concurrently with the IMS online control region and in their use of the MFS libraries.

In standard mode, the MFSUTL job control procedure can execute concurrently with the IMS control region. It stores control blocks in the IMS.FORMAT library.

In test mode, the MFSTEST procedure can execute concurrently with the IMS online control region. It stores control blocks in the IMS.TFORMAT library.

In batch mode, the MFSBTCH1 procedure places the control blocks in a temporary library, IMS.MFSBATCH. The MFSBTCH2 procedure transfers the control blocks to IMS.FORMAT. The MFSBTCH1 procedure can be executed many times, and control blocks can be accumulated in IMS.MFSBATCH before they are transferred to the staging library.

The language utility checks the syntax of the source language definitions and converts them to a form intermediate between the source language and the final online control block, called an *intermediate text block* (ITB). In standard mode, it writes these ITBs in the historical reference library, IMS.REFERAL. Although most ITBs are immediately converted to online control blocks and written in the staging library, IMS.FORMAT, the ITBs and the relationships between them are still retained

in IMS.REFERAL. When the language utility begins processing, a table of all ITBs currently in IMS.REFERAL and their interrelationships is created. Each new definition is then checked against the table. Newly entered definitions that have valid syntax, that belong to a complete format set (complete with DIF or DOF and associated MID or MOD), and have consistent references to other ITBs in the set, are converted to online control blocks and are immediately written in the IMS.FORMAT library (in standard mode) or the IMS.TFORMAT library (in test mode).

Two IMS commands are available to request format sets while using the language utility. To request use of a format set, a terminal operator enters the /FORMAT command. To test the format sets in IMS.TFORMAT, the terminal operator enters the /TEST MFS command. Then the /FORMAT command can be used to call test format sets from IMS.TFORMAT (and format sets from IMS.FORMAT, if necessary) into the communication line buffer pool for test MFS operation. After successful testing, the format sets can be written in the staging library, IMS.FORMAT.

The use of the MFS commands /FORMAT and /TEST is explained in the discussion of those commands in the *IMS/ESA Operator's Reference*.

MFS Service Utility

The MFS service utility performs optional indexing, reporting, and maintenance functions. The INDEX function puts index entries for specified IMS.FORMAT control blocks in a special real storage directory, to allow faster access to the control blocks. Other functions are used to delete or obtain reports on the contents of the libraries and directories.

MFS Device Characteristics Table Utility

The MFS device characteristics table (MFS DCT) utility is used to add new screen sizes to the DCT and generate new MFS default formats for those screen sizes without performing an IMS system generation. The definition of the new screen sizes to the utility is made on the new ETO device descriptor. New screen size definitions are added to screen sizes that were previously defined.

Related Reading: For an example of an MFS device descriptor used by the DCT, or for more information on ETO, see *IMS/ESA Administration Guide: Transaction Manager*. For more information on the MFS DCT utility, see *IMS/ESA Utilities Reference: Transaction Manager*.

MFS Message Editor

The MFS message editor formats messages according to the control block specifications generated by the language utility from control statement definitions you enter. The editor can also give control to optional user-written or IMS-provided field and segment editing routines (such as validity checks). The IMS-provided editing routines are shown in *IMS/ESA Customization Guide*.

MFS Pool Manager

MFS tries to minimize I/O to the format library by keeping referenced blocks in storage. This storage is managed by the MFS pool manager. The INDEX function of the MFS service utility allows you to customize this function, by constructing a list of the directory addresses for specified format blocks, eliminating the need for IMS to read the data set directory before fetching a block.

For more information, refer to the *IMS/ESA Administration Guide: Transaction Manager*.

MFSTEST Pool Manager

If the optional MFSTEST facility is used, MFS control blocks are managed by the MFSTEST pool manager. The communication line buffer pool space allowed for MFS testing is specified at system definition, but the space can be changed when the IMS control region is initialized. This space value is the maximum amount used for MFSTEST blocks at any one time—it is not a reserved portion of the pool.

Devices and Logical Units That Operate with MFS

In addition to 3270 devices, MFS operates with the 3600 and 4700 Finance Communication System (FIN), the 3770 Data Communication System, the 3790 Communication System, and with Secondary Logical Unit (SLU) types 1, 2, 4, 6, and P. Network Terminal Operations (NTO) devices are supported as secondary logical unit type 1 consoles.

Table 16 shows which devices or logical units can be defined for MFS operation in the IMS system by their number (3270, for example), and which can be defined by the type of logical unit to which they are assigned (SLU 1, for example). For a list of other compatible devices, see *IMS/ESA General Information*. Though the 3600 devices are included in the FIN series, you can specify them with their 36xx designations; MFS messages use the FIxx designations regardless of which form of designation you specify. In general, however, application designers and programmers using this book only need to know how the devices they are defining control blocks for have been defined to the IMS system in their installation.

Table 16. Terminal Devices That Operate with MFS

Device	Devices Defined by Number [1]	NTO Devices [2]	SNA Devices or Logical Units [3]				
			SLU 1	SLU 2	SLU 4	SLU P	LU 6.1
3180	X [4]			X [4]			
3270	X [4]			X [4]			
3290	X [4]			X [4]			
5550	X [4]			TYPE: 3270-An 3270-Ann			
3270 printers; 5553, 5557	X [4]		COMPT*n*= MFS-SCS1				
3730					X	X	
3767			COMPT*n*= MFS-SCS1				
3770 console, printers, print data set			COMPT*n*= MFS-SCS1			X	
3770 readers, punches, transmit data set			COMPT*n*= MFS-SCS2			X	
3790 print data set (bulk)			COMPT*n*= MFS-SCS1			COMPT*n*= MFS-SCS1 DPM-An	
3790 transmit data set			COMPT*n*= MFS-SCS2				

Table 16. Terminal Devices That Operate with MFS (continued)

Device	Devices Defined by Number [1]	NTO Devices [2]	SNA Devices or Logical Units [3]				
			SLU 1	SLU 2	SLU 4	SLU P	LU 6.1
3790 attached 3270				X [4]			
6670					X		
8100						X	
8100 attached 3270			X	X [4]			
8100 attached Series/1						X	
8100 attached S/32			X				
8100 attached S/34						X	
8100 attached S/38			X				
Finance	X					COMPT*n*= MFS-SCS1 DPM-An	
TTY		X					
3101		X					
Other systems (IMS to IMS or IMS to other)							COMPT*n*= DPM=Bn

Notes:

1. With options= (...,MFS,...) in the TERMINAL or TYPE macro.
2. Defined with UNITYPE= on the TYPE macro and PU= on the TERMINAL macro.
3. Defined by logical unit type or logical unit type with COMPT*n*= or TYPE= in the TERMINAL macro or ETO logon descriptor. The LU 6.1 definition refers to ISC subsystems.
4. Defaults to operate with MFS.

Definitions for SLU 1 and SLU 4 can specify MFS operation with SNA character strings (SCS) 1 or 2. SCS1 designates that messages are sent to a printer or the print data set or received from a keyboard in the 3770 Programmable or 3790 controller disk storage; SCS2 designates that messages are sent to or received from card I/O or a transmit data set.

Terminals defined as SLU 2 have characteristics like the 3270, and like the 3270, can be defined to operate with MFS. In general, a 3290 terminal operates like a 3270 terminal, and references to 3270 terminals in this book are applicable to 3290 devices. However, 3290 partitioning and scrolling support is only provided for 3290 devices defined to IMS as SLU 2.

Generally, the 3180 and 5550 terminals operate like a 3270 terminal, and references to 3270 terminals also apply to these devices. Likewise, the 5553 and 5557 printer devices operate like a 3270P.

Restriction: 5550 Kanji support is only provided for the 5550 terminal defined as an SLU 2 and for the 5553 and 5557 defined as SCS1 printers.

If IMS is to communicate with the user-written remote program in a 3790 or an FIN controller, the device must be defined as an SLU P. Definitions for SLU P must specify MFS operation as either MFS-SCS1 or DPM-An, where DPM means distributed presentation management and An is a user-assigned number (A1 through A15).

MFS Devices and Logical Units

Most of the MFS formatting functions currently available to other devices, except specific device formatting, are available to the user-written program. Under user control, these formatting functions (such as paging) can be divided between MFS and the remote program.

Using Distributed Presentation Management (DPM)

With distributed presentation management (DPM), formatting functions usually performed by MFS are distributed between MFS and a user-written program for SLU P devices or ISC nodes. If the 3790 or FIN controller has previously been defined to IMS by unit number, some changes must be made to convert to DPM.

With DPM, the physical terminal characteristics of the secondary logical unit do not have to be defined to MFS. MFS has to format only the messages for transmission to the user program in the remote controller or ISC node, which must assume responsibility for completing the device formatting, if necessary, and present the data to the physical device it selects.

For remote programs using DPM, the data stream passing between MFS and the remote programs can be device independent. The messages from the IMS application program can include some device control characters. If so, the IMS application program and the data stream to the remote program might lose their device independence.

If IMS is to communicate with other subsystems (such as IMS, CICS or user-written), the other subsystem must be defined as an ISC subsystem. Definitions for ISC must:

- Specify MFS operation as DPM-Bn, where DPM is as described above and Bn is a user-assigned number (B1 through B15).
- Define TYPE:LUTYPE6 on the TERMINAL macro during system definition.

DPM with ISC provides:

- Output paging on demand that allows paging to be distributed between IMS and another system
- Automatically paged output that allows MFS pages to be transmitted to another system without intervening paging requests
- Transaction routing that allows application programs to view the routing information when it is provided in the input message

Chapter 7. Message Formatting Functions

This chapter describes the message formatting functions of MFS. It elaborates on the control blocks introduced in Chapter 1. How Application Programs Work with the IMS Transaction Manager. It also explains how the control blocks format messages for different device types.

In this Chapter:

- "Input Message Formatting"
- "General Rules for Multiple DPAGE Input" on page 200
- "3270 and SLU 2 Input Substitution Character" on page 201
- "Input Format Control for ISC (DPM-Bn) Subsystems" on page 201
- "Output Message Formatting" on page 203
- "Output Format Control for ISC (DPM-Bn) Subsystems" on page 228
- "Your Control of MFS" on page 235
- "MFS Format Sets Supplied by IMS" on page 246
- "MFS Formatting for the 3270 or SLU 2 Master Terminal" on page 247
- "MFS Device Characteristics Table" on page 249
- "Version Identification Function for DPM Formats" on page 250

Input Message Formatting

This section describes how MFS is selected, and how MFS formats input messages, with examples of input messages before and after formatting.

How MFS Is Selected

Only input data from devices that are defined to IMS TM as operating with MFS can be processed by MFS. However, the use of MFS for specific input messages depends on the message content and, in some cases, on the previous output message.

274X, 3770, SLU 1, NTO, and SLU 4

For MFS to process data from a 274X, 3770, SLU 1, NTO, or SLU 4, these devices must be defined to operate with MFS at IMS TM system definition or with user descriptors if the extended terminal option (ETO) is available.

Related Reading: For more information on ETO, see *IMS/ESA Administration Guide: Transaction Manager*.

After the device is defined to operate with MFS, the terminal still operates in unformatted mode (using basic edit, not MFS) until one of the following occurs:

- //*midname* is entered and sent to IMS.
- An output message to the terminal is processed using a message output descriptor (MOD) that names a message input descriptor (MID) to be used to process subsequent input data.

When //*midname* is received, MFS gets control to edit the data using the named MID. If any data follows //*midname* (//*midname* must be followed by a blank when data is also entered), MFS discards the //*midname* and the blank and formats the data according to the named MID. If no data follows //*midname*, MFS considers the next line received from the terminal to be the first line of the message.

© Copyright IBM Corp. 1974, 2000

When an output message is processed by a MOD that names a MID, the MID is used to format the next input from that terminal. This output message can be created by an application program, the IMS TM /FORMAT command, a message switch, or some other IMS TM function.

Related Reading: For more information about the /FORMAT command, see the *IMS/ESA Operator's Reference*.

Once in "formatted mode" (using MFS, not IMS TM basic edit), the device continues to operate in formatted mode until one of the following occurs:

- // or //b (// followed by a blank) is received. The terminal returns to unformatted mode and the // (and blank) are discarded. The two slashes are escape characters.
- //b and data are received. The terminal is returned to unformatted mode, the // blank is discarded, and the data is formatted by IMS TM basic edit.
- An output message whose MOD does not name a MID is sent to the terminal.

3270 and SLU 2

All 3270 and SLU 2 devices are automatically defined to operate with MFS.

Exception: Situations in which 3270 and SLU 2 devices do not operate in formatted mode are:

- When first powered on
- After the CLEAR key is pressed
- When the MOD used to process an output message does not name a MID to use for the next input data received
- When MFS is bypassed by the application program using the DFS.EDT or DFS.EDTN modname

While in unformatted mode, input is limited to IMS TM commands, terminal test requests for BTAM (3270 only) or VTAM, paging requests, and transaction code or message switch data that does not require MFS.

Finance and SLU P Workstations

For MFS to process data from a Finance or SLU P workstation, the terminal must be defined to operate with MFS at IMS TM system definition or with user descriptors if ETO is available. For more information on ETO, see *IMS/ESA Administration Guide: Transaction Manager*. Even when so defined, the workstation operates in unformatted mode (using IMS TM basic edit, not MFS) until one of the following occurs:

- The Finance or SLU P workstation remote application program requests MFS formatting by specifying the name of a MID in the input message header.
- //*midname* is entered by a workstation operator and is sent to IMS TM by the remote application program as the first or only part of the input message itself.

  For proper SLU P formatting, include in the input message header a version identification (version ID). The version ID ensures that the correct level of MFS descriptor (Device Input Format, or DIF) is provided in mapping the input message. If this verification is not desired, the version ID can be sent with hexadecimal zeros (X'0000') or it can be omitted from the message header. For the specification of the version ID and additional details, see "Version Identification" on page 235.

  Processing occurs as described for the 274X.

When an output message sent to an SLU P or Finance workstation is formatted using a MOD that names a MID, IMS TM sends the name of the MID to the workstation as part of the output message header. Because IMS TM does not have direct control of the terminal devices in these systems, IMS TM cannot guarantee the proper MID is used to process the next input. It is the responsibility of the remote program to save the MID name and to include it in the next input message it sends to IMS TM as the DPN.

Finance and SLU P workstations continue in formatted mode only when the current message has an associated MID or MOD.

Intersystem Communication (ISC) Subsystems

For data from an ISC subsystem to be processed by MFS, the ISC subsystem must be defined as UNITYPE=LUTYPE6 on the TYPE macro at IMS TM system definition or with ETO user descriptors. For more information on ETO, see *IMS/ESA Administration Guide: Transaction Manager*. Even when so defined, the ISC subsystem operates in unformatted mode (using IMS TM basic edit or ISC edit, not MFS) until the ISC application program requests MFS formatting by specifying the name of a MID in the DPN field of the input message header.

When an output message sent to an ISC subsystem is formatted using a MOD that names a MID, IMS TM sends the name of the MID to the ISC subsystem in the RDPN field of the output message header. Because IMS TM does not have direct control of the ISC subsystem, IMS TM cannot guarantee the proper MID is used to process the next input. It is the responsibility of the ISC application program to save the MID name and to include it in the next input message it sends to IMS.

ISC subsystems continue in formatted mode only when the current message has an associated MID or MOD.

Related Reading: For an overview of ISC, see *IMS/ESA General Information*. For more detailed information, see *IMS/ESA Administration Guide: Transaction Manager*.

Formatting Messages from Terminals in Preset Destination Mode

Preset destination mode is used to fix a destination for all messages entered from a terminal. Use the /SET command to enter preset destination mode (/SET is described in *IMS/ESA Operator's Reference*). When a terminal is in preset mode, all input messages (processed by either MFS or basic edit) are routed to the destination established by the /SET command. You do not have to include the message destination in the input message.

When IMS TM basic edit processes input from a preset terminal, the preset destination name is added to the beginning of the first segment. When MFS processes input from a preset terminal, the preset destination name is not added to the beginning of the first segment; input message format is a result of your message definition and input. MFS provides many methods for reserving space in an input segment or for inserting a transaction code, without requiring you to specify a message destination.

Formatting of Messages Using Fast Path

If you plan to implement Fast Path, MFS functions like other IMS TM applications, with the restriction that all messages must be single-segment messages.

How MFS Formats Input Messages

Input data from MFS-supported devices in formatted mode is formatted based on the contents of two MFS control blocks—the message input descriptor (MID) and the device input format (DIF). The MID defines how the data should be formatted for presentation to the IMS TM application program and points to the DIF associated with the input device. The DIF describes the data as the data is received from the device.

If the message built by the MID is a command, the command must conform to the command format and syntax rules as documented in *IMS/ESA Operator's Reference*.

Input Message Formatting Options

MFS supports three message formatting options. The option selected determines how MFS interprets the MID definition and thereby formats the data into message fields for presentation to the application program. The MID's MFLD statement or statements describes message fields in terms of:

- Length
- The device field from which input data is to be obtained
- Literal data for message fields which will not or do not receive device data
- Fill characters to use when the input data does not fill the message field
- Field justification (left or right) or truncation (left or right) specifications
- Whether the first 2 bytes of the field should be reserved for attribute data

The formatting option is specified in the MID's MSG statement (OPT=). The selection of the proper option for an application is a design decision that should be based on the complexity and variability of the device data stream, the programming language used, and the complexity of the program required to process the application under a given option. In the following discussion, a NULL character is X'3F'.

Option 1: The effect of option 1 depends on whether a fill character of NULL has been defined. When no field in an option 1 message is defined to the MFS Language utility as having a fill character of NULL:

- Messages always contain the defined number of segments.
- Each segment is always of the defined length and contains all defined fields.
- All fields are filled with data, data and fill characters, or fill characters.

When fields in an option 1 message are defined as having a fill character of NULL:

- Each field with null fill and no input data from the device is eliminated from the message segment. If all fields in a segment are eliminated in this manner and no literals (explicit or default) are defined, the segment is eliminated; otherwise, the length of the segment is reduced and the relative position of succeeding fields in the segment is altered.
- Fields with null fill that receive device data that does not fill the field are not padded—the number of characters received for the device field becomes the number of characters of the input data. This alters the length of the segment and the relative position of all succeeding fields in the segment.

Option 2: Option 2 formatting is identical to option 1 unless a segment contains no input data from the device after editing. If this occurs and there are no more segments containing input data from the device, the message is terminated, and the last segment in the message is the last segment that contained input data from the

device. If a segment is created that has no input data from the device, but there are subsequent segments that do contain data from the device, a segment is created with a single byte of data (X'3F') signifying that this is a pad or null segment. If this occurs on a first segment that is defined to contain a literal, an invalid transaction code could result because MFS does not insert explicit or default literals into segments for which no device input data is received.

Option 3: Option 3 formatting supplies the program with only the fields received from the input device. A segment is presented only if it contains fields that were received from the device. Segments are identified by a relative segment number and fields within a segment are identified by a segment offset. Segments and fields are both of variable length if they are described as having a fill character of NULL. Empty fields (fields without data) are not padded with fill characters. Segments that are presented to the application program appear in relative segment number sequence. Fields within the segment are in segment offset sequence.

Option 3 messages do not contain literals (explicit or default) specified in the MID.

If option 3 is used with conversational transactions, the transaction code is not removed from the message, since fields and offsets of fields are maintained within the text. The transaction code is still found in the SPA also.

Restriction: You cannot use option 3 input message formats to enter IMS TM commands. However, IMS TM commands can be entered by using IMS-supplied default formats, from the cleared screen, or from your defined option 1 and option 2 input message formats.

Examples

The following examples illustrate the message segment definitions, then for options 1, 2, and 3, the contents, length in bytes, and a code for the type for each field.

The field types are labeled as shown in Table 17.

Table 17. Input Message Field Types

Type Code	Description
A	Total segment length, including fields A, B, C. 2 bytes, binary
B	Z1 field—reserved for IMS TM usage
C	Z2 field—indicates formatting option 1 byte, binary
D	Relative segment number 2 bytes, binary
E	Field length, including length of fields E, F 2 bytes, binary
F	Relative field offset in the defined segment 2 bytes, binary
G	Field

Notes:

1. No boundary alignment is performed for fields A, D, E, or F.
2. Fields A, B, and D must be on halfword boundaries. To do this, ensure the I/O area is on a boundary when the `GU` or `GN` call to IMS TM is made.
3. For the PLITDLI interface, the length (LL) field must be declared as a binary fullword. The value in the LL field is the segment length minus 2 bytes. For example, if the input message segment is 16 bytes, LL is 14 bytes, which is the sum of the lengths of LL (4 bytes minus 2 bytes), ZZ (2 bytes), and TEXT (10 bytes).

Example 1, Input Message Format: **Message definition**.

Input Message Formatting

```
Segment 1   |72|  |TRANCD(8)| MAN NO. (10) | NAME (50)   |
            -------------------------------------------

Segment 2   |59|  | DEPT (5) | LOCATION (50)   |
            -----------------------------------

Segment 3   |64|  | PART NO. (10)    | DESCRIPTION (50) |
            ---------------------------------------------

Segment 4   |19|  | QUANTITY (10) | ORDER PRIORITY (5) |
            ---------------------------------------------
```

All fields defined as left justified, with a fill character of blank.

You enter:

Field Name	**Input**
NAME	ABJONES
PART NO.	23696
DESCRIPTION	WIDGET

Transaction code is provided from the message input description as a literal. The input message would appear to the application program as one of the following:

If you specified option 1:

```
CONTENTS    |0072|XX|01|TRANCD  | blanks |ABJONES           |
            ------------------------------------------------
LENGTH       2   1  1  8         10     50
TYPE         A   B  C


CONTENTS    |0059|XX|01| blanks     | blanks              |
            ------------------------------------------------
LENGTH       2    1  1   5            50
TYPE         A    B  C


CONTENTS    |0064|XX|01|23696       |WIDGET               |
            ------------------------------------------------
LENGTH       2    1  1  10           50
TYPE         A    B  C


CONTENTS    |0019|XX|01| blanks | blanks |
            ----------------------------
LENGTH       2    1  1   10       5
TYPE         A    B  C
```

If you specified option 2:

```
CONTENTS    |0072|XX|02|TRANCD  |  blanks  |ABJONES          |
            ------------------------------------------------
LENGTH       2   1  1  8          10      50
TYPE         A   B  C


CONTENTS    |0005|XX|02|3F|
            --------------
LENGTH       2    1  1  1
TYPE         A    B  C


CONTENTS    |0064|XX|02|23696      |WIDGET                |
            ------------------------------------------------
LENGTH       2    1  1  10          50
TYPE         A    B  C
```

If you specified option 3:

```
CONTENTS    |0060|XX|03|0001|0054|0022|ABJONES               |
            ------------------------------------------------
LENGTH       2   1  1  2    2   2    50
TYPE         A   B  C  D    E   F    G


CONTENTS    |0074|XX|03|0003|0014|0004|23696  |0054|0014|WIDGET  |
            ------------------------------------------------------
LENGTH       2    1  1  2    2    2    10      2    2    50
TYPE         A    B  C  D    E    F    G       E    F    G
```

The option 3 example shows no transaction code in the first segment because literals are not inserted into option 3 segments. This message would be rejected unless it is received from a terminal in conversational or preset destination mode, because transaction code validation is performed after the messages are formatted.

Example 2, Input Message Format: Segments are similar to example 1. Fields are defined as in example 1 except for the following:

Field Name	**Contents**
NAME	null pad
DEPT	null pad
LOCATION	null pad
PART NO.	right justify, pad of EBCDIC zero
QUANTITY	null pad

You enter:

Field Name	**Input**
NAME	ABJONES
PART NO.	23696
DESCRIPTION	WIDGET
PRIORITY	HI

Transaction code is provided as a 3270 program function key literal or a special data field from a 274X or Finance workstation. The input message appears to the application program as one of the following:

If you specified option 1:

```
CONTENTS    |0029|XX|01|TRANCD  |  blanks    |ABJONES|
            ---------------------------------------
LENGTH       2    1  1 8          10        7
TYPE         A    B  C
```

No second segment is presented because all of its fields were null padded and no input data was received from the device for these fields.

```
CONTENTS    |0064|XX|01|0000023696|WIDGET                  |
            ------------------------------------------------
LENGTH       2   1  1  10         50
TYPE         A   B  C


CONTENTS    |0009|XX|01|HI      |
            ----------------------
LENGTH       2    1  1  5
TYPE         A    B  C
```

If you specified option 2:

```
CONTENTS    |0029|XX|02|TRANCD |blanks    |ABJONES|
            -----------------------------------
LENGTH       2   1  1  8       10        7
TYPE         A   B  C


CONTENTS    |0005|XX|02|3F|
            ---------------
LENGTH       2    1  1  1
TYPE         A    B  C


CONTENTS    | 0064|XX|02|0000023696|WIDGET                |
            ------------------------------------------------
LENGTH        2    1  1  10         50
TYPE          A    B  C


CONTENTS    |0009|XX|02|HI    |
            --------------------
LENGTH       2    1  1  5
TYPE         A    B  C
```

If you specified option 3:

```
CONTENTS    |0029|XX|03|0001|0012|0004|TRANCD  |0011|0022|ABJONES|
            ------------------------------------------------------
LENGTH       2    1  1  2    2    2    8        2    2    7
TYPE         A    B  C  D    E    F    G        E    F    G


CONTENTS    |0074|XX|03|0003|0014|0004|0000023696|0054|0014|WIDGET  |
            ------------------------------------------------------------
LENGTH       2    1  1  2    2    2    10         2    2    50
TYPE         A    B  C  D    E    F    G          E    F    G


CONTENTS    |0015|XX|03|0004|0009|0014|HI   |
            --------------------------------
LENGTH       2    1  1  2    2    2    5
TYPE         A    B  C  D    E    F    G
```

Cursor Position Input and FILL=NULL

With MFS, a problem might arise when the application program is told the cursor position on input. This problem occurs when:

- The input message uses formatting option 1 or 2.
- The MFLD used for cursor position data is defined in a segment where at least one MFLD is defined to use null fill (FILL=NULL).

When these conditions occur, cursor position 63 (X'3F') results in a 3-byte field containing compressed cursor data, rather than a normal 4-byte field. The MFLD with this potential problem is flagged with the message “DFS1150”.

To avoid this problem, change the MFLD statement for the cursor data field to specify EXIT=(0,2). This will cause the IMS TM-provided field edit routine to convert the field contents from binary to EBCDIC. The application program must also be changed to handle the EBCDIC format.

Input Logical Page Selection

An input logical page (LPAGE) determines the content of the input message that is presented to the application program. It consists of a user-defined group of related message segment and field definitions. An input LPAGE is identified by an LPAGE statement. When no LPAGE statement is present, all message field definitions in the MSG are treated as a single LPAGE. An input LPAGE identified by an LPAGE statement can refer to one or more input device pages (DPAGE).

An input DPAGE defines a device format that can be used for an input LPAGE. It consists of a user-defined group of device field definitions. An input DPAGE is identified by a DPAGE statement. When no DPAGE statement is present, all device field definitions following the DIV statement are treated as a single DPAGE. If multiple DPAGEs are defined, each DPAGE statement must be labeled. A DPAGE identified by a labeled DPAGE statement must be referred to by an LPAGE statement.

3270 and SLU 2 device input data is always processed by the currently displayed DPAGE. For other devices, if multiple DPAGEs are defined in their formats, a conditional test is performed on the first input record received from the device. The results of this test determine which DPAGE is selected for input data processing. The LPAGE that refers to the selected DPAGE is used for input message formatting.

If input LPAGEs are not defined, message fields can refer to device fields in any DPAGE, but input data from the device for any given input message is limited to fields defined in a single DPAGE.

Input Message Field and Segment Edit Routines

To simplify programming, MFS application designers should consider using (for all but SLU P devices) input message field and segment edit routines to perform common editing functions such as numeric validation or conversion of blanks to numeric zeros. While use by existing applications is unlikely, field and segment edit routines can simplify programming of new applications by using standard field edits to perform functions that would otherwise need to be coded in each application program. *IMS/ESA Customization Guide* lists the field and segment edit routines provided by IMS. The input message field or segment exit routines can be disabled for SLU P (DPM-An and ISC) devices, because editing is probably done by the remote program.

Using field and segment edit routines causes extra processing in the IMS TM control region and, if used extensively, creates a measurable performance cost. However, these edit routines can improve performance by reducing processing time in the message processing region, reducing logging and queuing time, and by allowing field verification and correction without scheduling an application program. Efficiency of these user-written routines should be a prime concern.

Because these routines execute in the IMS TM control region, an abend in the edit routine causes an abend of the IMS TM control region.

IMS-Supplied Field and Segment Edit Routines: IMS TM provides both a field and a segment edit routine that the MFS application designer might want to use. *IMS/ESA Customization Guide* lists the IMS-supplied routines.

Under MVS/ESA, any code written to replace these IMS-supplied routines must be able to execute in RMODE 24, AMODE 31 and be capable of 31-bit addressing even if they do not reference any 31-bit addressable resources. AMODE refers to addressing mode; when running modules in AMODE 31, Extended Architecture processors interpret both instruction and data addresses to be 31 bits wide.

Related Reading: For more information on running modules under MVS/ESA, refer to *MVS/ESA JES3 Conversion Notebook* .

Field Edit Routine (DFSME000): The functions of the field edit routine are based on the entry vector. It can use all three formatting options. For options 1 and 2, entry vector 1 can produce undesirable results if FILL=NULL was specified in the MFLD statement.

Input Message Literal Fields

Input message fields can be defined to contain literal data that you specify during definition of the MID:

- You can define a default literal that MFS always inserts as part of the input message.
- You can define a literal that MFS inserts as part of the input message when no data for the field is received from the device.

Using a default literal can simplify application programming. When used, application programs no longer need to test for “no data” conditions or to provide exception handling. Default literals make it possible for an application program to distinguish between zero-value data you enter and a condition of “no data entered”.

Example: Consider the following MFLD definition:

```
MFLD (DFLD1,'NO DATA'),LTH=7,JUST=R,FILL=C'0'
```

For example, an application program would view your entries as follows:

Your Entry	Program Data Viewed
296	0000296
0	0000000
no data entered	NO DATA

Without a default literal, the results of entering a value of 0 and of entering no data are the same—0000000.

Defaults can be altered without changing application programs, and multiple defaults can be provided by using different message descriptors or different input logical pages.

Default literals can also expand device independence by providing a device-independent method of inserting data in an input message field if no data is entered from the device for that field. This function of the default literal is used often for 3270 or SLU 2 devices, which have the same device format for input as for output. For these devices, the default (transaction code, data, or both) can be provided if you specify a default literal on input (MID).

Input Message Field Attribute Data

Nonliteral input message fields can be defined to allow for attribute data, extended attribute data, or both. When defined to do so, MFS initializes to blanks and reserves the first bytes of the message field for attribute or extended attribute data. These first bytes are filled in by a field edit routine or in its preparation of an output message. When attribute or extended attribute space is specified, the specified field length must include space for the attribute or extended attribute bytes.

Sometimes input messages are updated by an application program and returned to the device. The application program can simplify message definitions if the message uses attribute data as the output message, and the attribute data bytes are defined in the input message, also.

When a field edit routine is used, it can be designed (as the IMS-supplied field edit routine is) to set attribute bytes on fields in error. In this way, erroneous fields can be highlighted before the segment edit routine returns the message to the device. In this case, the application program is not concerned with attribute bytes.

IMS TM Password

The IMS TM password portion of an input message is defined in the input message definition. One or more input message fields can be defined to create the IMS TM password. Using this method of password definition allows passwords to be created from field data you enter, from data read by a 3270, SLU 2, 3770 operator identification card reader, or data from a 3270 magnetic stripe reader.

Recommendation: If you use an SLU 2 or a 3270, you can also define a specific device field as the location of the IMS TM password, but the method above is recommended and takes precedence if both an input message field and a device field are defined.

Fill Characters for Input Message Fields

MFS uses fill characters to pad message fields when the length of the data received from the device is less than the specified field length, no data for the field is received and no default literal is defined, or the data received from SLU P contains nulls and NULL=DELETE is specified. The fill characters that can be selected are a blank (X'40'), any EBCDIC hexadecimal character (X'hh'), or an EBCDIC graphic character (C'c'). Null compression, which causes compression of the message to the left by the amount of missing data, can also be selected. How MFS actually pads the message fields is a function of the selected fill character and the message formatting option being used (refer to “Input Message Formatting Options” on page 186).

Input Modes (Devices Other Than 3270, SLU 2, or ISC Subsystems)

MFS expects input message fields to be entered in the sequence in which they were defined to the MFS Language utility program. For devices other than SLU 2 and 3270, MFS application designers have a choice of how fields are defined and how MFS should scan those fields. You can select record mode or stream mode. Record mode is the default.

In record mode:

- Input fields are defined as occurring within a specific record (a line or card from the 274X, 3770, or SLU 1 or SLU 4; a transmission from the Finance or SLU P workstation) that is sent from the input device.
- Fields must not be split across record boundaries.
- Fields defined within a record must appear on that record to be considered by MFS.
- When MFS locates the end of a record, the current field is terminated and any other fields defined for that record are processed with no device data (message fill).
- If the record received by IMS TM contains more data fields than the number of fields defined for the record, the remaining data fields are not considered by MFS.

For input data from a Finance or SLU P workstation remote program, the input message header or //*midname* can be transmitted separately if the data fields for the first record do not fit in the same record. If no data follows the input message header or the //*midname*, MFS considers the next transmission received to be the first record of the input message.

In stream mode:

- Fields are defined as a contiguous stream of data unaffected by record boundaries.
- Fields can be split across input records and fields can be entered from any input record as long as they are entered in the defined sequence.

Input Field Tabs (Devices Other Than 3270 or SLU 2)

An input field tab (FTAB) is a character defined in the DEV statement for separating input fields if the length of the data entered is less than the defined field length, or for when no data is specified for a field. An FTAB causes the MFS input scan to move to the first position of the next defined field. FTABs can be defined only for input from devices other than the 3270 or SLU 2. When no FTABs are defined, each device input field is assumed to be of its defined length.

Select a character for input field separation that is never used for other user data in the data stream. If FTAB is not unique, the data might be misinterpreted by MFS.

Example: Figure 35 shows some DFLD field definitions and the device format that results from these definitions.

```
   Field Definition        MODE=STREAM     MODE=RECORD

     A         DFLD        POS=1,LTH=4     POS=(1,1),LTH=4
     B         DFLD        POS=6,LTH=3     POS=(1,6),LTH=3
     C         DFLD        POS=9,LTH=5     POS=(1,9),LTH=5

'Device Format'
```

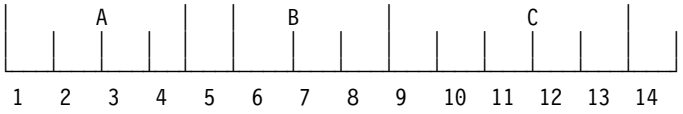


Figure 35. FTAB Qualification Descriptions

When an FTAB is defined, its use is qualified by specifying FORCE, MIX, or ALL. See Figure 36 on page 196 for how the descriptions in Figure 35 are read. Figure 35 shows how the FTAB qualification affects the results of an MFS input scan following variable operator input of a three-field message.

The shaded boxes in Figure 36 on page 196 indicate undesirable results. The double-headed arrows indicate that the FTAB qualification does not affect input scan. Input examples 2, 3, and 6 produce correct results using any of the FTAB qualifications but example 8 does not produce correct results regardless of FTAB qualifications.

FORCE: FORCE is the default value. Each device input field is assumed to be of its defined length until an FTAB is encountered. When the first FTAB is encountered, it signifies the end of data for the current field. The byte of data following the FTAB is considered the first byte of the next field. In record mode, all subsequent fields in the current record require an FTAB. In stream mode, all subsequent fields require an FTAB. FTABs used on subsequent fields indicate that the character following the FTAB is the first for the next defined field. (This is as if ALL were specified).

In Figure 36 on page 196, examples 1, 2, 3, 5, 6, and 7 produce the desired result. Example 4 fails because no FTAB is supplied following field B (compare with example 5). Example 8 fails because no FTABs are entered, the 0 is occupying the blank (undefined) position, and subsequent fields are thus incorrect (compare with example 1).

MIX: Each device input field is assumed to be of its defined length until an FTAB is encountered. When the first FTAB is encountered, it signifies the end of data for the current field. The byte of data following the FTAB is considered the first byte of the next field. Subsequent fields of the defined length do not require an FTAB; if one is entered and the next field is contiguous (like fields B and C in the example), undesirable results occur (see example 5). Mixed FTABs operate just like a typewriter with tab stops set at the first position of each defined field (columns 1, 6, and 9 in the example).

In Figure 36 on page 196, examples 1, 2, 3, 4, 6, and 7 produce the desired result. Example 5 fails because field B is of its defined length and does not require an

FTAB; the FTAB is interpreted to indicate no data for field C (compare with example 4). Example 8 fails because no FTABs are entered, the 0 is occupying the blank (undefined) position, and subsequent fields are thus incorrect (compare with example 1).

ALL: When ALL is specified, each device input field must be terminated by an FTAB regardless of whether it is greater than, less than, or equal to the defined length. When an FTAB is encountered, it signifies the end of data for the current field. The byte of data following the FTAB is considered the first byte of the next field.

In Figure 36, examples 2, 3, 5, and 6 produce the desired result. Examples 1, 4, 7, and 8 fail because the required FTABs are not entered.

	OPERATOR INPUT EXAMPLE	FIELD	MFS DFLD DATA					
			FTAB=(',',FORCE)		FTAB=(',',MIX)		FTAB=(',',ALL)	
			LTH	DATA	LTH	DATA	LTH	DATA
1	PART␣02315231	A	4	PART	4	PART	4	PART
		B	3	023	3	023	0	
		C	5	15231	5	15231	0	
2	P1,23,15231	A	2	P1	2	P1	2	P1
		B	2	23	2	23	2	23
		C	5	15231	5	15231	5	15231
3	PART,23,15	A	4	PART	4	PART	4	PART
		B	2	23	2	23	2	23
		C	2	15	2	15	2	15
4	PART,02315231	A	4	PART	4	PART	4	PART
		B	3	023	3	023	3	023
		C	0		5	15231	0	
5	PART,023,15231	A	4	PART	4	PART	4	PART
		B	3	023	3	023	3	023
		C	5	15231	0		5	15231
6	PART,,15231	A	4	PART	4	PART	4	PART
		B	0	15231	0	15231	0	15231
		C	5		5		5	
7	PART␣,15231	A	4	PART	4	PART	4	PART
		B	0	15231	0	15231	3	152
		C	5		5		0	
8	PART02315231	A	4	PART	4	PART	4	PART
		B	3	231	3	231	0	
		C	4	5231	4	5231	0	

Note: In this example, the comma is used as the FTAB character.

Figure 36. MFS Input Scan When FTABs Are Defined with FORCE, MIX, and ALL

Optional Deletion of Null Characters for DPM-An

MFS provides for optional deletion of trailing null characters in transmission records and input data fields from SLU P (DPM-An) remote programs. (A null character is a hexadecimal zero, X'00'.) In the DIV statement, the device input format can specify NULL=KEEP or NULL=DELETE. NULL=DELETE means that MFS scans data fields and transmission records for trailing nulls and deletes them. KEEP is the default and means that MFS leaves trailing nulls in the data and treats them as valid data characters.

If trailing null characters have been replaced by fill characters by the remote program, MFS treats the fill characters as valid data characters.

When NULL=DELETE is specified, nulls at the end of a record are deleted before the data fields are scanned. In record mode, the end of the record is determined either by the FTAB or by the first other non-null character found (searching backward from the end of the record). In stream mode, trailing nulls at the end of the record are deleted only if an FTAB indicates the end of the record; otherwise, the record is handled as received from the remote program.

During the data field scan, the first trailing null character encountered in the field signifies the end of the data for the current field. The data is edited into the message field using the message fill character to pad the field if required. If the entire field contains nulls (such as nulls at the end of the record), the entire message field is padded with the specified fill character.

The scan for trailing null characters within fields is performed for each record transmitted. If an FTAB character is encountered in the current record being processed, the scan for trailing nulls characters within fields is discontinued for that record and resumes with the next record.

Transmitting null characters to IMS TM, and the delete operation, are both costly in execution time. Weigh the relative costs when you decide whether to use the NULL=DELETE option or to delete the nulls via the remote program. You must also consider the effects of the FTAB options FORCE, MIX, and ALL. These costs are affected by the following:

- When FTAB=ALL is specified with NULL=DELETE, only trailing null characters at the end of the record can be removed by MFS.
- In stream mode, with NULL=DELETE, an FTAB should be used to show an omitted field at the end of a record. Otherwise, nulls (equal to the number of characters defined for the field or fields) must be transmitted.
- If FTABs are specified and NULL=DELETE, nulls and FTABs can be mixed. FTABs can be used for one record, nulls for the next. The nulls are removed from the record with no FTABs. With FTABs in the record, null characters are treated as data.
- With NULL=DELETE, binary data that might contain valid trailing hexadecimal zeros (not intended as null characters) must be preceded by an FTAB character for a previous field to prevent deletion of the trailing X'00'.

Examples of Optional Null Character Deletion for DPM-An

In the three examples that follow, the comma is the specified FTAB, X'5F' is input hexadecimal data, and characters are defined as follows:

```
X'6B'=C","
X'C1'=C"A"
X'C2'=C"B"
X'C3'=C"C"
C"b"=blank
X'40'=C"b"
```

Example 1, Input Binary Data and Nulls:

```
  Device Input Format             Message Input Definition
  INFMT FMT                       INMSG MSG  TYPE=INPUT,SOR=INFMT
        DEV TYPE=DPM-A1, FTAB=(;;MIX)   SEG
        DIV TYPE=INPUT, NULL=DELETE
        PPAGE
A       DFLD  LTH=3                     MFLD A, LTH=3
B       DFLD  LTH=2                     MFLD B, LTH=2
        FMTEND                          MSGEND
```

Input Message	Record	Field	DFLD Data	MFLD Data
(1) X'C1C2C3005F'	1	A	C"ABC"	C"ABC"
		B	X'005F'	X'005F'
(2) X'C1C26B005F'	1	A	C"AB"	C"ABb"
		B	X'005F'	X'005F'
(3) X'C1C200005F'	1	A	C"AB"	C"ABb"
		B	X'005F'	X'005F'
(4) X'C1C2C35F00'	1	A	C"ABC"	C"ABC"
		B	X'5F'	X'5F40'
(5) X'C1C26B5F00'	1	A	C"AB"	C"ABb"
		B	X'5F'	X'5F40'

Note: The X'00' (null) at the end of the record in input messages (4) and (5) is deleted before the data fields (A and B) are scanned. Therefore, the results are the same for field B, even though an FTAB (comma in this example) follows field A. If X'00' is to be considered as data for field B, an FTAB (comma in this example) should be entered following the X'5F00'.

Example 2, Record Mode Input:

```
  Device Input Format             Message Input Definition
  INFMT FMT                       INMSG MSG  TYPE=INPUT,SOR=INFMT
        DEV TYPE=DPM-A1, FTAB=(;;MIX),  SEG
            MODE=RECORD
        DIV TYPE=INPUT, RCDCTL=12,      MFLD A,LTH=3,FILL=C'*'
            NULL=DELETE
        PPAGE                           MFLD B,LTH=3,FILL=C'*'
A       DFLD  LTH=3                     MFLD C,LTH=3,FILL=C'*'
B       DFLD  LTH=3                   MFLD D,LTH=3,FILL=C'*'
C       DFLD  LTH=3                   SEG
D       DFLD  LTH=3                   MFLD E,LTH=5,FILL=C'*'
E       DFLD  LTH=5                   MFLD F,LTH=7,FILL=C'*'
F       DFLD  LTH=7                   SEG
G       DFLD  LTH=5                     MFLD G,LTH=5,FILL=C'*'
        FMTEND                          MSGEND
```

Input Message	Record	Field	DFLD Data	Segment	MFLD Data
(1) X'C10000C20000C3C3C3000000'	1	A	C'A'	1	C'A**'
		B	C'B'		C'B**'
		C	C'CCC'		C'CCC'
		D	no data		C'***'
X'C5C56BC6C66B00000000002000'		E	C'EE'	2	C'EE***'

Input Message	Record	Field	DFLD Data	Segment	MFLD Data
		F	C'FF'		C'FF*****'
X'0000000000'	3	G	no data	3	C'*****'
(2) X'C10000C20000C3C3C3'	1	A	C'A'	1	C'A**'
		B	C'B'		C'B**'
		C	C'CCC'		C'CCC'
		D	no data		C'***'
X'C5C56BC6C6'	2	E	C'EE'	2	C'EE***'
		F	C'FF'		C'FF*****'
no input record	3	G	no data	3	C'*****'

Note: In this example, no input data was entered for fields D and G. Input message 1 contains nulls in place of omitted fields. Input message 2 does not contain nulls for omitted fields, but the results are the same for both input messages.

Example 3, Stream Mode Input::

```
  Device Input Format             Message Input Definition
  INFMT FMT                       INMSG MSG  TYPE=INPUT,SOR=INFMT
        DEV TYPE=DPM-A1, FTAB=(;;MIX),  SEG
            MODE=STREAM
        DIV TYPE=INPUT, NULL=DELETE     MFLD A,LTH=3,FILL=C'*'
        PPAGE                           MFLD B,LTH=3,FILL=C'*'
A       DFLD  LTH=3                     MFLD C,LTH=3,FILL=C'*'
B       DFLD  LTH=3                     MFLD D,LTH=3,FILL=C'*'
C       DFLD  LTH=3                     SEG
D       DFLD  LTH=3                     MFLD E,LTH=5,FILL=C'*'
E       DFLD  LTH=5                     MFLD F,LTH=7,FILL=C'*'
F       DFLD  LTH=7                     SEG
G       DFLD  LTH=5                     MFLD G,LTH=5,FILL=C'*'
        FMTEND                          MSGEND
```

Input Message	Record	Field	DFLD Data	Segment	MFLD Data
(1) X'C10000C20000C3C3C3000000'	1	A	C'A'	1	C'A**'
		B	C'B'		C'B**'
		C	C'CCC'		C'CCC'
		D	no data		C'***'
X'C5C56BC6C66B000000000000'	2	E	C'EE'	2	C'EE***'
		F	C'FF'		C'FF*****'
X'00000000000000'	3	G	no data	3	C'*****'
(2) X'C10000C20000C3C3C3'	1	A	C'A'	1	C'A**'
		B	C'B'		C'B**'
		C	C'CCC'		C'CCC'
	2	D	C'EE'		C'EE*'
X'C5C56BC6C6'		E	C'FF'	2	C'FF***'
		F	no data		C'*******'
no input record	3	G	no data	3	C'*****'

Note: In this example, no input data was entered for fields D and G. Input message 1 contains nulls in place of omitted fields. Input message 2 does not contain nulls for omitted fields and produces undesirable results for fields D, E, and F.

Multiple Physical Page Input Messages (3270 and SLU 2 Display Devices)

Specifying multiple physical page input for 3270 and SLU 2 display devices allows creation of identical input messages for a transaction regardless of the physical capacity of the device being used. When this facility is used, an input message consisting of multiple physical pages can be entered using multiple physical pages of a single output logical page. If multiple physical pages are defined for output (see “Physical Paging of Output Messages” on page 207), the only action required to obtain multiple physical page input is to specify MULT=YES in the DPAGE statement.

For the 3290 Information Display Panel in partitioned mode, multiple physical page input from a single partition is supported only if the DPAGE statement for the current partition specifies MULT=YES. The multiple physical pages for a single input message must come from a single partition.

If MULT=YES is not specified on the DPAGE statement for the current partition, one physical page of a single partition constructs a single input message and the input message is restricted to a single logical page.

Input messages can be created from multiple DPAGEs. This function is available for devices other than 3270 and SLU 2.

General Rules for Multiple DPAGE Input

The following general rules apply to multiple DPAGE input:

1. If any mapped input LPAGE contains no data segments (as a result of segment routines canceling all segments, for example), the input message is rejected and an error message is sent to the other subsystem.
2. MFS echo to the input terminal is ignored.
3. MFS password creation occurs from any DPAGE, but once created, any other password is ignored. If the password is included in the attach FM header, this password is used for DPM-Bn.
4. Input message options 1, 2, and 3 apply to LPAGEs. If option 2 is requested, null segments at end of an LPAGE are eliminated. This alters the relative positions of the segments in the next LPAGE (if any) in the input message. If option 1 or 2 is requested, the first segment of the second and all subsequent LPAGEs have the page bit (X'40') in the Z2 field turned on regardless of any null segments resulting at the end of the previous LPAGE. If option 3 is requested, the segment ID is equal to 1 for every first segment in the new LPAGE.
5. Multiple DPAGE input requested in MFS definitions does not restrict message creation from the single DPAGE.
6. If your control request is entered with the first input DPAGE, the request is processed and the input message is rejected. If your control request is entered with an input DPAGE other than the first, the request is ignored and the input message is accepted.
7. If your logical page request is entered with the first input DPAGE (that is, an equals sign (=) in the first position of the input segment), the request is processed and the input message is rejected.

If multiple DPAGE input is not requested of MFS definitions, message creation from more than one DPAGE is not permitted and the following rules apply:

1. If a single transmission contains more data than defined for the DPAGE selected, the input message is rejected and an error message is sent to the other subsystem.
2. If the message has multiple transmissions, the input message is rejected and an error message is sent to the other subsystem.

3270 and SLU 2 Input Substitution Character

A X'3F' can be received on input by IMS TM from some terminals (such as by using the ERROR key). The substitution character (X'3F') provides a means of informing the host application that an error exists in the field. MFS also uses X'3F' for IMS TM functions on input data streams. To eliminate the confusion resulting from the two uses of the X'3F' characters, a parameter (SUB=) is provided on the DEV statement for use with 3270 and SLU 2 display devices.

With this parameter, a user-specified character can be defined to replace any X'3F' characters received by MFS in the 3270 and SLU 2 data stream. No translation occurs if any of the following is true:

The SUB= parameter is not specified.

The SUB= parameter is specified as X'3F'.

The input received bypasses MFS.

The specified SUB character should not appear elsewhere in the data stream, so, it should be nongraphic.

Input Format Control for ISC (DPM-Bn) Subsystems

This section describes the major input message formatting functions of MFS with ISC nodes.

Input Message Formatting

This section describes the DPAGE selection options and the creation of a message from multiple DPAGEs.

Input DPAGE Selection

The OPTIONS=(DNM) parameter on the DIV statement allows for DPAGE selection using data structure name (DSN).

If more than one DPAGE is defined, a DPAGE label must be specified in every DPAGE. If no DPAGE is selected, the message is rejected and an error message is sent to the other subsystem.

If OPTIONS=NODNM and multiple DPAGEs are defined, a conditional test is performed on the first input record. The results of the test (matching the COND= specification with the data) determines which DPAGE is selected for input data formatting. If the condition is not satisfied and all defined DPAGEs are conditional, the input message is rejected and an error message is sent to the other subsystem.

Single Transmission Chain

For single transmission chains, DPAGEs can be selected using conditional data.

DPAGE Selection Using Conditional Data: For multiple DPAGE input with single transmission chain, use the OPTIONS=NODNM parameter. The data in the first input record is used to select the first (or only) DPAGE for formatting. If the data supplied does not match any COND= defined, the last defined DPAGE is selected if

the COND= is not specified for this DPAGE. If the condition is not satisfied and all defined DPAGEs are conditional, the input message is rejected and an error message is sent to the other subsystem. If the DSN is supplied in the DD header, it is ignored. For any additional DPAGE (more data supplied than defined for the DPAGE selected), the data in the subsequent record is used to select the next DPAGE for formatting.

Multiple Transmission Chains

For multiple transmission chains, DPAGEs can be selected using DSN or by using a conditional test.

DPAGE Selection Using DSN: For multiple DPAGE input with multiple transmission chains, use the OPTIONS=DNM parameter. The DSN supplied in the DD header with each chain of the message is used to select the DPAGE for formatting. If no match is found, the message is rejected and an error message (DFS2113) is sent to the other subsystem.

DPAGE Selection Using Conditional Test on the Data: If DSN is supplied in the DD header with each chain (or any chain) of the message and OPTIONS=NODNM is specified on the DIV statement, the DSN is ignored. The data in the first record of each chain is used to select the DPAGE for formatting. If no condition is satisfied and the last defined DPAGE is unconditional (that is, COND= parameter is not specified), this DPAGE is selected for formatting. If the condition is not satisfied and all defined DPAGEs are conditional, the input message is rejected and an error message is sent to the other subsystem.

How conditional and unconditional DPAGEs are specified depends on whether OPTIONS=DNM or OPTIONS=NODNM is specified.

- For OPTIONS=DNM, conditional is specified with a label in the DPAGE statement.
- For OPTIONS=NODNM:
  - To specify conditional, specify the COND= keyword on the DPAGE statement.
  - To specify unconditional, omit the COND= keyword.

Input Modes

MFS supports two input modes: record and stream.

Record Mode

In record mode, one record presented to MFS by the ATTACH manager corresponds to one record defined to MFS. Records and fields defined for each record are processed sequentially. Fields must not be split across record boundaries. The data for fields defined in a record must be present in this record to be considered by MFS. If no data exists for fields defined at the end of the record, a short record can be presented to MFS. If the data for a field not at the end of the record is less than the length defined for the corresponding DFLD, or if no data exists for the field, then a field tab separator character must be inserted to show omission or truncation. If no data exists for the entire record, a null or a 1-byte record (containing a single FTAB character) must be present if additional data records follow it. The record can be omitted:

- At the end of the DPAGE for single DPAGE input.
- At end of the DPAGE for multiple DPAGE input with multiple transmission chains.
- At the end of the last DPAGE for multiple DPAGE input with a single transmission chain. The record cannot be eliminated from the DPAGE if data for another DPAGE follows.

Stream Mode

In stream mode, record boundaries are ignored and fields can span record boundaries. Data omitted for fields anywhere in the DPAGE must be indicated by an FTAB.

FTABs are not required for the data omitted to the end of the DPAGE:

- At the end of the DPAGE for single DPAGE input.
- At the end of the DPAGE for multiple DPAGE input with multiple transmission chains.
- At the end of the last DPAGE for multiple DPAGE input with single transmission chain. The FTABs cannot be eliminated from the DPAGE if data for another DPAGE follows.

On input to IMS, the ATTACH manager provides for four deblocking algorithms, UNDEFINED, RU, VLVB, and CHAINED ASSEMBLY, which specify the following:

- UNDEFINED or RU specify that one RU is equal to one MFS record processed. IMS TM defaults to the RU algorithm when UNDEFINED is specified in the ATTACH FM header.
- VLVB specifies that one VLVB record is equal to one MFS record processed.
- CHAINED ASSEMBLY specifies that one input chain is equal to a single MFS record processed for the entire DPAGE.

For MFS RECORD mode, use the VLVB deblocking algorithm. For MFS RECORD mode, do not use the following:

- CHAINED ASSEMBLY, because the entire input chain would be processed as a single MFS record.
- UNDEFINED or RU, because MFS record definitions would be dependent on the size of the RUs.

For the MFS STREAM mode, all deblocking options can be used. In most cases the UNDEFINED and RU algorithms use less buffer space.

Paging Requests

Use the FM headers for entering paging requests when using ISC.

Output Message Formatting

This section discusses MFS output message formatting, physical and logical paging, and requirements for output devices.

How MFS Is Selected

Whether an output message is processed by IMS TM basic edit or MFS depends on the device type, the device definition, and the message being processed.

Output messages to SLU 2 and 3270 devices are processed by MFS, unless bypassed by the application program.

Output messages to a 274X, 3770, Finance workstation, SLU 1, NTO, SLU 4, SLU P, or ISC subsystem are processed by MFS, if these devices are defined during IMS TM system definition to operate with MFS.

Even when a device is defined to operate with MFS, MFS does not process an output message unless a MOD name was specified by the application program, the

MID associated with the previous input message, or the /FORMAT command. Also, message switches from other MFS devices are processed by MFS if the message has an associated MOD.

If you attempt to access a transaction that is to be changed or deleted when the online change utility is run, and you do this after the online change command /MODIFY PREPARE has been issued but before /MODIFY COMMIT has been issued, you receive an error message. This is described in *IMS/ESA Operator's Reference*.

How MFS Formats Output Messages

Output messages processed by MFS are formatted based on the contents of two MFS control blocks: the message output descriptor (MOD) and the device output format (DOF). The MOD defines output message content and, optionally, literal data to be considered part of the output message. Message fields (MFLDs) refer to device field locations via device field (DFLD) definitions in the DOF. The device output format (DOF) specifies the use of hardware features, device field locations and attributes, and constant data considered part of the format.

Output Message Formatting Options

MFS provides three message formatting options for output data. The option selected determines how the data is formatted and governs the way in which the application program builds the output message. Option 1, 2, or 3 is specified in the OPT= operand of the MOD MSG statement. For examples of input messages formatted with the three options, see "Input Message Formatting" on page 183. Examples of output message formats are shown in "Option 1 or 2—Output Segment Example" on page 205 and "Option 3—Output Segment Example" on page 205.

Segments inserted by the application program must be in the sequence defined to the MFS Language utility program. Not all segments in a logical page must be present, but be careful when you omit segments (see "Logical Paging of Output Messages" on page 205). An option 1 or 2 segment can be omitted if all subsequent segments to the end of the logical page are omitted; otherwise, a null segment (X'3F') must be inserted to indicate segment position. Option 3 output message segments must include a 2-byte relative segment number.

Message fields in option 1 and 2 output segments are defined as fixed-length and fixed position. Fields can be truncated or omitted by two methods:

- One method is by inserting a short segment.
- The other method is by placing a NULL character (X'3F') in the field. Fields are scanned left to right for a null character; the first null encountered terminates the field. If the first character of a field is a null character, the field is effectively omitted, depending on the fill character used. Positioning of all fields in the segment remains the same regardless of null characters. Fields truncated or omitted are padded as defined to the MFS Language utility.

Message fields in option 3 segments can be placed in any order and with any length that conforms to the segment size restriction. Short fields or omitted fields are padded as defined to the MFS Language utility. Each field must be preceded by a 4-byte field prefix of the same format provided by MFS for option 3 input fields.

While option 3 fields do not have to be in sequence in the output segment, all fields must be contiguous in the segment; that is, the field prefix of the second field must begin in the byte beyond the first field's data. Null characters in option 3 fields have no effect on the data transmitted to the device. Like other nongraphic characters, they are replaced with a blank.

Restriction: Device control characters are invalid in output message fields under MFS. For 3270 display and SLU 2 terminals, the control characters HT, CR, LF, NL, and BS are changed to null characters (X'00'). For other devices, these characters are changed to blanks (X'40'.) All other nongraphic characters (X'00' through X'3F' and X'FF') are changed to blanks before transmission, with the exception of the shift out/shift in (SO/SI) characters (X'0E' and X'0F') for EGCS capable devices. (The SO/SI characters are translated to blanks only for straight DBCS fields.) An exception is allowed for SLU P (DPM-An) remote programs and ISC (DPM-Bn) subsystems, for which GRAPHIC=NO can be specified on output. If nongraphic data is allowed through this specification, the null (X'3F') cannot be used to truncate segments in options 1 and 2.

Option 1 or 2—Output Segment Example:

```
Definition              Output data length
Segment
Field, length=10        4
Field, length=20        field omitted
Field, length=5         5
Field, length=15        15
```

The segment shown produces the following results:

```
CONTENTS |54|0|0| DATA 1|*|    |*       | DATA 3 | DATA 4|
-----------------------------------------------------
LENGTH    2  1 1  4      1  5   20      5        15
```

Option 3—Output Segment Example: An option 3 segment that produces the same result appears as follows (the * represents a null (X'3F') character):

```
CONTENTS |42|0|0|04|08|04| DATA 1|09|34| DATA 3 |19|39| DATA 4|
-------------------------------------------------------------
LENGTH    2  1 1 2  2  2   4      2  2    5      2  2   15
```

The examples under “Input Message Formatting Options” on page 186 explain the sequence of fields within the segment for different formatting options.

Logical Paging of Output Messages

Logical paging is the means by which output message segments are grouped for formatting. When logical paging is used, an output message is defined with one or more logical pages (LPAGEs). Each LPAGE relates one segment, or a series of segments, produced by an application program to a corresponding device format.

Using logical paging, the simplest message definition consists of one LPAGE and one segment. As shown in Table 18, each segment produced by the application program is formatted in the same manner using the corresponding device page.

Table 18. Output Message Definition with One LPAGE Consisting of One Segment

MSG Definition	Device Page	Application Program Output
LPAGE1	DPAGE1	Segment 1
SEG1		
		or
		Segment 1
		Segment 1
		Segment 1

The next level of complexity, shown in Table 19, is a message defined with one LPAGE consisting of a series of segments. When these messages are built by the application program, the segments must be inserted in the sequence in which they were defined. Not all segments in an LPAGE have to be present, but be careful when you omit segments. An option 1 or 2 segment can be omitted if all segments to the end of the LPAGE are omitted; otherwise, a null segment must be inserted to indicate segment position. Option 3 output message segments must include the segment number identifier.

Table 19. Output Message Definition with One LPAGE Consisting of a Series of Segments

MSG Definition	Device Page	Application Program Output
LPAGE1	DPAGE1	Segment 1 [1]
SEG1		Segment 2
SEG2		⋮
⋮		Segment *n*
SEG*n*		
		Segment 1 [1]
		Segment 2
		Segment 1 [2]
		Segment 2
		⋮
		Segment *n*

Notes:

1. Page bit optional.
2. Page bit required.

Multiple series of segments can be presented to IMS as an output message. If the LPAGE is defined as having *n* segments, segment *n* +1 is edited as if it were segment 1, unless a segment with the page bit (X'40') in the Z2 field is encountered prior to segment *n* +1. When multiple series of output segments are presented and segments are omitted, the segment which begins a series must have bit 1 (X'40') of the Z2 field turned on.

A message definition with multiple LPAGEs is the most complex. Table 20 shows an example of such a definition, with application output.

Table 20. Output Message Definition with Multiple LPAGEs

MSG Definition	Device Page	Application Program Output
LPAGE1	DPAGE1	Segment 1 [1] (LPAGE1 condition specified)
SEG1		Segment 2
SEG2		⋮

Table 20. Output Message Definition with Multiple LPAGEs (continued)

MSG Definition	Device Page	Application Program Output
⋮		Segment *n*
SEG*n*		
		Segment 1 [1] (LPAGE2 condition specified)
LPAGE2	DPAGE2	Segment 2
SEG1		
SEG2		Segment 1 [1] (LPAGE2 condition specified)
		Segment 2
		Segment 1 [1] (LPAGE2 condition specified)
		Segment 1 [2] (LPAGE1 condition specified)
		Segment 2
		⋮
		Segment *n*

Notes:

1. Page bit optional.
2. Page bit required.

When multiple LPAGEs are defined, the LPAGE to be used for formatting is based on a user-defined condition present (provided by the application program) in the data of the first segment in the series. If the LPAGE to be used cannot be determined from that segment, the last defined LPAGE is used. The rules for segment omission described in “Logical Paging of Output Messages” on page 205 apply here as well.

LPAGE definitions enable specification of a MID name to use to format the input expected in response to the output logical page. If specified, this MID name overrides the name specified in the MOD’s MSG statement.

Operator Logical Paging of Output Messages

Output messages can be defined to permit operator logical paging (PAGE= operand in the MOD’s MSG statement). Use operator logical paging to request a specific logical page of an output message. For a complete description of operator logical paging and other MFS control functions see “Your Control of MFS” on page 235.

Operator logical paging is also available to your written remote program for SLU P (DPM-An) or ISC subsystem (DPM-Bn). The remote program can request IMS to provide a specific logical page of the output message.

Physical Paging of Output Messages

A logical page can be defined to consist of one or more physical pages. Physical paging allows data from a logical page to be displayed in several physical pages on the device. Physical page assignments are made in the format definition. For display devices, the size of a physical page is defined by the screen capacity (the

number of lines and columns that can be referred to). For most printer devices, a physical page is defined by the user-specified page length (number of lines) and the printer's line length.

For SLU P (DPM-An) or ISC subsystems (DPM-Bn), a physical page is defined by the user-specified paging option and the DPAGE or PPAGE statement specifying device pages or presentation pages. Physical paging allows data from a message to be transmitted to the remote program or subsystem in several presentation pages or logical pages.

Typically, a logical page has just one physical page. Multiple physical pages per logical page are generally only used when the logical page is designed for a large screen but is also to be displayed on a small screen device. The physical pages can have a totally different format from the pages defined for the large screen device. Figure 37 illustrates the use of physical paging with a message that creates one physical page on a 3277 model 2 or on a 3276/3278 with 24×80 screen size.

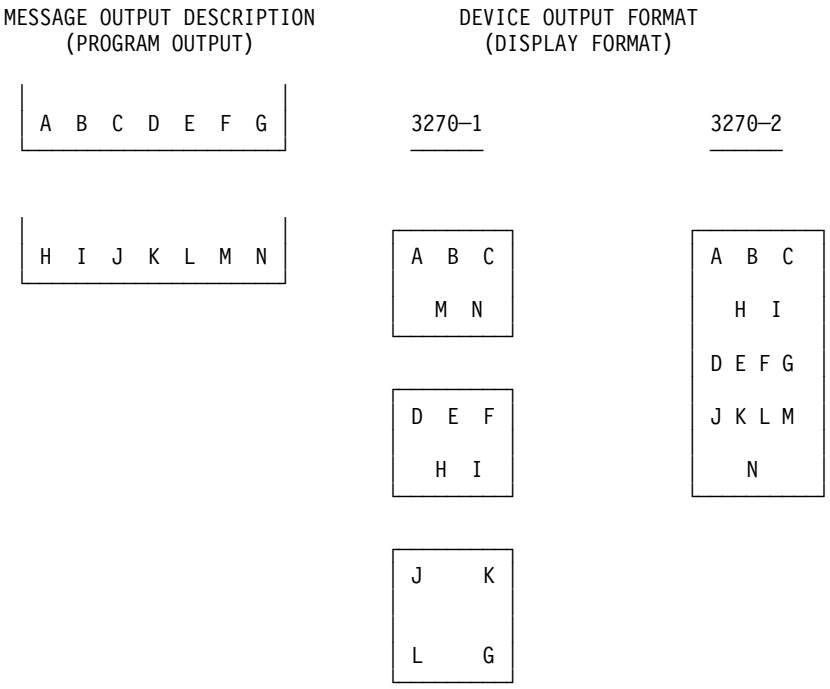


Figure 37. Physical Paging for 3270 or SLU 2

Fill Characters for Output Device Fields

MFS uses fill characters to pad output device fields when the length of the data received from the application program is less than the specified length or no data for the field is received. A fill character is defined in the message definition (MSG statement), the format definition (DPAGE statement), or both. If a fill character is specified in both, the fill character specified in the DPAGE is used. If FILL=NONE is specified in the DPAGE statement, the fill character from the MSG statement is used. The fill character specified in the MSG statement is used for all nonliteral fields defined in the DOF, not just those defined by MFLDs in the MOD. Using a fill character tailored to the device type generally improves message presentation and device performance. You can select the following fill characters on a DPAGE statement:

- Blank (X'40')

- Blank (C' ')
- Any hexadecimal EBCDIC graphic character (X'*hh*')
- An EBCDIC graphic character (C'*c*')

You can select the following characters on a MSG statement:

- Blank (C' ')
- EBCDIC graphic character (C'*c*')

For the 3270 or SLU 2 display, the EBCDIC graphic fill character fills in any fields or partial fields on the formatted display that do not receive any data or only partial data. This erases information remaining on the display from the previous message, however, using the fill character increases transmission time.

Null fill can be specified, in which case fields are not filled on the 3270 or SLU 2 formatted screen (and data from the previous message that is not updated by the current message is still displayed). For devices other than 3270 or SLU 2 display, compacted lines are produced when message data does not fill device fields. Using null fill for 3270 or SLU 2 display devices reduces transmission time, but might result in confusion if a partial field does not cover all the data remaining from a previous display. Using null fill for other devices causes additional processing in the IMS control region but reduces transmission and printing time.

For 3270 or SLU 2 formatted screen, a program tab function can be requested that erases any data remaining in a device field after new data for this field has been displayed, but does not produce any fill characters. With program tab fill, display fields on a formatted screen are not cleared unless new data is transmitted to them.

When the program sends only a few of the output data fields, the unwanted display of leftover data in unprotected fields can be prevented by specifying the “erase all unprotected” function in the system control area “System Control Area (SCA) and Default SCA (DSCA)”.

For 3270 output when EGCS fields are present, specify only FILL=PT or FILL=NULL on the DPAGE or MSG statement. Any other specification can result in the device rejecting the message.

System Control Area (SCA) and Default SCA (DSCA)

The system control area (SCA) is the means by which specific device operations are requested when an output message is sent to the device. These device requests can be defined in the message field (via the SCA) or in the device format definition (via the default SCA, or DSCA). An SCA is defined as a message field. The IMS application program can use the SCA to specify device operations to be performed when output is sent to a terminal device.

The 3270 and SLU 2 functions that can be requested are:

- Force format write.
- Erase unprotected fields before write.
- Erase all partitions before sending message.
- Sound device alarm.
- Unprotect screen for this message.
- Copy output to candidate printer.

For 3270 and SLU 2 devices, MFS interprets the IMS application program information and performs the specified operations.

A “sound device alarm” can be requested for output to an FIN workstation in the SCA; in this case, MFS in turn specifies “device alarm” in the header of the output message sent to the FIN workstation.

For an SLU P (DPM-An) or ISC subsystem (DPM-Bn), all the functions allowed for the 3270 and FIN can be specified by the IMS application program in a message field defined as an SCA. Define a device field (DFLD statement) as an SCA in the DOF. For the SLU P remote programs or ISC subsystems, MFS does not interpret the specifications from IMS. MFS only relays the specifications in the user-defined device field SCA that it sends to the remote program or ISC subsystem.

For devices other than 3270, SLU 2, FIN, SLU P, and ISC, the SCA is ignored.

For all devices that can have SCAs, a default system control area (DSCA) can also be defined in the DOF (in the DEV statement) in which the same kinds of functions can be specified. Whenever the DOF DSCA is used, the functions are performed if appropriate for the destination device. DSCA-specified functions are performed regardless of whether an SCA field is provided. If DSCA and SCA requests conflict, only the DSCA function is performed. Any invalid flag settings in the DSCA specifications are reset, and only the valid settings are used.

For SLU P remote programs, DSCA information can similarly override SCA specifications. The SCA or DSCA information is not interpreted by MFS but is transmitted to the remote program in the device field defined as an SCA.

IMS application programs that control output through specifications in the SCA can be device-dependent.

Related Reading: For additional information, see “System Control Area (SCA)” on page 278 and “DEV Statement” on page 330.

Output Message Literal Fields

Output message fields can be defined to contain literal data you specified during definition of the MOD. MFS includes the specified literal in the output message before sending the message to the device.

You can define your own literal field, select a literal from a number of literals provided by MFS, or both. The MFS-provided literals are called *system literals*, and include the following:

- Various date formats
- The time stamp
- The output message sequence number
- The logical terminal name
- The number of the logical page
- The queue number of the message waiting

Related Reading: For a description of EGCS literals, see the “DFLD Statement” on page 370. For a description of the system literals, see “MFLD Statement” on page 323.

Output Device Field Attributes

Device field attributes are defined in the DOF’s DFLD statement. For 3270 display devices, specific attributes can be defined in the ATTR= keyword or EATTR= keyword of the DFLD statement, or default attributes are assumed.

For 3270 printers, 274X and 3770 terminals, and 3601 workstations, attribute simulation can be defined by specifying ATTR=YES or ATTR=*nn* in the DFLD statement. The message field definition corresponding to the device field can specify that the application program can dynamically modify, replace, or simulate device field attributes.

Extended Field Attributes for Output Devices

Extended field attributes apply to 3270 display devices and to printers defined as 3270P or SCS1, that support the 3270 Structured Field and Attribute Processing option. These attributes also apply to 3270P or SCS1 printers that support the Extended Graphics Character Set (EGCS) if field outlining or DBCS operation is desired. These extended field attributes provide additional field attribute definition beyond that provided in the existing 3270 field attribute. They are associated with a field of characters just as the existing 3270 field attributes are, but they do not take up display positions in the characters buffer. They can define such field characteristics as:

- Color (seven-color models only)
- Highlighting
- Programmed Symbols (PS)
- Validation
- Field outlining
- Input control of mixed DBCS/EBCDIC data

Extended field attributes are defined in the EATTR= keyword of the DFLD statement. They can be dynamically modified by specifying ATTR=*nn* on the ATTR=YES or ATTR=*nn*. corresponding MFLD statement.

Any combination of existing and extended field attributes (except protect and validate) can be transmitted in one display output stream.

When dynamic attribute modification (ATTR=YES) is specified for a device field with predefined attributes, an attribute is sent to the device for that field in every output operation, even if the data for this device field is not included in the output message.

These attributes are used in the following ways:

- If the output message field has an attribute and the attribute is valid, then the dynamic attribute modification is performed.
- If the message field is not included in the LPAGE being used or the attribute is not valid, the predefined attribute for the device field is used.

The default attributes for nonliteral 3270 display device fields are:

- Alphabetic
- Not protected
- Normal display intensity
- Not modified

The default attributes for literal display device fields are:

- Numeric
- Normal display intensity

The forced attributes for literal display device fields are:

- Protected
- Not modified

Attribute simulation can be defined for non-3270 display devices but these attributes are applied only when requested by an application program. The device field definition reserves the first byte of the field for attribute data. If the application program then specifies an attribute request, that request is represented in the first byte of the device field.

Field attributes that can be simulated are:

Table 21. Simulated field attributes for literal display devices

Attribute	Action Taken
High-intensity display	An asterisk (*) is placed in the first byte
Modified field	An underscore character (_) is placed in the first byte
High-intensity and modified field	An underscore character (_) is placed in the first byte
No display	No data is sent regardless of other attributes, except for DPM

Cursor position for the 3604 can also be specified as a simulated attribute.

If a field is defined to receive simulated attribute data but none is provided by the application program, the first byte is a blank.

For an application program to modify, replace, or simulate attribute data, the message field definition must specify ATTR=YES or ATTR=*nn*. When attributes are defined this way, the first bytes of the output message field are reserved for attribute data. Any error in the specification causes the DFLD ATTR= or EATTR= specification for that attribute byte to be used, although other attribute or extended attribute specifications are processed.

For DPM devices, fields can be defined to receive attribute data, extended attribute data, or both, from the IMS application program by specifying ATTR=YES or ATTR=*nn* on the DFLD statement corresponding to the MFLD definition with ATTR=YES or ATTR=*nn*. The 3270 attributes from the IMS application program can either be converted to simulated attributes and placed in the first byte of the device field or placed unchanged (2 binary bytes as received from the IMS application program) in the first 2 bytes of the device field. The decision to send attributes, extended attributes or simulated attributes is made when the device format is defined. If a field is defined to receive attribute data but none is provided by the IMS application program, the first byte contains a blank if attribute simulation was requested, or the first 2 bytes contain binary zeros if binary attributes were requested.

Extended Graphic Character Set (EGCS)

Extended Graphic Character Sets (EGCS) extend the number of graphic characters beyond the limit available using EBCDIC. This is an extension of the programmed symbol feature. The programmed symbol is an optional feature on the IBM 3270 Information Display Station and SCS1 printers that store and use the additional character sets.

Where DBCS or DBCS/EBCDIC mixed fields are discussed in context with 3270 displays or SCS1 printer devices, it is assumed that these devices are capable of handling DBCS data. Such devices include, for example, the 5550, supported as a 3270 display, and the 5553 and 5557, supported as SCS1 printers.

Definition: The *Double Byte Character Set* (DBCS) is a subset of EGCS. In it, each graphic character is represented by 2 bytes. The valid code range is X'4040' or X'41' through X'FE' for byte 1, and X'41' through X'FE' for byte 2.

EGCS Fields: An EGCS field is defined by the EATTR= parameter on the DFLD statement for 3270 displays or SCS1 device types.

All EGCS literals are in the form G'SO XX .... XX SI', where SO (shift out)=X'0E' and SI (shift in)=X'0F'.

For SCS1 device types, EGCS is specified as a pair of control characters framing the data in the form of: G'SO XX XX XX SI'. The framing characters SO (shift out) and SI (shift in) are not actual characters, but are 1-byte codes: X'0E' or X'0F'.

EGCS literals must be specified as an even number of characters; otherwise, a warning message is issued. All characters (X'00' through X'FF') are valid in an EGCS literal; however, a warning message is issued for all characters not within the range of defined graphics, X'40' through X'FE'.

Restriction: An EGCS literal cannot be equated using the EQU statement if a hexadecimal value within the literal is an X'7D', which is equivalent to a quote character.

For the MFS Language utility to recognize an EGCS literal, observe the following restrictions when defining the EGCS literal:

- SO and SI characters cannot be defined as alphabetic characters using the ALPHA statement.
- The three characters G'SO (SO is a single character) must not span continuation lines as input to the MFS Language utility, but must appear on the same line. The same is true for the two characters SI'.

An EGCS literal can be continued on the next line. An SI character can be coded in column 70, 71, or 72 to terminate EGCS data and is not included in the literal. If an SI is in column 70, the data in column 71 is ignored, except when it is a single quotation mark. On continuation lines for literals, an SO character is not required but can be used, if it is placed in column 15. (This indicates the beginning of EGCS data and is not included in the literal).

Restriction: IMS does not support a 2-byte fill function, inbound or outbound. For outbound data, the MFS fill function is at the message level. To avoid MFS insertion of RA (Repeat to Address) orders for EGCS fields that contain no data or are omitted in the output message, FILL=PT (the default) or FILL=NULL must be specified.

The MFS Language utility uses SO and SI characters in its output listing only for the initial input statement and for error messages that display EGCS literals from the input record. EGCS literals that are a part of the device image map are displayed as a series of Gs. Additional utility output that is created by using the EXEC PARM= operands DIAGNOSTIC, COMPOSITE, and SUBSTITUTE, and that contains EGCS literals, does not have the G, SO, and SI characters inserted. Only the data between the SO and SI characters is included.

You must define the screen location (row and column) where the field is to be displayed. This includes any screen placement constraints imposed by a particular product implementation. Warning messages are issued when:

- The DFLD attribute is EGCS and the field position parameter does not specify an odd column number (3270 only)
- An EGCS literal is not specified as an even number of characters
- The DFLD length is not specified as an even number

When defining an EGCS field for a 3283 Model 52, you must ensure that the length specified is an even number and, if an EGCS field spans device lines, specify WIDTH= and POS= so that an even number of print positions are reserved on each of the device lines.

Mixed DBCS/EBCDIC Fields

The Double Byte Character Set (DBCS) is a graphic character set in which each character is represented by 2 bytes. It is a subset of the Extended Graphic Character Set (EGCS). DBCS is used in several Asian countries, such as Japan, Taiwan, and Korea, because the number of characters in those countries' written languages is more than the 256 characters that can be represented by one byte. As with EGCS, this representation is accomplished by an extension of the programmed symbol feature.

Because DBCS is a subset of EGCS, DBCS fields are specified using EGCS keywords and parameters and are treated by MFS in much the same way as EGCS data. However, DBCS data can be used in two field types, a DBCS field and a DBCS/EBCDIC mixed field. The DBCS field accepts only DBCS data and no special control characters are needed with this type of field. (The valid code range of DBCS data is X'4040', or X'41' through X'FE' for both bytes.) But, in a mixed field, where DBCS data is *mixed* with EBCDIC data, the DBCS data must be enclosed by SO (shift out) and SI (shift in) control characters.

Using DBCS requires display and printer devices capable of handling DBCS data. One such group of devices is the 5550 Family (as 3270); however, other 3270 DBCS devices are available.

Mixed DBCS and EBCDIC Fields: When DBCS data is enclosed by SO/SI characters, a mixed field on a 3270 DBCS device accepts both EBCDIC and DBCS data. Such a mixed field can contain multiple DBCS data entries enclosed by SO/SI control characters, as shown in Figure 38.

The DBCS data should always be enclosed by SO/SI control characters for both inbound and outbound data to a 3270 display. However, if the data is inbound, the control characters are automatically created by the terminal. To explicitly specify DBCS/EBCDIC mixed fields, use the keywords MIX and MIXS on the EATTR= parameter of the DFLD statement.

Example: Figure 38 shows the case of a DBCS/EBCDIC mixed field.

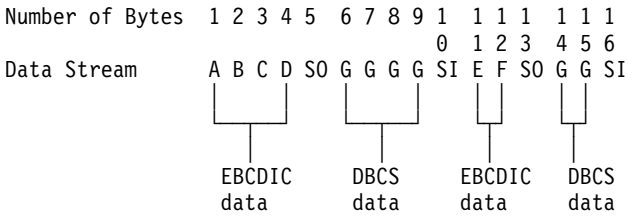


Figure 38. DBCS/EBCDIC Mixed Data

The DBCS/EBCDIC mixed data shown in Figure 38 consists of the following 16 characters:

- EBCDIC data 'ABCD' and 'EF' (6 bytes)
- DBCS data 'GGGG' and 'GG' (6 bytes)
- Two sets of SO/SI control characters (4 bytes)

The SO control character is represented by X'0E' and the SI control character is represented by X'0F'.

When DBCS is used, MFS sends the data directly to the 3270 display but performs SO/SI blank print processing before sending it to the SCS1 printer. The SO/SI control characters for 3270 displays and SCS1 printers are treated as follows:

- On 3270 displays, an SO or SI control character takes up one position on the display and appears as a blank.
- On SCS1 printers:
  - If EATTR=MIXS is specified, an SO or SI control character does not take up a position on the listing. To prevent insertion of blanks, specify EATTR=MIXS (SO/SI blank print suppress option).
  - If EATTR=MIX is specified, the SO/SI blank print option inserts a blank before an SI control character and after an SI control character in a mixed data field. Specifying MIX results in identical 3270 display output and SCS1 printer output.

The length of the mixed data containing SO/SI in the application program is different from the length of the same data on the printed output.

The length of the DBCS/EBCDIC mixed data shown in Figure 38 is 16 bytes in the application program. If the string is sent to a field specified with DFLD EATTR=MIX, the data is printed as a 16-byte string. However, if sent to a field specified as DFLD EATTR=MIXS, the data is printed as a 12-byte string (4 bytes of SO/SI control characters are suppressed). The length attributes of the DFLDs are LTH=16 and LTH=12, respectively.

SO/SI Control Character Processing: For 3270 displays, DBCS data enclosed by SO/SI control characters can be included as part of an existing EBCDIC field. When DBCS data is mixed in an existing EBCDIC field, the IMS application program must check that correct DBCS data is placed in the 3270 display field. DBCS data within an EBCDIC field is correct when the following conditions are met:

- The length of DBCS characters is an even number of bytes.
- There are no unpaired SO or SI control characters.

When MIX or MIXS is specified on the DFLD statement, MFS checks the above conditions, aligns the DBCS data enclosed by SO/SI control characters, and corrects invalid SO/SI control characters.

DBCS/EBCDIC Mixed Literals: DBCS/EBCDIC mixed literals can be specified as DFLD/MFLD literals, as shown in Figure 39.

```
    literal format:' .......SO___SI..SO__SI'

 DFLD
  'literal'

 MFLD
  ,'literal'
  ,(dlfdname,'literal')
```

Figure 39. DBCS/EBCDIC Mixed Literal

The DBCS data in a DBCS/EBCDIC mixed literal is expressed as a series of Gs in the device image map in the MFS listing.

When the MFS Language utility specifies a DFLD/MFLD literal containing DBCS/EBCDIC mixed data within an EBCDIC field without specifying EATTR=, a check for mixed field is performed for both 3270 display and SCS1 printer output. A DBCS/EBCDIC mixed field attribute with EATTR=MIX is assigned for SCS1 only. The LTH parameter is ignored even if specified. As a result, the field length is the same as the length of the literal.

Table 22 shows the processing performed by the IMS MFS Language utility for SO/SI control characters within a DBCS/EBCDIC mixed field.

Table 22. SO/SI Processing Performed by IMS MFS Language Utility

Device, Field	DFLD/MFLD Output Literal	MFLD Input Literal
3270 display, DBCS/EBCDIC mixed field	• Check SO/SI pairing. • Check even length. • Adjust boundary alignment (with warning message).	SO/SI checking not done
SCS1 printer, DBCS/EBCDIC mixed field	• Check SO/SI pairing. • Check even length. • Perform SO/SI correction and boundary adjustment according to SO/SI blank print option.	Not applicable

Table 23 shows the processing performed by the MFS message editor on SO/SI control characters within a DBCS/EBCDIC field.

Table 23. SO/SI Processing Performed by MFS Message Editor

Device, Field	Outbound Data Fields	Inbound Data Fields
3270 display, DBCS/EBCDIC mixed field	• Check SO/SI pairing. • Check even length. • Adjust boundary alignment.	SO/SI checking not done

Table 23. SO/SI Processing Performed by MFS Message Editor (continued)

Device, Field	Outbound Data Fields	Inbound Data Fields
SCS1 printer, DBCS/EBCDIC mixed field	• Check SO/SI pairing. • Check even length. • Perform SO/SI correction and boundary alignment according to SO/SI blank print option.	Not applicable

Continuation Rules for DBCS/EBCDIC Mixed Literals: The continuation rules for mixed literals are the same as the continuation rules for EGCS literals. The continuation rules are as follows:

- An EGCS literal can be continued on the next line.
- An SI character can be coded in column 70, 71, or 72 to terminate EGCS data and is not included in the literal. If an SI is in column 70, the data in column 71 is ignored, except when the character is a single quotation mark.
- On continuation lines for literals, an SO character is not required, but can be used in column 15. (This indicates the beginning of EGCS data and is not included in the literal.)

Because mixed literals have the DBCS character string, there are some considerations for their continuation:

- When data is mixed EBCDIC and DBCS, the DBCS data must be enclosed by SO and SI control characters. The SI characters can be located from column 70 to 72 in an EGCS literal; in a mixed literal, SO and SI are part of the user data. Therefore, you must fill the data up to column 71, put a non-blank character in column 72, and start the next line from column 15 (if SO) or from column 16. Examples of continuations in mixed literals are shown in Figure 40.
- When the first byte of the DBCS character is in column 71, you can put a non-blank character in column 72 and put the second byte of the DBCS character in column 16 of the next line to continue the literal. Another solution is to start the first line from column 17.

Mixed Literal

```
'abc{K1K2K3}'def where
                  abc & def  = EBCDIC characters
                  K1K2K3     = DBCS characters
                  {          = shift out X'0E'
                  }          = shift in X'0F'
```

Examples of Continuations in Mixed Literals

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
               'zzzz
```

- Within a 3270 display field or SCS1 printer field with EATRR=MIX specified, all SO control characters (except the last unpaired SO control character in the field) and all duplicate SI control characters are replaced with blanks.

  For the last unpaired SO control character in the field, an SI control character is placed in either the last, or second from the last, byte so that the length of the DBCS field is even. If an SI control character is placed in the second from the last byte, the last byte is replaced by a fill character. If an SO control character is in the last byte of a field, it is replaced with a blank.
- Within an SCS1 printer field with EATRR=MIXS specified, all SO control characters (except the last unpaired SO control character in the field) and all duplicate SI control characters are removed.

  For the last unpaired SO control character in the field, an SI control character is placed in the last, or second from the last, byte so that the length of the DBCS field is even. If the SI control character is placed in the second from the last byte, the last byte is replaced by a fill character. If an SO control character is in the last byte of a field, it is replaced with a fill character.

For SCS1 printers, all paired and unpaired SO/SI control characters exceeding the number of SO/SI pairs defined for the field are:

- Replaced with blanks, if EATTR=MIX is specified
- Removed, if EATTR=MIXS is specified

If the length of DBCS data within a DBCS/EBCDIC field is odd, the odd SI position is moved one byte to the left and the rest of the field is padded with blanks.

Input Control and DBCS/EBCDIC Mixed Field (3270 Display): When sending DBCS/EBCDIC data to a DBCS/EBCDIC field, MFS checks for SO/SI pairs and even length and performs SO/SI correction and boundary adjustment if necessary. In this way, the DBCS/EBCDIC field appears correctly on the 3270 display screen or SCS1 printer output.

When receiving DBCS/EBCDIC data from a mixed field, MFS passes the data as is. This is because SO/SI pairing and even length are always ensured when using the 3270 display.

However, when sending DBCS/EBCDIC data to a DBCS/EBCDIC field and receiving user-entered DBCS/EBCDIC data from the same field, the application program must account for changes in the data. When receiving user-entered DBCS data, the 3270 display builds the data and SO/SI control characters and then truncates or realigns the data to assure SO/SI paring and even length. The IMS application program must take this into account when using a part of the send data as receive data.

DBCS/EBCDIC Mixed Field and Horizontal Tab (SCS1 Printer): When using an online horizontal tab setting, tabs are not set within a DBCS/EBCDIC field. This is because it is not possible to determine beforehand whether the actual position of the DBCS data within a mixed field is on an odd or even boundary.

Field Outlining: This function is used for user-defined 3270 display and SCS1 printer fields.

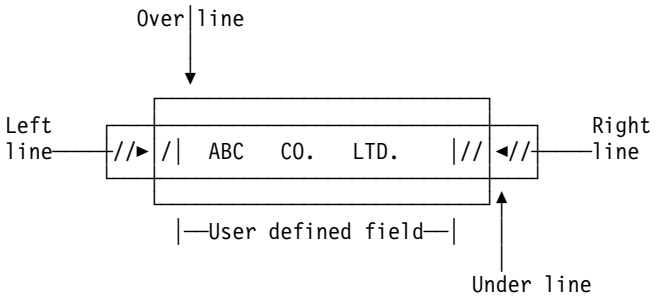


Figure 41. User Field and Field Outlining

Field outlines are referred to as OVER, UNDER, LEFT, and RIGHT lines and they can be specified independently or in any combination.

The shaded area at the left and right ends of the field shown in Figure 41 are:

- For 3270 displays, 3270 basic attribute bytes. The left attribute byte describes the first field; the right attribute byte describes the following field.
- For SCS1 printers, left and right blanks, reserved for the user-defined field by MFS.

Connecting Field Outlines and Joining Fields: You can outline multiple fields jointly as shown in Figure 42.

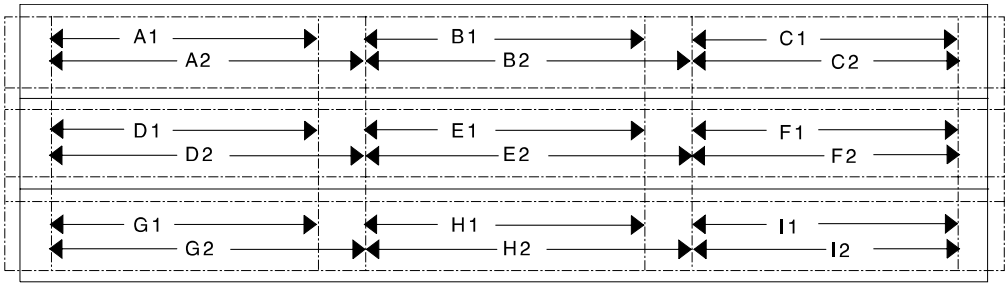


Figure 42. Field Outlining When Connecting User Fields

Figure 42 consists of nine logical fields. A1, B1, ... I1 are fields defined for the 3270 display and A2, B2, ... I2 are fields defined for the SCS1 printer. Note that for 3270 displays, 3270 basic attribute bytes are placed between fields. For SCS1 printers, the fields are connected without losing any print positions and the field outlines are connected. The outline specification for each field in Figure 42 is shown in Table 24.

Table 24. Outline Specification for Each Field

	LEFT	RIGHT	OVER	UNDER
A1, A2	X		X	
B1, B2			X	
C1, C2		X	X	
D1, D2	X		X	
E1, E2			X	
F1, F2		X	X	

Table 24. Outline Specification for Each Field (continued)

	LEFT	RIGHT	OVER	UNDER
G1, G2	X		X	X
H1, H2			X	X
I1, I2		X	X	X

You need to define only the message field for 3270 displays in your IMS application program to produce the same output on displays and printers.

When field outlining is specified for an SCS1 printer, the MFS Language utility attempts to reserve 1 byte for the left and right lines, but if adjacent fields cannot be reserved, a warning message is issued.

Cursor Positioning

On 3270, 3604, or SLU 2 display devices, the cursor is positioned by its line and column position on a physical page. When a specific cursor position is always required (and device-dependence is not an issue), you can define cursor position in the DPAGE statement.

The DPAGE statement can also be defined so that cursor position is known to the application program on input and is specified dynamically by the application program on output. To dynamically define cursor position on output, specify a device field name along with its line and column position. If this field is then referred to by a MID MFLD statement, the cursor position is provided in that message field on message input. If the message field is referred to in a MOD MFLD statement, the message field can be used by the application program to specify cursor position on output.

The application program cursor position request is used if its specified size is within the line and column specifications of the SIZE= operand of the TERMINAL macro for device type 3270-An; or within the line and column boundaries of 3270, model 1 or 2. Otherwise, the line and column positions specified on the DPAGE statement or the default positions (line 1, column 2) are used.

Related Reading: For a description of the TERMINAL macro, see *IMS/ESA Installation Volume 2: System Definition and Tailoring*.

The option of providing cursor location on input is available only for 3270 or SLU 2 devices. This method of cursor positioning is not recommended for output, because it requires the application to use a specific device field position, making the application device-dependent. MFS considers cursor position as a device field attribute; the field attribute facility can be used to establish cursor position.

Positioning the Cursor Dynamically: Application programs can dynamically replace, modify, or simulate attributes for a device field whose corresponding message field is defined as ATTR=YES or ATTR=nn. At least the first 2 bytes of a message field defined in this way are reserved for attribute data or extended attribute data provided by the application program.

For a 3290 in partitioned-format mode, the first partition descriptor (PD) statement defined in the partition descriptor block (PDB) is the first partition created. The cursor is placed in this partition, which becomes the active partition unless overridden by the Jump Partition key or by the ACTVPID= keyword in the DPAGE statement associated with a subsequent output message.

Using the Jump Partition key causes the cursor to jump to the next sequential partition defined by the application program and that partition becomes the active one. The ACTVPID= keyword allows the application program to activate and locate the cursor in a specific partition.

Prompt Facility

The prompt facility provides a way to automatically notify you if the current page of output is the last page of the message. The notification text is defined as a literal which MFS inserts into a specified device field when it formats the last logical page of the message. To further assist you, the prompting text can be used to tell you what input is expected next.

Recommendation: For a 3270 or SLU 2 device, the combination of PROMPT and FILL=NULL should be used with care because, once the prompt literal is displayed, it can remain on the screen if your input does not cause reformatting of the screen.

System Message Field (3270 or SLU 2 Display Devices)

Output formats for 3270 or SLU 2 display devices can be defined to include a system message field. If defined in this way, all IMS messages except REQUESTED FORMAT BLOCK NOT AVAILABLE are sent to the system message field whenever the device is in formatted mode. Using a system message field or setting byte 1 bit 5 to B'0' in the DSCA specification prevents an IMS message from destroying a screen format.

When MFS sends a message to the system message field, it activates the device alarm (if any) but does not reset modified data tags (MDTs), move the cursor, or change the protect/unprotect status of the display, except in the event of a multi-segment message. In this case, the status is changed to protected, and the enter key must be pressed to view the next segment or segments of the message. Because IMS error messages are an immediate response to MDTs in input, MDTs remain as they were at entry and you must correct the portion of the input that was in error.

After input from an operator identification (OID) card reader, the device is no longer in formatted mode. Therefore, an IMS message is not sent to a SYSMSG field; it is sent using the default system message format. This is also the case after an XRF takeover because the device is no longer in formatted mode.

Printed Page Format Control

The PAGE= keyword of the DEV statement provides much of the formatting control of the format of output messages sent to printer devices.

The WIDTH= keyword provides additional formatting control. In conjunction with the FEAT=(1...10) keyword, WIDTH= provides additional formatting control for printer devices specified as 3270P. (See WIDTH= under the DEV statement for additional information.) The WIDTH= keyword, in conjunction with the HTAB=, VTAB=, VT=, SLDI= and SLDP= keywords, provides additional formatting control for 3770, SLU 1, or SLU 4 printer devices.

Using a PAGE= operand (DEFN, SPACE, FLOAT, or EJECT), with the page depth (the number of lines per page), determines how MFS controls the printing of the output message. The PAGE= operands are described below.

DEFN MFS prints each line as defined by DFLD statements. In this mode, if the first DFLD defined line is greater than 1, the printer position is moved to the first defined line. The printer position is also moved over the blank lines between defined DFLDs. However, MFS does

not add blank lines to the bottom of the page of output if the last defined line is less than the page depth. The next page of output begins on the line following the current line of output. The number specified in the PAGE= keyword is used to check the validity of the line specification of the DFLD POS= keyword.

SPACE This produces the same printing mode as DEFN except that lines are added to the bottom of the page if the last defined line is less than the page depth. The printer is positioned through a series of new lines. This option can be used for devices that do not have the page eject feature so that pages are not grouped together.

FLOAT This operand is used to request that lines not be printed if they are defined by DFLD statements, or if they contain no data after formatting (all blank or NULL).

EJECT This operand is specified for FIN, 3770, SLU 1, or SLU 4 printers. The following options can be specified for EJECT (or any combination of these):

BGNPP or ENDPP	MFS ejects the page before (BGNPP) or after (ENDPP) each physical page of the output message.
BGNMSG	MFS ejects the page before any data in the output message is printed.
ENDMSG	MFS ejects the page after all the data in the output message is printed.

MFS does not add lines to or delete lines from the page. EJECT can be specified for FIN, 3770, SLU 1, or SLU 4 printers.

Format Control for 3770, SLU 1, and SLU 4 Printers

MFS provides several specifications to control the format of output messages to 3770 printer devices and SLU 1 (print data set) (DEV TYPE=SCS1) or SLU 4 (DEV TYPE=SCS1). Printer formatting features are listed and described below.

Print Mode: The section, “Printed Page Format Control” on page 221, describes print mode for 3770, SLU 1 or SLU 4 printers.

Page Depth: The page depth, as specified in the PAGE= keyword, is discussed in the section “Printed Page Format Control” on page 221.

Line Width: The WIDTH= keyword of the DEV statement is used to specify the maximum width of a print line, relative to column 1. The specified width is used in place of the physical device line width. Specification of a line width also establishes the right margin of the printed page (relative to column 1). Valid values are less than or equal to the physical device line width. For example, if WIDTH=80 is specified, data can be printed in columns 1 through 80.

Left Margin Position: The left margin operand of the HTAB= keyword of the DEV statement can be used to specify where MFS should set the left margin for the device before sending an output message. A left margin specification should be made if output fields always start at a column position other than column 1 (the default). For example, if fields are always defined in columns 5 through 80, HTAB=(5) and WIDTH=80 can be specified on the DEV statement.

Horizontal Tabbing: The HTAB= keyword of the DEV statement is used to specify where MFS should set horizontal tab stops before sending an output message.

MFS can insert tab control characters into the message to reduce the number of characters transmitted. To control when tab control characters are inserted, specify the ONLINE or OFFLINE operand for the HTAB= keyword. OFFLINE specifies that MFS insert the tab control characters during compilation of the control blocks by the offline MFS Language utility program. ONLINE specifies that MFS insert the control characters during online processing of the message. MFS can only be directed to insert tab control characters into messages that have legitimate fill characters specified (FILL=X'hh' or FILL=C'c' in the DPAGE statement), or use the default fill character, X'40'.

Specify OFFLINE when the message definition always supplies data to most defined device fields, or the fill character is not a blank. Specify ONLINE if some device fields do not receive data, or the data contains blanks. Even though the ONLINE specification increases MFS online processing, it reduces character transmission to the device.

Vertical Tabbing: The VT= keyword of the DEV statement is used to specify where MFS should insert vertical tab control characters into the page of the output message. MFS assumes that the vertical tab stops are relative to line 1 and have been set at the device by the specification of the VTAB= keyword or other means prior to message transmission. VT= must be specified if vertical tabbing is required. There are no default values. VT= is invalid if page control specifications direct MFS to delete lines that contain no data after formatting. EJECT BGNMSG or EJECT BGNPP should be specified in conjunction with the VT= keyword to ensure proper alignment at the beginning of a page. A specification of VT= without a suitable EJECT operation defined can result in invalid device formatting.

Top and Bottom Margins: Top and bottom margins can be specified for printers specified as DEV TYPE=SCS1 by using the VTAB= keyword on the DEV statement. VTAB= is invalid if page control specifications (PAGE=n,FLOAT) direct MFS to delete lines that contain no data after formatting.

When used together, the page depth (PAGE=), vertical tab (VT=), and top and bottom margin (VTAB=) specify a “set vertical format” data stream.

Line Density: For printers specified as DEV TYPE=SCS1, the density of lines on an output page can be specified with the SLDx= keyword on the DEV statement, the DFLD statement, or both. Line density can be set in terms of lines per inch or points per inch. If SLDx= is specified on both the DEV and DFLD statements, two SLD data streams are sent, one at the beginning of a message and one within the message, just before the field on which the SLDx specification, was encountered, but after any vertical tabs and new line characters. The SLDx specification within the message changes the line density from that set at the beginning of the message to that specified within the message. The line density specified within the message remains in effect until explicitly reset.

Output Format Control for 3270P Printers

MFS provides several specifications to control the format of messages to 3270P printer devices.

Print Mode: “Printed Page Format Control” on page 221 describes print mode for 3270P printers.

Page Depth: The page depth, as specified in the PAGE= keyword, is discussed in "Printed Page Format Control" on page 221.

Line Width: The WIDTH= keyword of the DEV statement is used to specify the maximum width of a print line relative to column 1. The specified width is used in place of the physical device line width. The default for 3270P printers is 120. When WIDTH= is specified, a feature code from 1 to 10 must also be specified via the FEAT= keyword on the DEV statement.

Output Format Control for SLU P DPM-An

For SLU P devices with the DPM-An option, You can use several specifications in MFS to control the format of output messages.

The RCDCTL= operand of the DIV and RCD statements identifies a related group of device field (DFLD) definitions that are within one record, which is usually sent to a remote program as one transmission (that is, if the RCDCTL= value is less than or equal to the value in the OUTBUF= parameter of the system definition TERMINAL macro).

The number of device fields in the record is determined by the length (numeric value) specified in RCDCTL. Device fields can be arranged in records through the RCD statements. The records created can be smaller than the size specified in RCDCTL. The SPAN/NOSPAN parameter determines whether fields are allowed to span record boundaries. All output messages are sent in record mode.

The PPAGE statement identifies a presentation page of a device format and can contain one or more records.

The DPAGE statement defines a logical page of a device format and can contain one or more records.

Paging: The MSG, DPAGE, or PPAGE operands of the OPTIONS= specification of the DIV statement is used to determine how the output message is sent to the remote program.

MSG This specifies that all the data in the output message is to be transmitted together to the remote program in one chain. This is the default.

After transmitting the message to the remote program, IMS does not transmit another output message if PROGRAM2 has been specified as the media parameter of the COMPTn operand of the system definition TERMINAL macro. An input request is required from the remote program before the next message is sent. If PROGRAM1 is specified, IMS does not wait for an input request, but sends another output message if one is available.

DPAGE This specifies that all the data in the logical page is to be transmitted together to the remote program in one chain. A paging request is required from the remote program to retrieve the next logical page of the output message.

PPAGE This specifies that all the data in the presentation page is to be transmitted together to the remote program in one chain. A paging request is required from the remote program to retrieve the next presentation page of the output message.

A paging request can be specified through the input message header or through an operator control table. For OPTIONS=DPAGE or PPAGE, when the last logical or presentation page has been sent to the remote program, IMS MFS action is the same as for 3270 and 3604 devices (shown in Table 18 on page 205) regardless of PROGRAM1 or PROGRAM2 specification.

Each chain contains an output message header. The DATANAME in the output message header is the format name if OPTIONS=MSG is specified, the current name of the device logical page (DPAGE) if OPTIONS=DPAGE is specified, or the current name of the presentation page if OPTIONS=PPAGE is specified.

The output message header is always present in the first transmission record of the chain. For OPTIONS=MSG, the first transmission record contains only the output message header, and the next transmission begins the data for the message.

For OPTIONS=DPAGE or PPAGE, the data follows the output message header in the first transmission record if either of the following occurs:

- RCDCTL=(,SPAN) is specified, and the RCDCTL length is greater than the output message header length.
- RCDCTL=(,NOSPAN) is specified, the RCDCTL length is greater than the output message header length, and at least the first data field defined in the current DPAGE or PPAGE can be fully contained within the first transmission record.

Output Message Header: The basic output message header contains the following MFS fields, presented in this sequence:

VERSION ID

MIDNAME

DATANAME

DATANAME is the FMT label for OPTIONS=MSG, the DPAGE label for OPTIONS=DPAGE, and the PPAGE label for OPTIONS=PPAGE.

If a forms literal is specified in the DEV statement, the FORMSNAME field is present in the output message header. For OPTIONS=MSG the FORMSNAME is present in the basic header after the DATANAME. For OPTIONS=DPAGE OR PPAGE, an optional forms output message header precedes the basic output message header. It contains the following fields:

MIDNAME

FORMSNAME

The forms header is sent to the remote program as the only element of a chain. A paging request is required after the header has been processed and the remote program is ready to process the first logical or presentation page of an output message.

The length of the output message header can be defined in the HDRCTL= operand of the DIV statement as fixed or variable.

The length of the fixed basic output message header (without FORMSNAME) is 23 bytes for OPTIONS=MSG and 25 bytes for OPTIONS=DPAGE or PPAGE. If FORMSNAME is present, the maximum length of the basic output message header for OPTIONS=MSG is 40 bytes, and the maximum length for OPTIONS=DPAGE or PPAGE is 33 bytes.

- If HDRCTL=FIXED is specified, the MIDNAME and DATANAME fields are always padded with blanks to the maximum definable length: MIDNAME to 8 bytes (if

MIDNAME is not supplied, 8 blanks are presented), FMT name to 6 bytes, and DPAGE or PPAGE name to 8 bytes. For this reason, the position of the DATANAME is always at the same displacement in the basic output message header, and the FORMSNAME, if present, is always at the same displacement, following the FMT name if OPTIONS=MSG and following the MIDNAME if OPTIONS=DPAGE or PPAGE.

- If HDRCTL=VARIABLE is specified, neither MIDNAME nor DATANAME is padded. If MIDNAME is less than 8 bytes or is not present, the position of the DATANAME, FORMSNAME, or both within the output message header is variable.

Figure 43 shows the format of the fixed output message header for OPTIONS=MSG.

FIELD	BASE	L1	MIDNAME	L2	DATANAME	L3	FORMSNAME
BYTES	7	1	8	1	6	1	(user–coded literal}

Figure 43. Fixed Output Message Header Format for OPTIONS=MSG

BASE The base DPM-An output header with a length of 7 bytes, including the version ID.

L1 The full length of the MIDNAME plus 1. Contains the value 9.

MIDNAME Contains the MIDNAME to be used for input. If this name is less than 8 characters, it is padded with blanks to a full 8 bytes. If the MIDNAME is not specified, this field contains 8 blanks.

L2 The full length of the format name (DATANAME) plus 1. Contains the value 7.

DATANAME The name of the format that was used to format the data fields. If the format name specified is less than 6 characters, it is padded to a full 6 bytes.

L3 Contains the length of the forms literal plus 1. The maximum value is 17.

FORMSNAME Contains the literal specified in the FORS= parameter of the DEV statement. It can have a length of 1-16 bytes. If FORS= is not specified in the DEV statement, the L3 and FORMSNAME fields are not included in the output message header.

If a variable output message header is specified in the HDRCTL= operand of the DIV statement, the output message header for OPTIONS=MSG will have the same format, but MIDNAME and DATANAME will have trailing blanks omitted and their length fields adjusted accordingly. If MIDNAME is not used, neither the MIDNAME field nor its length is present.

Figure 44 shows the format of the fixed basic output message header (without FORMSNAME) for OPTIONS=DPAGE or PPAGE.

FIELD	BASE	L1	MIDNAME	L2	DATANAME
BYTES	7	1	8	1	8

Figure 44. Fixed Basic Output Message Header (Without FORMSNAME) for OPTIONS=DPAGE or PPAGE

BASE Content is the same as for OPTIONS=MSG (Figure 43 on page 226).

L1 Content is the same as for OPTIONS=MSG (Figure 43 on page 226).

MIDNAME Content is the same as for OPTIONS=MSG (Figure 43 on page 226).

L2 This is the full length of the DPAGE or PPAGE name (DATANAME plus 1). Contains the value 9.

DATANAME Contains the name of the DPAGE or PPAGE that was used to format the data fields for the current logical or presentation page. If the DPAGE or PPAGE name specified is less than 8 characters, it is padded with blanks to the full 8 bytes.

Figure 45 shows the format of the optional forms output message header for OPTIONS=DPAGE or PPAGE.

FIELD	BASE	L1	MIDNAME	L2	FORMSNAME
BYTES	5	1	8	1	(user—coded literal)

Figure 45. Optional Forms Output Message Header for OPTIONS=DPAGE or PPAGE

BASE The base of the optional forms output message header does not include a version ID.

L1 Contains the value 9.

MIDNAME Content is the same as for OPTIONS=MSG (Figure 43 on page 226).

L3 Contains the length of the coded forms literal plus 1.

FORMSNAME Contains a user-coded literal, as in the fixed output message header for OPTIONS=MSG. (See Figure 43 on page 226.)

Naming Conventions: Establish naming conventions for formats, device logical pages, and presentation pages (that is, for the labels of the FMT, DPAGE, and PPAGE statements). For example, you can establish conventions for FMT, DPAGE, and PPAGE names that allow the remote program to interpret them in terms of 3790 panels or functional program subroutines. Also standardize DPM-An output message headers.

User-written labels for PPAGE statements must be unique within a format definition. It is recommended that labels also be unique within the IMS system.

If OPTIONS=PPAGE has been selected for a format definition, the PPAGE label is sent as the DATANAME in the output message header. The label should give the remote program information that can be used in deciding how to process the data. When you have not coded a label for a PPAGE, MFS generates a label for it and sends this generated name in the output message header. The MFS-generated names can be used by the remote program, but leaving the label specification up to MFS is not recommended, because the generated name for a given PPAGE can change every time the MFS definitions are recompiled.

Deletion of Null Characters in DPM Output Records: See the discussion of FILL=NULL in the DPAGE statement in "Chapter 10. MFS Language Utility" on page 311 for a discussion of deletion of null characters in transmission records.

Output Format Control for ISC (DPM-Bn) Subsystems

This section describes the major output message formatting functions of MFS with ISC nodes.

Format Control

For ISC nodes, MFS allows several specifications to control the format of output messages. If OPTIONS=DPAGE or OPTIONS=PPAGE is specified on the DIV statement, MFS sends an output message in multiple logical or presentation pages. Transmission of these pages within the message occurs on demand or automatically when you set byte 1 bit 5 of the system control area (SCA). For details, see "System Control Area (SCA)" on page 278.

Function Management (FM) Headers

FM headers are headers on output messages that control functions such as paging.

Paged Output Messages

For DPM-Bn paging support, if OPTIONS=DPAGE or OPTIONS=PPAGE is specified on the DIV statement, MFS sends an output message in multiple logical or presentation pages.

Demand Paging

With demand paging, the logical or presentation pages are sent only when a paging request is received from the other subsystem. The initial output for the message contains only the ATTACH FM header. If DIV OPTIONS=DNM is specified, the data structure name (DSN) is also transmitted.

Autopaged Output

This option is available message-by-message, based on SCA values. With this facility, the logical or presentation pages are sent immediately, in multiple transmission chains (one transmission chain per page). With this option, the receiver obtains an entire output message in multiple transmission chains. Each transmission chain contains the DSN, if required.

Restriction: Paging requests cannot be entered to control receipt of the message.

If no data exists for variable-length fields of a page within the message, a null data chain can result.

Byte 1 bit 5 in the DSCA= operand of the DEV statement or in the SCA option of the MFLD statement indicates autopaged output.

If PAGE=YES is specified in the corresponding MSG definition and autopaged output is requested, the PAGE=YES specification (operator logical paging) function is reset and the output message is dequeued at the end of the message. Operator logical paging applies only to MFS demand paged output.

Output Modes

For output from IMS, the ATTACH manager provides for two blocking algorithms: variable length, variable blocked (VLVB) records and chained Request/Response Unit (RUs, MFS stream mode). Each record presented by MFS to the ATTACH manager is preceded by a length field when sent to the other subsystem. The length field contains the size of the record presented by MFS. The record itself is sent in as many RUs as required. Fields span RU boundaries but do not span record boundaries. The number of VLVB records in the transmission chain and the maximum size of the MFS record depend on the output mode selected and the paging option specified.

In stream mode, the way DFLDs are defined depends on the OPTIONS= keyword used:

- For OPTIONS=MSG (paging is not defined), DFLDs are defined in a DPAGE.
- For OPTIONS=DPAGE (paging is defined), DFLDs are defined in a DPAGE.
- For OPTIONS=PPAGE (paging is defined), DFLDs are defined in a PPAGE.

For all three OPTIONS= keyword settings, All the DFLDs defined in a DPAGE (or PPAGE) are grouped into a single MFS record for transmission, and all the data in one DPAGE (or PPAGE) is equal to one MFS record and equal to one output RU chain. One or more RUs are sent in the single transmission chain of the output message.

If the OFTAB parameter of a DIV or DPAGE statement is defined, contiguous output field tab separator characters are removed and are not sent to the subsystem in the following cases:

- At end of message for OPTIONS=MSG
- At end of DPAGE for OPTIONS=DPAGE
- At end of PPAGE for OPTIONS=PPAGE

In record mode, the DFLDs defined in a DPAGE or PPAGE are grouped into smaller records for transmission. The RCDCTL parameter of the DIV statement is used to define the maximum length of the MFS record created. If the RCDCTL= parameter is not specified, the default value allows for records of up to 256 bytes in length. The RCD statement is used to start a DFLD on a new record boundary.

If the OFTAB parameter is defined, contiguous output field tab separator characters at the end of the record (for omitted fields and possible short last data field) are removed before transmission. If the entire record is thus eliminated and additional data records follow, a 1-byte record containing the single output field tab separator character is sent. The record is eliminated in the following cases:

- At end of message for OPTIONS=MSG
- At end of DPAGE for OPTIONS=DPAGE
- At end of PPAGE for OPTIONS=PPAGE

One or more VLVB records are sent in a single transmission chain of the output message (OPTIONS=MSG) or the page (OPTIONS=DPAGE or PPAGE).

Variable-Length Output Data Stream

The output field tab separator character (OFTAB) provides an alternative to fixed-length field output and reduces the number of bytes transmitted over the communication lines when only graphic data is sent.

Output Field Tab Separator Character

If the length of the data supplied by an IMS application is less than the length defined for the corresponding DFLD, or if there is no data for the field, you can direct MFS to insert field tab separators to delimit output fields. You can also direct MFS to insert field tab separators for all output fields, regardless of their data length. To do this, specify the output field tab separator character (OFTAB operand). If OFTAB is used, output fields are not padded to their defined lengths.

The following definition is provided on the DIV and DPAGE statements:

```
,OFTAB=( X'hh', MIX  )
         C'c'   ALL
```

Follow these rules when you specify an OFTAB operand:

1. For OPTIONS=MSG, specify the OFTAB operand on the DIV statement only. If you specify the OFTAB operand on the DPAGE statement it is ignored.
2. For OPTIONS=DPAGE and OPTIONS=PPAGE, specify the OFTAB operand on the DIV statement, the DPAGE statement, or both. If you specify the OFTAB operand on the DIV statement, the output field tab separator character specified is used as a default output field tab separator specification for each field of the entire output message. If you also specify the OFTAB operand on a DPAGE statement, the output field tab separator character specification on the DPAGE is used for the DPAGE being described.
3. The output field tab separator character cannot be defined as X'3F' or as a blank (X'40' or C' ').

Additionally, the following guidelines apply when you specify OFTAB.

- The output field tab separator specification overrides any FILL=NULL specification or default on the DPAGE or MSG statement. The MFS Language utility issues a warning diagnostic if the FILL= operand is specified on the DPAGE statement and the OFTAB= parameter is present on the DIV or the DPAGE statement.
- The user-defined output field tab separator character cannot be present in the data from the IMS application program. If it is, MFS changes it to a blank (X'40').
- Any JUST=R (right-justify) specification on the MFLD statement for an output message that uses the output field tab separator is ignored and the JUST=L (left-justify) specification is assumed.
- If GRAPHIC=YES is specified on the SEG statement that maps to a DPAGE where the OFTAB specification applies, the output field tab separator should be a nongraphic character (X'FF', or X'00' through X'3E'), instead of an EBCDIC graphic character (X'40' through X'FE'), because EBCDIC characters can be present in the data from the IMS application program.
- If GRAPHIC=NO is specified in the SEG statement, an output field tab separator specification can produce undesirable results. However, MFS does not restrict the use of nongraphic data with the output field tab separator. If GRAPHIC=NO is specified on the SEG statement that maps to a DPAGE where the OFTAB specification applies, the output field tab separator character must be a unique character that is not present in your data. Additionally, if X'3F' is present in your data, it is compressed. Carefully examine your applications before you choose

the above combination, because this function effectively prohibits sending binary or packed decimal data from the application program.

- If MIX is specified (or the default used), the output field tab separator character is inserted only if the data length is less than the DFLD defined length.
- If ALL is specified, the output field tab separator character is inserted after every DFLD.
- If MODE=RECORD is specified, contiguous output field tab separator characters at the end of a record are removed. Records with no data at the end of DPAGE or PPAGE are not sent. Otherwise, a 1-byte record containing the output field tab separator character is sent.

For OPTIONS=DPAGE and OPTIONS=PPAGE, the OFTAB specification on the DPAGE statement (instead of on the DIV statement) allows the following:

- Mixing of fixed-length fields and variable-length fields in one output message. With proper design, this function allows all graphic segments to be mapped to a DPAGE with an OFTAB specification to produce a transmission chain of variable-length fields. This function also allows any nongraphic segments to be mapped to a DPAGE without an OFTAB specification to produce a transmission chain of fixed-length fields.
- A different output field tab separator character to be used for each DPAGE.

For OPTIONS=MSG, the OFTAB specification on the DIV statement imposes the following restrictions:

- If the OFTAB= specification is used, fields in the entire message are treated as variable-length fields.
- The output field tab separator character cannot be present in the entire output message from the IMS application program. Therefore, output field tab separator characters should not be specified if nongraphic data is being sent.

FILL=NULL Specification

Specify FILL=NULL on the DPAGE or MSG statement and specify the OFTAB= parameter in the DIV or DPAGE statement to preserve field separation. If FILL=NULL is specified on the DPAGE or MSG statement and the OFTAB= parameter is not present on the DIV statement or the DPAGE statement, a compressed output data stream is produced and field separation is not evident.

Use FILL=NULL for graphic data. If GRAPHIC=NO and FILL=NULL are specified in the SEG statement, any X'3F' in the non-graphic data stream is compressed out of the segment and undesirable results can be produced. Send non-graphic data on output as fixed length output fields and do not specify FILL=NULL.

Output message segments and message fields defined for each segment are processed sequentially by MFS if option 1 or 2 is defined in the OPT= operand of the MSG statement. Message fields in option 1 and 2 segments are defined as fixed-length fields and in fixed position. The data for these fields can be supplied as fixed-length fields, or it can be shortened by the application program. The data can be shortened by two methods:

- By inserting a short segment if no data exists for fields defined at the end of a segment.
- By placing a null character (X'3F') in the field. MFS scans segment data left to right for a null character. The first null character encountered terminates the data for a corresponding MFLD. Positioning of all fields in the segment remains the same as the positioning of defined fields regardless of null characters.

Trailing Blank Compression

Blanks at the end of segments are compressed if all of the following are true:

- OFTAB= is specified on the DIV or DPAGE statement, or if FILL=NULL or FILL=PT.
- GRAPHIC=YES is specified for the segment.
- OPT=1 or OPT=2 is specified in the MSG statement.

Specifying COMPR

You can specify trailing blank compression (COMPR=) as FIXED, SHORT, or ALL.

FIXED: If COMPR=FIXED is specified, MFS removes trailing blanks from fixed-length data fields. The resulting mapping in the DFLD is as if the application program inserted a short data field (by inserting X'3F' in the position after significant data or by inserting a short segment) or omitted the field (by inserting X'3F' in the first position of the field or by inserting a short segment) if the entire field contains blanks.

Fields shortened by an application program are not compressed in the same way as when COMPR=FIXED is specified. This option is provided for application programs that always supply maximum-length fields (such as the NAME field) for simplicity of the application program, and these blanks are not significant to the receiver. The receiver can assume that fields shortened or omitted by the compress option or by the application program have the same meaning.

SHORT: If COMPR=SHORT is specified, MFS removes trailing blanks from the data fields shortened by the application program. The resulting mapping in the DFLD is as if the application program inserted a short field with no trailing blanks or omitted the field. Fixed-length fields do not undergo this compression.

This option is provided for application programs written for the 3270 and without application program changes.

ALL: If COMPR=ALL is specified, the trailing blanks in the fixed-length and short fields are removed.

Trailing blanks in a short field or a single blank short field causes a specific operation on the 3270 (that is, to clear the entire field on the screen for a single blank and insert a program tab character (FILL=PT), or to clear the remaining portion of the updated field and insert one or more null characters (FILL=NULL)).

Saving Line Transmission Time

Line transmission time can be saved by using one of the following methods:

- Specifying COMPR=ALL, which removes the trailing blanks in fixed-length and short fields
- Defining record mode, and defining the fields as occurring at the end of the record

Blank Compression on Variable-Length Output

Examples of variable-length output with blank compression are shown in Figure 47 and in Figure 48 on page 235.

Figure 46 shows the data entered by the IMS application.

Segment 1:

```
DLZZ  FIELD A1   | FIELD A2 |FIELD A3 |FIELD A4 |FIELDC1|FIELD C2
0200  AAAAA44444|1234563...|43.......|A4A4A4
0800       00000|      F   |0F
```

Segment 2:

```
DLZZ  FIELD B1   | FIELD B2 |FIELD D1 |FIELD D2 |FIELD D3|FIELD E1
0300  BBBBBBBBBB|4444444444|DDDDDD43.|3........|D3D3D3D3
0400             |0000000000|      0F |F        |
```

Note: Both segments entered are shortened by the program.

Figure 46. Data Entered by the IMS Application

Table 25 shows the MFS definitions used in Figure 46.

Table 25. MFS Definitions for Data Entered by IMS Application

MSGOUT	MSG	TYPE=OUTPUT, SOR=FMTOUT
	SEG	
	MFLD	A1,LTH=10
	MFLD	A2,LTH=10
	MFLD	A3,LTH=10
	MFLD	A4,LTH=10
	MFLD	C1,LTH=10
	MFLD	C2,LTH=10
	SEG	
	MFLD	B1,LTH=10
	MFLD	B2,LTH=10
	MFLD	D1,LTH=10
	MFLD	D2,LTH=10
	MFLD	D3,LTH=10
	MFLD	E1,LTH=10
	MSGEND	
FMTOUT	FMT	

Figure 47 shows how blank compression and mapping occurs in record mode.

```
VLVB    FIELD A1  THRU A4: (First record)
01      AAAAA,123456,,A4A4A4
06
VLVB    FIELD B1:          (Second record)
00      BBBBBBBBBB
0C
VLVB    NO DATA:           (Third record)
00
03
VLVB    FIELDS D1 and D3:  (Fourth record)
01        DDDDDD,,D3D3D3D3
02
```

Notes:

1. Field A2 was short.
2. Field A3 had no data.
3. Field A4 was short. Trailing separators in a record are not transmitted.
4. Field B2 had no data.
5. Fields C1 and C2 had no data. A 1-byte record is transmitted because more data follows.
6. Field D1 was short.
7. Field D2 had no data.
8. Field E1 had no data. A record is not transmitted because no more data follows.

Figure 47. Variable-Length Output with Blank Compression in Record Mode

Table 26 shows the MFS definitions used for record mode output as shown in Figure 47.

Table 26. MFS Definitions for Record Mode

	DEV	TYPE=DPM-B1, FEAT=5, MODE=RECORD	
	DIV	TYPE=OUTPUT, OFTAB=(c',',MIX), COMPR=ALL	X
A1	DFLD	LTH=10	
A2	DFLD	LTH=10	
A3	DFLD	LTH=10	
A4	DFLD	LTH=10	
	RCD		
B1	DFLD	LTH=10	
B2	DFLD	LTH=10	
	RCD		
C1	DFLD	LTH=10	
C2	DFLD	LTH=10	
	RCD		
D1	DFLD	LTH=10	
D2	DFLD	LTH=10	
D3	DFLD	LTH=10	
	RCD		
E1	DFLD	LTH=10	

Figure 48 shows how compression and mapping occur in stream mode.

```
VLVB    FIELDS A1 THROUGH D3:  (Single record)
03      AAAAA,123456,,A4A4A4,BBBBBBBBBB,,,DDDDDD,,D3D3D3D3
```

Note: In stream mode, a separator is not transmitted for field D3, which is short, and for field E1, which is omitted.

Figure 48. Variable-Length Output with Blank Compression in Stream mode

Table 27 shows the MFS definitions used for stream mode output as shown in Figure 48.

Table 27. MFS Definitions for Stream Mode

	DEV	TYPE=DPM-B1, FEAT=6, MODE=STREAM	
	DIV	TYPE=OUTPUT, OFTAB=(c',',MIX), COMPR=ALL	X
A1	DFLD	LTH=10	
A2	DFLD	LTH=10	
A3	DFLD	LTH=10	
A4	DFLD	LTH=10	
B1	DFLD	LTH=10	
B2	DFLD	LTH=10	
C1	DFLD	LTH=10	
C2	DFLD	LTH=10	
D1	DFLD	LTH=10	
D2	DFLD	LTH=10	
D3	DFLD	LTH=10	
E1	DFLD	LTH=10	
	FMTEND		

Data Structure Name

The data structure name is sent in a separate DD header unless you code OPTIONS=NODNM on the DIV statement. If you code OPTIONS=DNM or the default is used, the DD header is present in each transmission chain of an output message, or each transmission chain of a demand paged output message.

In addition to the data structure name parameter in the DD header, the version identification parameter is present in the only transmission chain of an output message or in the first transmission chain of paged output messages.

Version Identification

You have an option of coding a 2-byte value on the DEV statement to be included in the DOF or DIF control block as the version ID. If this parameter is not coded, the version ID is generated by MFS using a hashing algorithm on the date and time. The value is also printed in the MFS Language utility output so that you can reference it in format definitions in remote programs.

Your Control of MFS

This section describes the MFS facilities that can assist you, or allow a remote program to control the display or transmission of output messages. This section also describes paging action at the device, the unprotected screen option, and your control when using the 3290 Information Panel in partitioned format mode.

Operator Logical Paging

Operator logical paging allows you (or, for SLU P, a remote program, or ISC subsystems) to request a specific logical page of an output message. It is defined on a message basis in the PAGE= operand of the MOD's MSG statement.

Functions Provided

When a MOD is defined to allow operator logical paging, the following functions are available to you once the first physical page of the output message is displayed:

- Enter = to display the next logical page of the current message.
- Enter *=n*, *=nn*, *=nnn, or =nnnn* (where *n* is the logical page number) to display a specific logical page of the current message.
- Enter *=+n*, *=+nn*, *=+nnn, or +nnnn* to display the *n* th logical page past the current logical page.
- Enter *=−n*, *=−nn*, *=−nnn, or =-nnnn* to display the *n* th logical page before the current logical page.
- Enter =L to display the first physical page of the last logical page of the current message.

Format Design Considerations

When operator logical paging is permitted, message and device formats should be designed to allow you to enter the page request onto a currently displayed page and have the request edited to the first field of the first input segment. If this is not done, or the PAGEREQ function is not used (see "Operator Logical Paging"), paging requests can only be entered on a cleared device.

Preferably, the installation standard for device formats should include a specific device field for you to enter logical page requests, transaction codes, and IMS commands. If the transaction code is normally provided through a message or program function key literal, the PAGEREQ function can be used, or a field can be defined at the beginning of the first segment using the null pad character. A page request field on the device can map to this field. If you do not enter a page request, the null pad causes the field to be removed from the segment and the second field (literal transaction code) appears at the beginning of the segment.

Transaction Codes and Logical Page Requests

If the PAGEREQ function is not used to specify a page request, MFS formats input data according to the defined MID prior to determining whether operator logical paging was specified, and whether the input contained a page request. If operator logical paging was not specified, the message undergoes standard IMS destination determination.

If operator logical paging was specified, MFS examines the first data of the first message segment (first field if the message uses format option 3) for an equals sign (=). If MFS does not find an equals sign, it routes the message to its destination. If an equals sign is present, all following characters up to a maximum of 4, or the first blank, are considered to be a page request.

A message destined for a single-segment command or transaction, as required in Fast Path applications, should be defined as single-segment in its MID. If the MID defines more than one segment, you must ensure that only one segment is created when the destination is a single-segment command or transaction. This can be achieved by careful input and the use of option 2, null compression (FILL=NULL) or both.

Operator Control Tables

Input device fields can be defined to invoke MFS control functions when either the data or the data length satisfies a predefined condition. Do this by defining one or more operator control tables and including the related table name in the device field definition. When a device field is defined with an associated operator control table, MFS processes the device input field and performs the requested control function if the input data satisfies the conditions of the operator control table.

The following control functions are available when you use operator control tables:

NEXTPP Provides the next physical page of the current message.

NEXTLP Provides the next logical page of the current message.

PAGEREQ Provides the logical page requested by the second through last characters of this field. PAGEREQ functions are specified as in operator logical paging. The first character is a page request "trigger" character defined by the customer. The remaining characters must be *n*[*nnn*], +*n*[*nnn*], −*n*[*nnn*], or L (an equals sign (=) is not allowed).

NEXTMSG Dequeues the current output message and provides the first physical page of the next message, if any.

NEXTMSGP Dequeues the current output message and provides the first physical page of the next message, if any; or notifies you that there are no other messages in the queue.

ENDMPPI Terminates a multiple physical page input message. Available only for the 3270.

Unlike operator logical paging requests, these functions are always located by MFS during the editing process.

3270 or SLU 2-Only Feature Definitions

If you use SLU 2 or a 3270, MFS provides several ways to invoke MFS control functions:

- Program function keys and display device fields defined as detectable by the selector light pen can be defined for all MFS control functions except PAGEREQ.
- The PA1 key is equivalent to, and reserved for, the NEXTPP function.
- The PA2 key is equivalent to the NEXTMSG function.
- The PA3 key, when not used for the copy function, is equivalent to the NEXTMSGP function.
- The PF12 key, or PA3 key on data entry keyboards, requests the copy function. This IMS-supported copy function causes a copy of the currently displayed physical page to be printed on an available candidate printer. This printer must be attached to the same control unit (3271 or 3274, for example) as the display station containing the information to be copied.

  Restriction: The request for a copy function is ignored if the device is not defined to allow the copy function or the device does not support the copy function.

  For more information about the copy function, see the DFLD statement field definitions for ALPHA/NUM and NOPROT/PROT 377.

Paging Action at the Device

The paging operation for an MFS device depends on MFS control block definitions, the output message content, and your input. If the device is a printer, each physical page of each logical page is transmitted to the device in sequence and the message is dequeued.

During output paging, if online change processing occurs that changes the format of the output message you access, you can get an error message or get the message in a format different from the one expected.

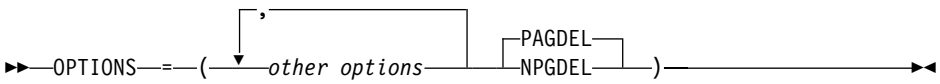
If operator logical paging is **not** specified for a 3604, 3270, SLU 2 display, or SLU P using the DPM paging option, each physical page of each logical page can be viewed in sequence using the NEXTPP function. Because operator logical paging is not specified, entering NEXTPP after the last physical page of the last logical page has been displayed causes the next message to be transmitted if only one exists in the queue. If no message is in the queue, no action takes place.

If operator logical paging is specified for a 3604, 3270, SLU 2 display, or SLU P using the DPM paging option, the NEXTPP function can be used to view pages sequentially. However, entering NEXTPP after the last physical page of the last logical page causes MFS to return an error message and reset the page position to the first page. As noted in “Operator Logical Paging of Output Messages” on page 207, if you are going to view pages out of sequence, the formats should be designed to use the PAGEREQ capability or to have the page request edited to the first field of the first input segment. If not, the screen must be cleared before the page request is entered as unformatted input. For performance reasons, avoid this method.

Table 28 on page 240 describes IMS actions, and the possible message and device status from your input or remote program actions after a successful message transmission.

The following factors must be considered and are included in the figure:

- Macro/statement specifications:
  1. TERMINAL (or TYPE) macro (IMS system definition)



or

When you use the default (PAGEDEL=YES), your input that invokes processing for a new transaction causes the output message for the current transaction to be dequeued. To prevent current output from being dequeued, OPTIONS=(...,NPGDEL,...), or PAGDEL=NO for nonswitched 3270 devices, must be specified.

  2. MSG statement (MOD definition)

PAGE=YES specifies that operator logical paging is permitted. PAGE=NO specifies that paging is not permitted.

- Whether the last physical page of the last logical page in the current message has been sent.
- An IMS action performed automatically after successful message transmission and before your input.
- Your input or remote program action after receiving a message:
  - PAGE ADVANCE: NEXTPP request is entered (or you press PA1 key on 3270 or SLU 2).
  - LOGICAL PAGE ADVANCE: NEXTLP request is entered.
  - =PAGE: specific logical page is requested.
  - PAGEREQ: specific logical page is requested.
  - MESSAGE ADVANCE: NEXTMSG request is entered (or you press the PA2 key on a 3270 or SLU 2 device).
  - MESSAGE ADVANCE PROTECT: NEXTMSGP request is entered (or you press PA3 key on 3270 or SLU 2 when PA3 is not defined for copy function).
  - You enter (or a remote program enters) data that does not invoke an operator control function, followed by enter (or 3270 or SLU 2 PFK, CARD, IMMEDIATE DETECT).

    3270 or SLU 2 operators can also press the CLEAR key. The CLEAR key causes the screen to be unprotected, and subsequent input is edited by IMS basic edit. CLEAR does not affect the status of the current output message. The result of any operator action after using CLEAR is the same as if CLEAR had not been used.
- Table 28 on page 240 uses the following abbreviations to describe IMS action:

  MSG DEQ Message dequeue. IMS removes the current output message from the message queue. The message is available until this action takes place.

  MSG ENQ Message enqueue. IMS places the input message in the message queue.

  PROTECT IMS prevents the device from receiving output from IMS.

  UNPROTECT IMS makes the device eligible to receive output from IMS. If a message is currently queued for this device, IMS sends it (subject to controls established by response mode, conversational or exclusive device status).

If a paged message is sent to the terminal with the unprotected screen option set to “unprotected” (during system definition or using the DSCA or SCA specification), the screen is not protected between pages and the IMS-described actions shown in Table 28 on page 240 should be ignored. If the message is sent to the terminal with the unprotected screen option set to “protect”, the IMS actions shown in Table 28 on page 240 apply. For more information about the unprotected screen option, see “Unprotected Screen Option” on page 242.

Table 28. Paging Operation for a Device with MFS. IMS-MFS Action and Resulting Terminal and Message Status

System/Message definition values and page position in current message with PAGDEL option specified:				
PAGE=	NO		YES	
Last physical page of last logical page of current msg sent?[1]	YES	NO	YES	NO
	IMS action (after successful IMS transmission of message and terminal receipt of message):			
	MSG DEQ, protect	Protected	Protected	
Valid operator action:	**Resulting IMS action:**			
Request PAGE ADVANCE (NEXTPP)	Unprotected	Send next physical page unprotected	Send error message, protected[2]	Send next physical page, protect
Request LOGICAL PAGE ADVANCE (NEXTLP)	Unprotected	Send first physical page of next logical page in current msg [3]	Send error message, protected [2]	Send first physical page of next logical page in current msg [3]
Request specific logical page using =PAGE	Send error message, protected [4]	MSG DEQ, send error message protected [4]	If valid, send first physical page of requested logical page, protected. [2] If invalid, send error message protected. [2]	
Request specific logical page using PAGEREQ	Send error message, protected	Send error message, protected [2]	If valid, send first physical page of requested logical page, protected. [2] If invalid, send error message protected. [2]	
Request MESSAGE ADVANCE (NEXTMSG)	Unprotected	MSG DEQ, unprotected	MSG DEQ, unprotected	
Request MESSAGE ADVANCE PROTECT (NEXTMSGP)	Protected [5]	MSG DEQ, protected [5]	MSG DEQ, protected [5]	
Enter data	MSG ENQ, unprotected	MSG DEQ, MSG ENQ, unprotected	MSG DEQ, MSG ENQ, unprotected	
System/Message definition values and page position in current message with NPAGDEL option specified:				
PAGE=	NO		YES	

Table 28. Paging Operation for a Device with MFS (continued). IMS-MFS Action and Resulting Terminal and Message Status

Last physical page of last logical page of current msg sent? [1]	YES	NO	YES	NO
	IMS action (after successful IMS transmission of message and terminal receipt of message):			
	MSG DEQ	Protected	Protected	
Valid operator action:	**Resulting IMS action:**			
Request PAGE ADVANCE (NEXTPP)	Unprotected	Send next physical page: protected	Send error message, protected. [2]	Send next physical page, protected
Request LOGICAL PAGE ADVANCE (NEXTLP)	Unprotected	Send first physical page of next logical page in current msg. [3]	Send error message, protected. [2]	Send first physical page of next logical page in current msg [3]
Request specific logical page using =PAGE	Send error message, protected [4]	Send error message, protected [4, 2]	If valid, send first physical page of requested logical page, protected. If invalid, send error message protected [2]	
Request specific logical page using PAGEREQ	Send error message, protected	Send error message, protected [2]	If valid, send first physical page of requested logical page, protected. If invalid, send error message protected [2]	
Request MESSAGE ADVANCE (NEXTMSG)	Unprotected	MSG DEQ, unprotected	MSG DEQ, unprotected	
Request MESSAGE ADVANCE PROTECT (NEXTMSGP)	Protected [5]	MSG DEQ, unprotected	MSG DEQ, protected [5]	
Enter data	MSG ENQ, unprotected	MSG ENQ [6]	MSG ENQ [6]	

Table 28. Paging Operation for a Device with MFS (continued). IMS-MFS Action and Resulting Terminal and Message Status

Notes:

1. If an error message has been sent to the last page, do not follow this chart. See note 2.
2. The original message is still in the queue. See *IMS/ESA Messages and Codes* for the proper response to the message.
3. If the current page was the last logical page, no new page is sent, and device status is unprotected.
4. If the device is preset or in conversation, the input is queued; no error message is sent and the device status is unprotected.
5. If a message is in the queue and exclusive or conversational status does not prevent it from being sent, it will be sent. If no message can be sent, a system message is sent indicating that no output is available.
6. The original message is still in the queue. The first physical page of the first logical page is sent unless the device is currently involved in an active conversation. If in conversation, an error message is sent. To continue after a conversational response, NEXTMSG must be entered to dequeue that response.

Unprotected Screen Option

IMS allows you to leave the screen in unprotected status when an output message is sent to the 3270 display and the message is formatted by MFS. This option is provided on a terminal-by-terminal basis or on a message-by-message basis, except messages bypassing MFS. The terminal option of unprotected status applies to:

- All user-output messages that bypass MFS
- All IMS-generated messages (for example, error, `/BROADCAST`, and `/DISPLAY` command output)
- All messages that are formatted by MFS with one of the IMS-supplied default formats or with user-supplied formats

If you do not select the unprotected screen option your messages that are formatted by MFS with user-supplied formats or IMS-supplied default formats, and IMS-generated messages, leave the screen protected or unprotected on a message-by-message basis.

If the message is paged, the screen is unprotected between pages. Therefore, this option is not recommended for paged messages.

Use this option through one of the following:

- SCA output message option of the MFLD statement
- System definition TERMINAL macro specification
- DSCA specification on the DEV statement

Byte 1, bit 5 in the DSCA= operand of the DEV statement and in the SCA output message option of the MFLD statement is defined for protecting or not protecting the screen when the message is sent to the 3270 display:

B'0' Protects the screen when output is sent. B'0' (protected) is the default. This bit is used for autopaged output in ISC.

B'1' The screen is unprotected when output is sent.

If the DSCA value is set to B'0' and PROT (protected) is specified or used as the default on the TERMINAL or TYPE macro, the application program can request that the screen be unprotected when this output is sent (by setting the SCA value to B'1'). If unprotected status is requested when operator logical paging (OLP) is used for the message (PAGE=YES is specified in the corresponding MSG definition), then OLP is reset. You can modify IMS-supplied default formats to set the DSCA value to B'1'.

Whether your messages that bypass MFS leave the display protected or unprotected depends on the OPTIONS specification on the TERMINAL or TYPE macro during system definition. The default is protected.

If MFS formats an IMS message sent to the SYSMSG field of a user-defined format the screen is protected or unprotected depending on the DSCA or SCA option of the format on the device.

When the display is in unprotected status, IMS can send output to the terminal at any time. If you press ENTER, a PA key, or a PF key just before the IMS output, your input or request can be lost. This can be avoided if MFS is used for output and input and you enter the NEXTMSGP function or press PA3 (if PA3 is not used for copy) to obtain protected status before entering input data.

If MFS is not used or is only used for output, and the MOD name specifies DFS.EDT, then PA3 protects input data and must not be used for copying.

Table 29 illustrates the action to be taken (protected or unprotected) by IMS based on the OPTIONS specification on the TERMINAL or TYPE macro during system definition, and the type of output message sent.

Table 29. IMS Protect or Unprotect Action Based on OPTIONS Specification

Output Message	IMS System Definition (PRO)	IMS System Definition (UNPRO)
IMS-generated message with: DSCA|SCA=PROTECT	PROTECT	UNPROTECT
IMS-generated message with: DSCA|SCA=UNPROTECT	UNPROTECT	UNPROTECT
Message using MFS bypass	PROTECT	UNPROTECT
Your message using MFS and user-supplied format or IMS-supplied default format with: DSCA|SCA=PROTECT	PROTECT	UNPROTECT
Your message using MFS and user-supplied format or IMS-supplied default format with: DSCA|SCA=UNPROTECT	UNPROTECT	UNPROTECT

Note:

1. PROTECT: Do not send additional output; wait for input.
2. UNPROTECT: Send output if an output message is available and eligible to be sent.

The 3290 in Partitioned Format Mode

This section describes interactions with the 3290 in partitioned format mode.

Support of 3290 partitioning and scrolling is provided for devices defined to IMS as SLU 2 terminals. Partitioning and scrolling are not provided for devices using BTAM or non-SNA VTAM.

Partition Initialization Options and Paging

You can choose one of three different options for initializing the partition set and paging. The option you select determines how many logical pages of the output message are presented to their appropriate partitions at the initial transmission of a message to a partition formatted screen. (An output message consists of one or more logical pages, each destined for a particular partition according to the DPAGE specifying that partition.) The option also determines how paging requests present additional logical pages to their appropriate partitions. You can specify the option on the PAGINGOP= operand of the partition descriptor block (PDB) statement.

The three options are:

Option 1 The initial data stream presented to the 3290 LU consists of the first logical page of the output message, which is mapped via the DPAGE to the appropriate partition. Thereafter you control all paging with keyed-in paging requests. You use the PA1 and PA2 keys just as in standard, non-partitioned mode. The terminal can be using basic paging support or OLP.

When you request the next logical page, MFS gets the next sequential logical page and sends it to its associated partition. It does not matter which partition is active. A request for the next page results in the next sequential page in the message being sent to the inputting (active) partition or to another partition.

Example: If you enter `=+1`, the next logical page in the message is presented to the appropriate partition, whatever that partition might be. If you enter `=+3`, the page that is sequentially third from the last logical page presented is presented next.

Option 2 The initial data stream presented to the 3290 LU consists of the first logical page of the message and additional logical pages in sequence until the second logical page of any partition is reached, or until the end of the message. Thereafter you control all paging with keyed-in paging requests as described for Option 1.

Option 3 The initial data stream presented to the 3290 LU consists of the first logical page of each partition of the partition set. Thereafter you control all paging with keyed-in paging requests, with one crucial difference from Options 1 and 2: the order in which subsequent logical pages are presented to the partitions depends on the active partition, from which the request is entered. All requests for logical pages apply only to logical pages associated with the active partition.

Example: If you enter `=+1`, the next logical page destined for the active partition is presented—not necessarily the one that happens to be sequentially next in the message. This means that, for the 3290 operator, management of logical paging within the active partition is identical to paging support in a non-partitioned environment.

Regardless of the option chosen, one partition is active after the initial data stream is sent. The active partition is the one in which the cursor is located.

An ACTVPID operand might have been specified on one of the DPAGEs that points to an initialized partition. The ACTVPID allows the application program to declare which partition is the active partition. If option 2 or 3 is being used and data has been sent to several partitions, it is possible that more than one partition has been specified by ACTVPID keywords. In that case, the last partition activated is the active partition. If no ACTVPID keywords are encountered, the active partition is the partition defined by the first partition descriptor (PD) statement in the PDB.

Clearing the Display

There are two levels of clearing the screen and buffer:

- The CLEAR key (X'6D') resets the 3290 to base state, (non-partitioned mode), sets the buffer positions to null, and places the cursor in the upper left corner of the screen. It also places the active message back onto the queue and deletes the control block structure that was created for partitioning.
- The CLEAR PARTITION key (X'6A') resets only the active partition buffer to nulls and clears the active partition viewport. It also places the cursor in the top left corner of the partition. The partition is considered unformatted; any input from it is considered unformatted by MFS and is processed by basic edit.

The JUMP PARTITION Key

Using the JUMP PARTITION key, you can move from one partition to the next, in the order that the PD statements define the partitions in the PDB.

Movement between partitions is determined by the order of the PD statements, not by the order of the associated partition identifier (PID) values.

The partition to which the cursor moves becomes the active partition. Using this key causes no interaction with the host.

Scrolling Operations

The VERTICAL SCROLLING keys cause the data to move up or down in the viewport, so that different parts of the presentation space appear in the scrolling window. The scrolling window is the portion of the presentation space that is mapped to the viewport at a given time. If the viewport has the same depth as the presentation space, the viewport is nonscrollable. If the viewport depth is smaller than the presentation space, it is scrollable.

The amount scrolled each time depends on what is specified by the SCROLLI keyword on the PD statement. The default scrolling increment is one row. Scrolling causes no interaction with the host.

The 3180 in Partitioned Format Mode

IMS support for the 3180 in partitioned format mode is provided through 3290 partitioning and scrolling support. Although interaction with the 3180 and the 3290 in partitioned format mode are similar, the following differences apply:

- With the 3180, only one partition with specific size limits is possible. The 3290 supports multiple partitions of various sizes.
- Logical unit display screen size and viewport location for the 3180 cannot be specified in picture elements (pels). The 3290 supports rows, columns, and pels.
- With the 3180, the single partition is the only one initialized. With the 3290, the application program can determine, with the ACTVPID keyword, which of the various partitions to initialize.

Partition Option and Paging

Because only one active partition is available on the 3180, you can either specify Option 1 on the PAGINGOP= operand of the PDB statement or accept the default of 1. With this option, the initial data stream presented to the 3180 LU consists of the first logical page of the output message, which is mapped by the DPAGE to the single partition. When you request the next logical page, MFS gets the logical page that is sequentially next in the message and sends it to the partition. For more information, refer to the description of Option 1 in “Partition Initialization Options and Paging” on page 244.

Clearing the display and scrolling is handled in the same way on the 3180 as on the 3290 in partitioned format mode.

MFS Format Sets Supplied by IMS

Several format sets are provided by IMS for system use and to serve as defaults when you have not supplied a correct MOD name. The IMS-supplied control blocks reside in the IMS.FORMAT library. When the MFSTEST facility is in use, these control blocks also reside in the IMS.TFORMAT library. They can be used in any IMS installation with MFS by specifying the appropriate MOD name after the `/FORMAT` command. In addition, the format definitions can be used independently by specifying the format name in the SOR= operand of the user-written message definition.

The format definitions supplied by IMS combine with various message definitions to create several separate message formats. All of the format sets except the MFS 3270 and the SLU 2 master terminal formats use one of the following format definitions:

DFSDF1

DFSDF2

DFSDF4

The format for the master terminal is described in “MFS 3270 or SLU 2 Master Terminal Format” on page 247. These format definitions include literals for two of the 3270 or SLU type 2 program function keys, PFK1 and PFK11. Pressing PFK1 inserts the `/FORMAT` command into the first message segment, in front of the entered data. Pressing PFK11 causes a NEXTMSGP request.

System Message Format

The system message format is used for single-segment output messages from IMS and single-segment broadcast messages. It permits two segments of input (transaction, command, or message switch). DFSDF1 is the format name. The MOD name is DFSMO1, and the MID name is DFSMI1. Messages that use this format are eligible for the SYSMSG field on 3270 or SLU 2 devices.

Multisegment System Message Format

The multisegment system message format is used for multisegment messages from IMS and multisegment broadcast messages. It permits an output message of up to 22 segments. DFSDF2 is the format name. The MOD name is DFSMO5, and the MID name is DFSMI2. Messages that use this format are eligible for the SYSMSG field on 3270 or SLU 2 devices. Use the PA1 key to obtain subsequent segments.

Output Message Default Format

For 3270 or SLU 2 devices, the output message default format is used for message switches from other terminals and application program output messages with no MOD name specified. It permits two segments of input (transaction, command, or message switch). DFSDF2 is the format name. The MOD name is DFSMO2, and the MID name is DFSMI2.

Block Error Message Format

The block error message format is used for the `DFS057I REQUESTED BLOCK NOT AVAILABLE` message sent by MFS when an error is encountered during output format block selection. This message is accompanied by a return code (indicating the severity of error) and the block name (the name of the MOD or DOF in error). It can include up to 21 segments of output per logical page. This format permits two segments of input (transaction, command, or message switch). DFSDF2 is the format name. The MOD name is DFSMO3, and the MID name is DFSMI2.

/DISPLAY Command Format

The `/DISPLAY` command format is used for `/DISPLAY` command output. Up to 22 segments per logical page are permitted. This format permits two segments of input (transaction, command, or message switch). DFSDF2 is the format name; The MOD name is DFSDSP01, and the MID name is DFSMI2.

Multisegment Format

The multisegment format is used for entering multisegment transactions and commands. A `/FORMAT` command specifying a MOD name of DFSMO4 can be used to obtain this format. This format is also used for multisegment output messages not exceeding four segments. Up to four segments of input are permitted. DFSDF4 is the format name. The MOD name is DFSMO4, and the MID name is DFSMI4.

MFS 3270 or SLU 2 Master Terminal Format

The MFS 3270 or SLU 2 master terminal format is used when the optional IMS-supplied MFS support for the 3270 or SLU 2 master terminal is selected. This support is described in “MFS Formatting for the 3270 or SLU 2 Master Terminal”.

MFS Sign-On Device Formats

The MFS sign-on device format is used for terminals that require user signon, such as terminals defined with the extended terminal option (ETO). (For more information about ETO, see *IMS/ESA Administration Guide: Transaction Manager*.) The format applies to 3270 and SLU2 devices only. For devices that can receive the formatted `/SIGN ON` command panel (devices with at least 12 lines and 40 columns), the MOD is DFSIGNP, and the MID is DFSIGNI. For devices with smaller screens, the MOD is DFSIGNN, and the MID is DFSIGNJ.

MFS Formatting for the 3270 or SLU 2 Master Terminal

If the IMS master terminal is a 3270 or SLU 2 display device defined as a 3275, 3277 model 2, or 3270-An with SIZE=24×80, you can select the IMS-supplied format that uses MFS. To use the IMS-supplied format you must specify OPTIONS=(...,FMTMAST,...) in the COMM macro during IMS system definition.

When this format is used, the display screen is divided into four areas and several program function keys are reserved.

MFS Formatting

The four areas of the screen are:

Message Area
This area is for IMS command output (except /DISPLAY and /RDISPLAY), message switch output, application program output that uses a MOD name beginning with DFSMO, and IMS system messages.

Display Area This area is for /DISPLAY and /RDISPLAY command output.

Warning Message Area
This area can display the following warning messages:

MASTER LINES WAITING
MASTER MESSAGE WAITING
DISPLAY LINES WAITING
USER MESSAGE WAITING

You can also enter an IMS password in this area.

User Input Area
This area is for your input.

Related Reading: The format and use of these screen areas is described in *IMS/ESA Operations Guide*.

The IMS-supplied master terminal format defines literals for nine of the 3270 or SLU 2 program function (PF) keys. PF keys 1 through 7 can be used for IMS command input. Pressing a PF key inserts a corresponding command into the first message segment in front of the entered data. The keys and their corresponding commands are:

PF Key	Command
1	/DISPLAY
2	/DISPLAY ACTIVE
3	/DISPLAY STATUS
4	/START LINE
5	/STOP LINE
6	/DISPLAY POOL
7	/BROADCAST LTERM ALL

The PF11 key issues a NEXTMSGP request, and the PF12 key requests the copy function.

Do not change the definitions for the master terminal format, with the exception of the PFK literals.

When the master terminal format is used, any message whose MOD name begins with DFSMO (except DFSMO3) is displayed in the message area. Any message whose MOD name is DFSDSPO1 is displayed in the display area. Messages with other MOD names generate the warning message: USER MESSAGE WAITING.

MFS Device Characteristics Table

The MFS Device Characteristics table (DFSUDT0x) is generated during system definition for the 3270 or SLU 2 devices defined as TYPE=3270-An in the TYPE or TERMINAL macro statement.

The MFS Device Characteristics table can be updated with the MFS MFS Device Characteristics Table utility, which allows updates to the table without system regeneration. The '*x*' in DFSUDT0 *x* corresponds to the parameter specified on the SUFFIX= keyword of the IMSGEN macro. Each entry in the table contains the user-defined device type symbolic name (3270-An), associated screen size (from SIZE= parameter), and physical terminal features (from FEAT= parameter). Different specifications of the physical terminal features (FEAT= parameter) for the same device type symbolic name cause separate entries to be generated in the MFS Device Characteristics table.

Related Reading: For a description of the TYPE and TERMINAL macros, see *IMS/ESA Installation Volume 2: System Definition and Tailoring*.

MFS source definitions specify TYPE=3270-An and FEAT as operands on the DEV statement. For the specified device type, MFS extracts the screen size from the specified DFSUDT0 *x* in the IMS.RESLIB library.

The MFS Language utility uses the screen size, feature, and device type specifications to build a DIF/DOF member in the IMS.FORMAT library to match the IMS system definition specification. Because the screen size is specified only during IMS system definition, an IMS system definition must be performed before execution of the MFS Language utility for user-defined formats with DEV TYPE=3270-An.

The MFS Device Characteristics table is created during stage 2 of IMS system definition using the same suffix as the IMS composite control block, nucleus, and security directory block modules as specified in the SUFFIX= keyword of the IMSGEN macro. If terminals defined with ETO are added to the system, the MFS MFS Device Characteristics Table utility can be used to add to or update the table without regenerating the system definition.

Related Reading: For more information about ETO, see *IMS/ESA Administration Guide: Transaction Manager*.

The alphanumeric suffix (*x*) of the table name (DFSUDT0 *x*) is the level identification for the version of the table to be read. The *x* suffix can also be specified using the DEVCHAR= parameter of the EXEC statement for the MFSUTL, MFSBTCH1, MFSTEST, and MFSRVC procedures. Repetitive use of the same suffix by the MFS Language utility causes the same version of the MFS device Characteristics table to be read from the IMS.RESLIB library.

If an MFS Device Characteristics table is required, and either no suffix was provided or the suffixed table is not present in the IMS.RESLIB library, the MFS Language utility attempts to load the IMS Device Characteristics table using the default name (DFSUDT00).

During the logon process for an ETO terminal, the MFS Device Characteristics table is used to determine the MFS device type for the terminal. The screen size from the BIND unique data and the device features from the ETO logon descriptor are used as search arguments.

Related Reading: For more information about ETO, see *IMS/ESA Administration Guide: Transaction Manager*.

Associate only one symbolic name with a given screen size. Establish a standard for relating the device type symbolic name to the screen size.

Recommendation: Use the following screen sizes; for each of the user-defined symbolic names below:

User-Defined Symbolic Name	Screen Size
3270-A1	12×80
3270-A2	24×80
3270-A3	32×80
3270-A4	43×80
3270-A5	12×40
3270-A6	6×40
3270-A7	27×132
3270-A8	62×160

Version Identification Function for DPM Formats

The MFS DOF defines how data is formatted for presentation to the remote program so the remote program can efficiently locate and process the data. The MFS DIF defines how data is presented to IMS from the remote program.

To ensure proper formatting and to present and interpret the data correctly the MFS DOFs and DIFs and the remote program control blocks of the data formats must be at the same level. The current level of the MFS control block is a unique 2-byte field called the version identification (version ID). The version ID is either user-supplied on the DEV statement or, if not specified, it is created by the MFS Language utility at the time the source definition is stored in the IMS.REFERRAL library in an ITB format. The version ID is printed in the information messages DFS1048I and DFS1011I of the MFS Language utility for the DOF or DIF, and must be included in the remote program if verification is to be performed.

The version ID of the DOF used in mapping the output message is provided in the output message header and must be used by the remote program to verify that the control block in the remote program is at the same level as the DOF's version ID.

The version ID of the control block used in mapping the input message to IMS must be provided by the remote program in the input message header. It is used to verify that the correct level of the DIF is provided to map the data for presentation to the IMS application program. If the version ID sent on input does not match the version ID in the DIF, the input data is not accepted and an error message is sent to the remote program. If the verification is not desired, the version ID can be sent with hexadecimal zeros (X'0000') or it can be omitted from the input message header. In this case, both the remote program and MFS assume that the DIF can be used to map the data correctly.

Chapter 8. MFS Application Program Design

Design objectives for MFS application programs should focus on device independence, operator convenience, and application program simplicity. Effective design requires a fundamental understanding of the MFS functions and of the factors that affect MFS operation and performance. This chapter addresses those factors that should be understood and considered when MFS applications are designed.

In this Chapter:

- "Relationships Between MFS Control Blocks"
- "Format Library Member Selection" on page 258
- "3270 or SLU 2 Screen Formatting" on page 261
- "Performance Factors" on page 265

Relationships Between MFS Control Blocks

Several levels of linkage exist between MFS control blocks. You must understand these linkages to design an application environment properly.

Figure 49 on page 252 shows the interrelationships between MFS control blocks. Figure 50 on page 253 through Figure 53 on page 255 illustrate the four levels of linkages, which are then summarized in Figure 54 on page 256.

© Copyright IBM Corp. 1974, 2000

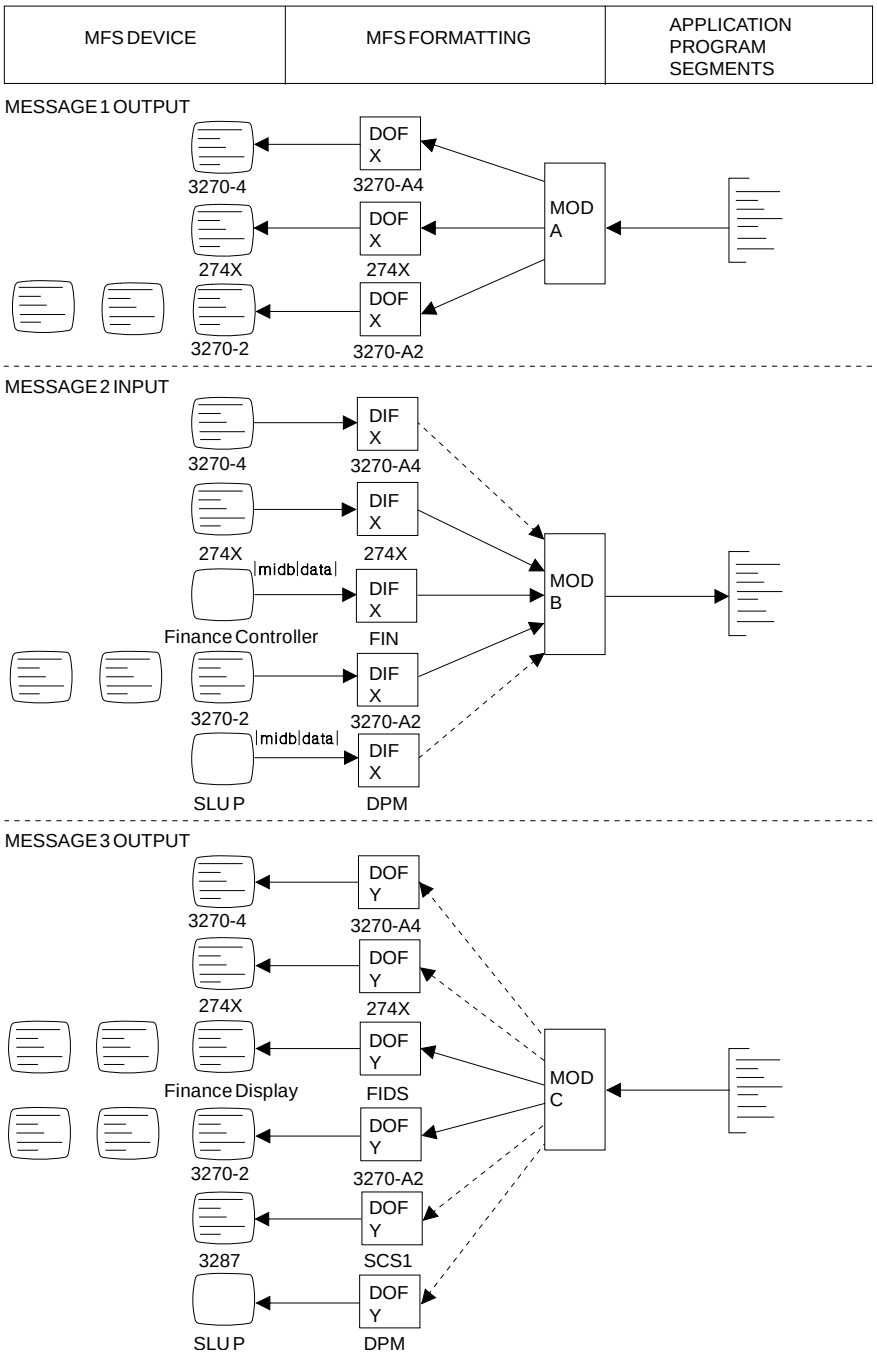


Figure 49. Control Block Interrelationships

Figure 50 on page 253 shows the highest-level linkage, that of chained control blocks.

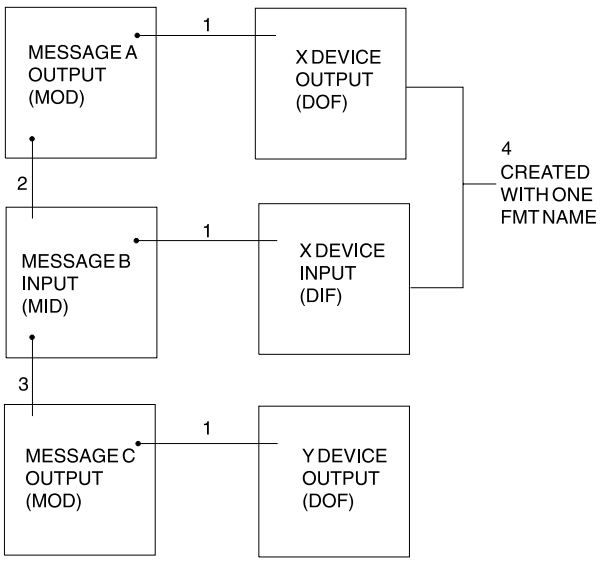


Figure 50. Chained Control Block Linkage

Notes to Figure 50:

1. This linkage must exist.
2. If the linkage does not exist, device input data from 3270 devices is not processed by MFS. For other devices, the MID name can be provided by the operator.
3. This linkage is provided for application program convenience. It provides a MOD name to be used by IMS if the application program does not provide a name via the format name option of the DL/I `ISRT` or `PURG` call. This MOD name is also used if the input is a message switch to an MFS-supported terminal.
4. The user-provided names for the DOF and DIF used in one output-input sequence are normally the same. The MFS language utility alters the name for the DIF to allow the MFS pool manager to distinguish between the DOF and DIF.

   The direction of the linkage allows many message descriptors to use the same device format if desired. One common device format can be used for several application programs whose output and input message formats as seen at the application program interface are quite different.

Figure 51 on page 254 shows another level of linkage that exists between message fields and device fields. The dots show the direction of reference, not the direction of data flow, in the MFS language utility control statements; that is, the item at the dotted end of a line references the item at the other end of the line.

References to device fields by message fields do not need to be in any particular sequence. An MFLD does not need to refer to any DFLD. In this case, MFLD defines space in the application program segment that is to be ignored if the MFLD is for output and padded if the MFLD is for input. Device fields do not need to be referenced by message fields. In this case the fields are established on the device, but no output data is transmitted to them and any input data from them is ignored.

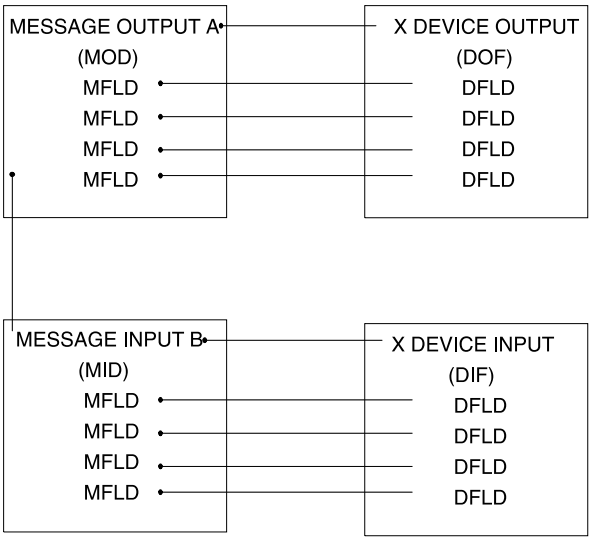


Figure 51. Linkage between Message Fields and Device Fields

Figure 52 shows a third level of linkage, which exists between the LPAGE and the DPAGE.

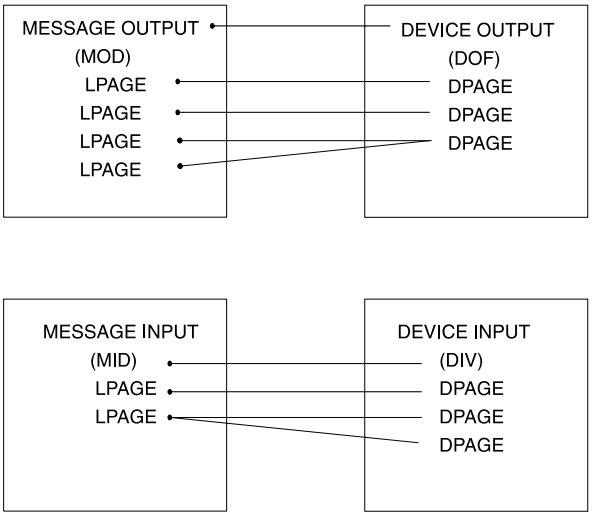


Figure 52. LPAGE and DPAGE Relationships

A MOD LPAGE must refer to a DPAGE in the DOF. However, not all DPAGEs must be referred to from a given MOD.

If no MID LPAGE is defined, the defined MFLDs can refer to fields in any DPAGE. However, input data for any given input message from the device is limited to fields that are defined in a single DPAGE.

If one or more MID LPAGEs are defined, each LPAGE can refer to one or more DPAGEs. All DPAGEs must be referred to by an LPAGE. When input data is processed as defined by a particular DPAGE, the LPAGE referring to it governs the message editing.

Figure 53 shows a fourth level of linkage that is optionally available to allow selection of the MID based on which MOD LPAGE is displayed when input data is received from the device.

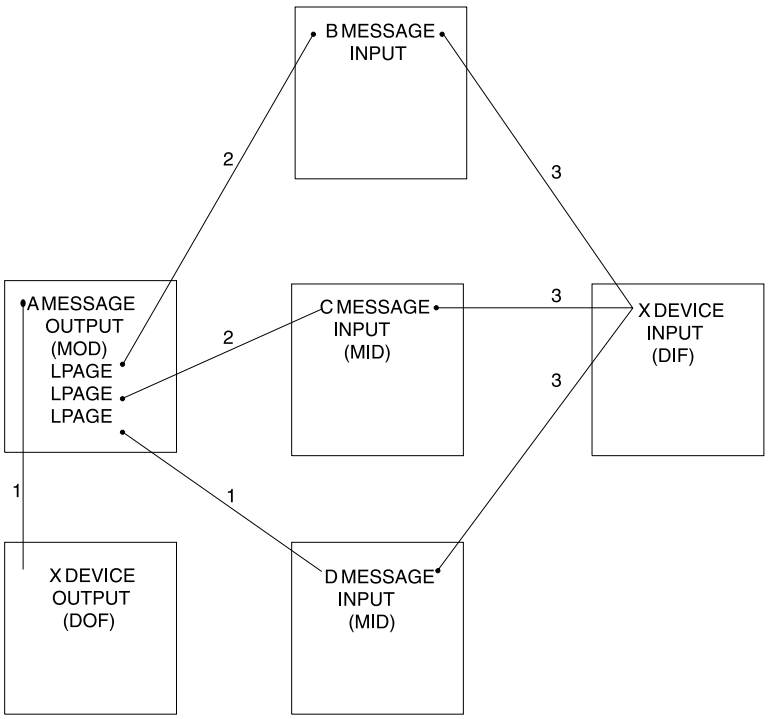


Figure 53. Optional Message Descriptor Linkage

Notes to Figure 53:

1. The next MID name provided with the MSG statement is used if no name is provided with the current LPAGE.
2. If a next MID name is provided with the current LPAGE, input is processed using this name.
3. When the format definition includes 3270 or SLU 2 devices, all MIDs must refer to the same DIF. The same user-provided name must be used to refer to the DOF when the MOD is defined.

Figure 54 summarizes the previously explained MFS control block linkages.

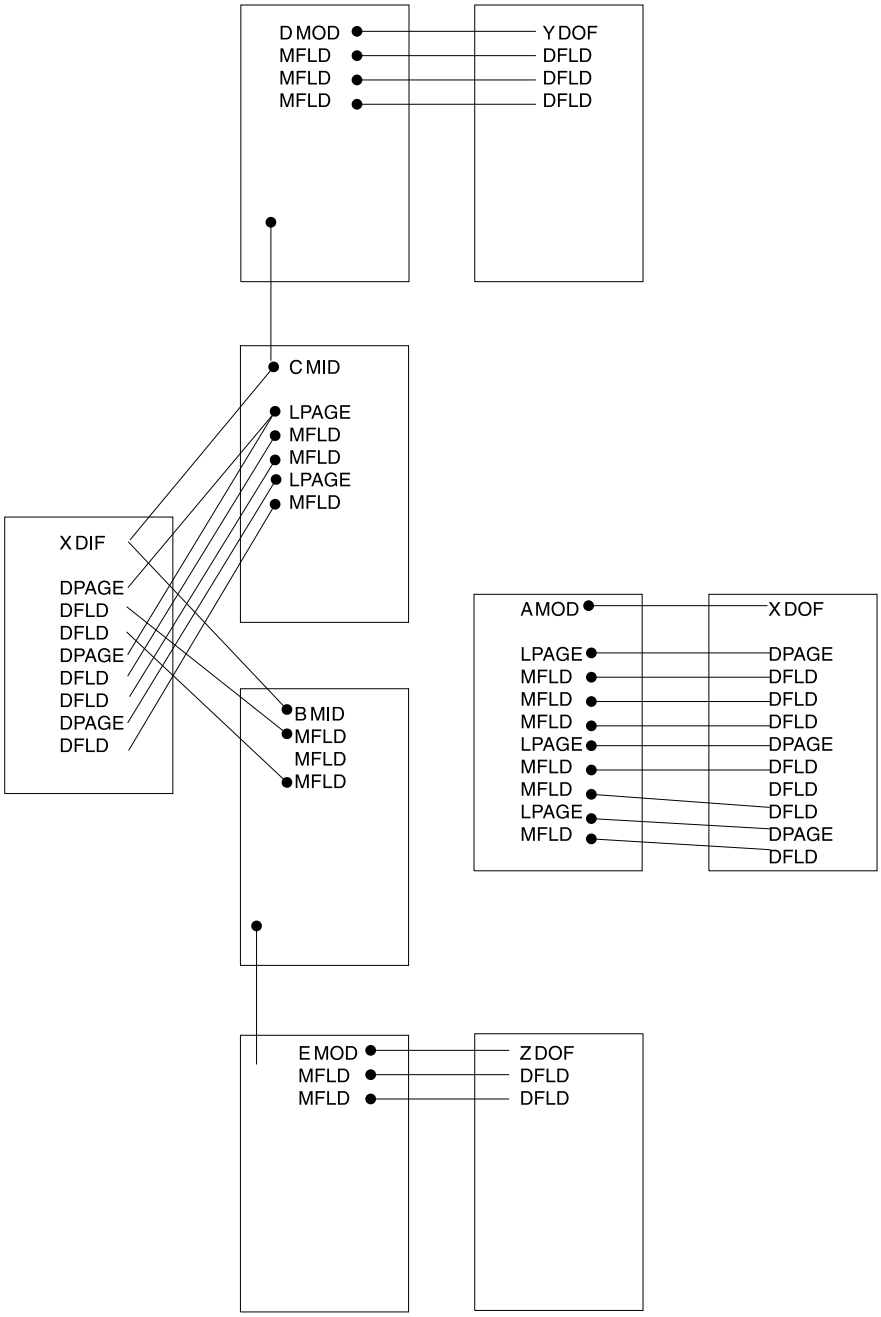


Figure 54. Summary of Control Block Linkages

Device Considerations Relative to Control Block Linkages

Control block linkages are fundamental to MFS functions but there are a few device-oriented conditions that could affect application design.

3270 or SLU 2 Display Devices

Because output to these devices establishes fields on the device using hardware capabilities, and field locations cannot be changed by the operator, special linkage restrictions exist. Because formatted input can only occur from a screen formatted by output, the DPAGE and physical page definition used for formatting input is

always the same as that used to format the previous output. Control block compilation by the MFS language utility verifies that the MID referenced by the MOD refers to the same FMT name that the MOD references. During online processing, if the DIF corresponding to the previous DOF cannot be fetched, an error message is sent to the display.

3290 Information Panel in Partitioned Format Mode

The screen of the 3290 can be divided into several rectangular areas called *partitions*. Depending on LPAGE/DPAGE selection, each logical page of an output message is sent to the partition specified on the DPAGE statement.

When the 3290 is operating in partitioned mode, the usual control block linkages are in effect. There are, however, additional functions, because the logical pages described in the MOD can be sent to different partitions. The partition descriptor block (PDB) is a type of intermediate text block (ITB). The PDB describes the set of partitions that can appear on the screen in response to a single output message. Among other things, the PDB contains one partition definition statement coded with a partition descriptor (PD) for each partition. Taken together, the PDs define a *partition set*.

The linkages work as follows:

1. A MOD is requested for a particular message. The MOD names an FMT and becomes associated with the appropriate DEV statement—in this case, the DEV statement for the 3290. A DOF is created to format the 3290 for the message.
2. The DEV statement itself names a PDB. Thus the MOD is linked to the DOF, which in turn links to the PDB via the DEV statement for the 3290. This linkage gives the logical pages of the MOD (defined by the LPAGE statements) access to the PDs in the PDB.
3. Each LPAGE statement in the MOD names a DPAGE statement in the DOF.
4. For the 3290 with partitioning, a DPAGE statement contains a PD keyword, which identifies one of the partition descriptors in the PDB.

Because of this linkage, each logical page is associated with its appropriate partition that is described by a partition descriptor. When the logical page is retrieved from the message queue, it is sent to that partition.

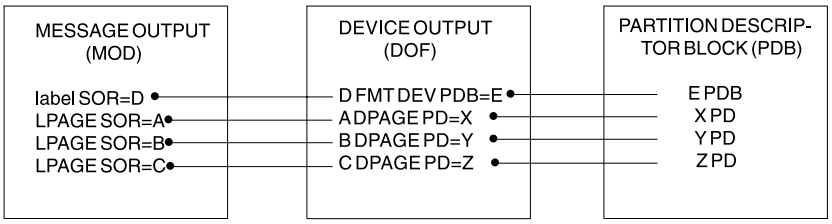


Figure 55. Linkages in Partitioned Format Mode

274X, Finance, 3770, SLU 1, NTO, SLU 4, or SLU P

Because no hardware-established field capabilities exist, no correlation is necessary between output fields and input fields on these devices. Operator input or the user-written program in the Finance or SLU P workstation controller can determine which FMT is used (by specifying a MID name) and which DPAGE within the FMT is used (by the COND= specification for the DPAGE).

Finance or SLU P Workstations

Because of the asynchronous capabilities of the Finance and SLU P workstations, MFS cannot automatically maintain the chain between the MOD and the MID. Therefore the MID name is sent to the device in the output message header. The chain can be maintained, transparent to the operator, if the user-written application program in the remote controller returns the MID name in the input message header.

ISC Subsystem (DPM-Bn)

The NXT=midname that is specified on the MSG TYPE=OUTPUT becomes the RDPN on output and, if not changed by the remote program or subsystem, becomes the DPN on input.

Format Library Member Selection

When a message is received as input or prepared for output, the DIF or DOF is selected on the basis of the user-provided name from the message descriptor and the device type and features of the terminal.

The MFS language utility constructs the member name of each DIF and DOF in the IMS.FORMAT library from the FMT label and the DEV TYPE= and FEAT= specifications as follows:

Byte	Contents
1	Device type indicator (hexadecimal). For a list of device types by indicator, see page 258.
2	Device feature indicator (hexadecimal). For a list of indicators by feature, see page Table 30 on page 260.
3	If DOF, first character of label provided in the FMT statement. If DIF, first character of label provided in the FMT statement **converted to lowercase**.
4-8	Remaining characters from the label of the FMT statement.

For byte 1 of the DEV specification FMT=, the device type indicators are as follows:

Indicator (Hex)	Device Type
00	SLU 2, Model 1 display
01	3284-1 or 3286-1 printer
02	3277, or SLU 2, Model 2 display
03	3284-2 or 3286-2 printer
05	3604-1 or 2 (FIDS)
06	3604-3 (FIDS3)
07	3604-4 (FIDS4)
08	3600 (FIN)
09	3610, 3612 journal printer (FIJP)
0A	3611, 3612 passbook printer (FIPB)
0B	3618 administrative printer (FIFP)
0C	SCS1: 3770; NTO; SLU 1 (print data set), and SLU 4

0D	SCS2: 3521 card punch, 3501 card reader, 2502 card reader, SLU 1 (transmit data set), and SLU 4
0E	3604-7 (FIDS7)
11 through 1F	DPM-A1 through DPM-A15, respectively
21 through 2F	DPM-B1 through DPM-B15, respectively
41 through 4F	3270-A1 through 3270-A15, respectively

Recommendation: You should define device formats for each device type expected to receive a given message. If the MOD or the DOF with the required device type and feature specification cannot be located during online execution, the IMS error default format (containing an error message) is used to display the output message. If the MID or the DIF with the required device type and feature specification cannot be located, input is ignored and an error message is sent to the device that entered it.

However, it is possible to use the same format for a variety of specific devices. Formats defined as TYPE=3270,2 with FEAT=IGNORE specified, can be used as default formats for users of the following devices:

- 3275
- 3276, models 2/3/4
- 3277, model 2
- 3278, models 2/3/4
- 3279, models 2/3

To define the terminal to IMS, you must specify TYPE=3270-An with SIZE=(n,80), where n≥24.

Restriction: The IGNORE feature is not supported in MFSTEST mode.

The terminal must be defined to IMS as TYPE=3270,2 or MFS searches for a block with the exact TYPE and FEAT specification, and if one is not found, MFS searches for the default TYPE=3270,2 with FEAT=IGNORE.

Another level of defaulting occurs for ETO terminals prior to the already described defaulting. If an ETO terminal is defined with a screen size of 12x40 or 24x80 in the VTAM PSERVIC information, and that format block is not found, an additional search is made for a format of the same name using TYPE=3270,1 (12x40) or TYPE=3270,2 (24x80) and using the same features. If that search is unsuccessful, the already described default search is performed. This new default search is also used when in MFSTEST mode, whereas the old default search is not.

Device format selection is based upon the features of the destination terminal as defined at IMS system definition. If feature selection is used, a device format must be created for every combination of features in the system that can receive a message using feature selection. Feature selection is performed based on the specification of the message descriptor (MOD or MID). If the IGNORE option is specified on the MOD, device formats must be created with the IGNORE feature option to ensure proper operation.

Because the screen size for 3270 or SLU 2 devices, other than 3270 model 1 or 2, is specified during IMS system definition, an IMS system definition must be performed before execution of the MFS language utility for user-defined formats.

Format Library Member Selection

Use feature selection when devices with different feature combinations are to receive or enter a message and the special features of each device are to be used.

Example: An operator at a device with program function keys can enter a literal in a field using a program function key, and another operator at a device without program function keys can enter the same literal by typing it in a field on the screen. To the application program, these literals are the same. To the application program, the following input devices can enter messages that can look identical regardless of how they were entered:

- Device Features
- Print Line 120
- Print Line 126
- Print Line 132
- Data Entry Keyboard
- Program Function Keys
- Selector Light Pen Detect
- Magnetic Card Reading Devices (OICR and MSR)
- Dual Platen
- User-defined features for the 3270, SCS1, and SCS2 devices and DPM programs

Use the device feature indicator values listed in Table 30 for byte 2 of the DEV FEAT= specification:

Table 30. Example of Device Feature Indicator Values

Device Feature	Indicator (Hex)
P.L. 120 (Print Line 120)	40
P.L. 126	50
P.L. 132	60
DEK (data entry keyboard)	C8
PFK (program function keys)	C4
SLPD (selector light pen detect)	C2
OICR/MSR (magnetic card reading devices)	C1
IGNORE	7F
DEK,SLPD	4A
DEK,OICR	C9
DEK,SLPD,OICR	4B
PFK,SLPD	C6
PFK,OICR	C5
PFK,SLPD,OICR	C7
SLPD,OICR	C3
DUAL (dual platen)	C1
P.L. 132,DUAL	61
No features (3270)	40

Table 30. Example of Device Feature Indicator Values (continued)

Device Feature	Indicator (Hex)
3270,3270P,3770,SLU 1, SLU 2,SLU 4,SLU P,ISC (User-defined features)	Indicators available for definition: 1. 01 2. 02 3. 03 4. 04 5. 05 6. 06 7. 07 8. 08 9. 09 10. 0A

3270 or SLU 2 Screen Formatting

MFS is designed to transmit only required data to and from the 3270 display device. Device orders to establish fields and display literals can cause significant transmission time—there can be more orders and literal data than program data. Under normal operation, when the format to be displayed already exists on a device, only user-supplied data from the message and modifiable field attributes are transmitted. The current format on the device is determined by the device output format name, the DPAGE within the format, and the physical page within the DPAGE. The following conditions cause MFS to perform a full format operation (device buffer erased and all fields and literals are transmitted) for device output:

- The device output format changes.
- The DPAGE changes within a device output format.
- The physical page number changes.
- The operator presses the CLEAR key.
- The operator presses the CLEAR PARTITION key, which causes a full format write to the cleared partition.
- DSCA option of the DEV statement requests format write.
- SCA field in the output message requests format write.
- The MFS bypass has been used.
- Terminal has been stopped as a result of a permanent I/O error. [4]
- The operator uses the operator identification card reader. [4]

A full format operation must be carefully planned. Several factors can result in undesirable screen displays, program input, or both:

1. If the program depends upon the existence of data in nonliteral fields and does not include this data in the output message, the data might not be on the screen when the device receives the output message. Several actions can cause this to occur:
   - The terminal operator pressing the CLEAR key
   - A device error
   - Another message sent to the device before the response

4. The screen is cleared and the next output is a full format operation.

- An IMS restart

This dependency also makes the application 3270 device-dependent.

2. If the program sends only part of an output field, data that already exists in the nonliteral fields can cause confusion. If a partial field is transmitted to a filled-in field, any modification of the field causes the old data remaining in the field to be included in the new input. Use the PT (program tab X'05') as a fill character on the DPAGE statement to solve this problem. If the PT fill character is specified, message data fields (and message literal fields) that are to be transmitted are followed by a program tab order if the data does not fill the device field. This clears the remainder of the device field to nulls.

   When a program sends only a few of the output data fields on a given display screen, it might be desirable to clear all the unprotected filled-in fields first. The unprotected fields can be cleared by specifying the “erase all unprotected” option in the application program output with the system control area (SCA) operand of the MFLD statement or the default SCA (DSCA) operand of the DEV statement.
3. Premodified attributes can be requested by the application program to ensure input of field data. If premodified attributes are requested and the message was completely transmitted to the device and not operator logically paged, then a device error, or IMS restart, prevents input. This error occurs because the screen is not reestablished with the message when the terminal is started or IMS is restarted.
4. If dynamic attribute modification is specified for a device field with predefined attributes, an attribute is sent to the device for that field in every output operation, even if the data for this device field is not included in the output message. These attributes are used in the following ways:
   - If the output message field has an attribute and the attribute is valid, then the dynamic attribute modification is performed.
   - If the message field is not included in the LPAGE being used or the attribute is not valid, the predefined attribute for the device field is used.

Recommendation: For application design, you should:

1. Use a common device format for as many applications as possible. Reducing the number of full format operations can significantly reduce response time. Format block pool requirements are reduced as well as format pool I/O activity.
2. Allow MFS to determine when a format operation is required. This results in transmission time savings when formatting is not required.
3. Ensure that the application program output message contains all nonliteral data required by the device operator. Do not rely on previous data remaining on the device.
4. Use the PT fill option to ensure that fields on the device that receive program output data contain only data from the message.
5. Use the erase all unprotected option of the SCA or DSCA if the application requires that unprotected fields be cleared.

Two MFS facilities are available for controlling format operations. Both the system control area (SCA) of the message field and the default SCA (DSCA) option of the DEV statement provide the ability to cause IMS to force a reformat or to erase all unprotected fields or all partitions before transmitting output. The force format write option causes the device buffer to be erased, all fields to be established, and all literals to be transmitted. The erase all option causes all unprotected fields or all partitions to be cleared to NULLs before data is written. For more information, see “System Control Area (SCA) and Default SCA (DSCA)” on page 209.

3290 Screen Formatting

A 3290 screen can be divided into several independent areas, called *logical units* (LUs). Each LU can be in base state or formatted state. If it is in formatted state, the LU can be in standard or *partitioned* format mode. Descriptions of 3290 screen formatting follow.

Screen Division

The 3290 has a large screen, which allows the display of up to 62 rows by 160 columns for small character cells (6 × 12 pels), and up to 50 rows by 106 columns for large character cells (9 × 15 pels).

The 3290 screen can be divided into several areas, each of which interacts independently with the operator. This can be done in two ways:

- By dividing the screen into separate LUs
- By dividing a logical unit into separate partitions

In the first case, the 3290 terminal and its screen can be defined as up to four separate LUs. Each LU is independent of the others, and is defined to IMS as a separate terminal with its own address. This support is transparent to IMS. Defining multiple LUs is useful if the IMS application calls for more than one input or output message (or both) to be concurrently active between the 3290 terminal and IMS. For each logical unit, however, only one input or output message can be active.

In addition, with software partitioning, each logical unit can be divided into as many as 16 partitions. Each application message can specify a set of partitions, and each logical page of the message is associated with a particular partition of the partition set. Software partitioning is useful if:

- The operator needs to view more than one logical page at a time.
- One partition is needed to view an output page and another to input data.
- A partition is to be defined to receive IMS system error messages while the logical unit is in formatted mode. This function could be used in place of the current MFS SYSMSG field support.
- Scrolling is desired. Scrolling moves data up and down in the partition viewport. It can be defined only for a 3290 in partitioned mode. With explicit partition scrolling, you can define MFS pages for a presentation space larger than the viewport on the physical screen. This reduces the number of interactions between IMS and the terminal that must occur to display the message.

The 3290 allows a maximum of 16 partitions per physical device. Also, each LU defined in partitioned state must have available to it a minimum of 8 partitions, no matter how many partitions are actually defined for it. Thus, if one LU is defined with 9 partitions, no other LU can be in partitioned state, because there are only 7 partitions left for the physical device. Consequently, no more than 2 LUs (of the maximum 4 allowed) can be in partitioned state.

The following considerations also apply to defining partitions:

- Partitions must be rectangular.
- A single input message is constructed from one physical page of a single partition unless Multiple Physical Page Input is used. If it is used, then multiple physical pages for a single input message must come from a single partition.
- If the current PDB does not define a partition for system messages, and if the DOF does not define a system message field, then a system message destroys

the current partitioned format mode and the 3290 (or the particular LU in question) returns to standard format mode.

Terminal States and Modes

The 3290 as a single LU, or any of the LUs into which it has been divided, can be in terminal base state or terminal formatted state.

In terminal base state, the 3290 operates in the same way as any other currently supported SLU 2 node when it is initially connected to IMS or when the clear key has been pressed. In this state, input messages to IMS are edited with basic edit, and output messages without an associated MOD are formatted using the default MFS MOD.

In terminal formatted state, the 3290 can be in:

- Standard format mode
- Partitioned format mode

The choice of format mode is made dynamically at the time of message output. The output message is associated with a MOD, which in turn names a DOF. The specifications in the DOF determine the 3290 format mode:

- The 3290 is in standard format mode if the DOF does not name a partition descriptor block (PDB). The terminal is then formatted and operated as an ordinary SLU2 node.
- The 3290 is in partitioned format mode if the DOF names a partition descriptor block (PDB).

Partition Set Initialization, Paging, and Activation

If the 3290 (or any of the LUs into which it can be divided) is in partitioned format mode, there are various ways in which:

- The partitions are initialized with one or more logical pages from the output message.
- The operator subsequently controls the flow of logical pages to the partitions.
- One particular partition becomes the *active partition*.

Initialization and operator-controlled paging are determined by selecting one of the three options. The option is specified by the PAGINGOP operand of the PDB. According to the selected option, initialization can consist of:

1. The message's first logical page going to the appropriate partition
2. The message's initial logical pages going to their appropriate partitions until the second logical page of any partition is reached
3. Each partition receiving its first appropriate logical page

The option also determines whether operator-controlled paging is affected, depending on which partition is active.

When the 3290 enters partitioned format mode, one particular partition is the *active* partition. This is determined in one of two ways:

- Logical pages are routed to their partitions via DPAGE statements. An ACTVPID operand might have been specified on one of the DPAGEs that points to an initialized partition. The ACTVPID allows the application program to declare which partition is the active partition.
- If no ACTVPID keywords are encountered, the active partition is the partition defined by the first PD statement in the PDB.

The active partition can be a partition that has not initially received any data.

For more details about initialization, paging, and activation, see “The 3290 in Partitioned Format Mode” on page 243.

3180 Screen Formatting

Like the 3290, the 3180 terminal is supported by IMS as an SLU 2 device. Partitioning and scrolling support for the 3180 is similar to what is provided for the 3290.

Exceptions: For the 3180:

- Only one partition with specific size limits can be defined. (For the 3290, multiple partitions of various sizes can be defined.)
- Logical unit display screen size and viewport location cannot be specified in picture elements (pels). (The 3290 supports pels.)
- You cannot specify an active partition. (For the 3290, active partitions can be specified.)

These restrictions apply only if you want the 3180 screen size when it is connected to IMS to differ from the 3180 screen size when it is connected to other subsystems. If no change is required, then the 3180 customer set up installation instructions can be used and no special IMS code is necessary.

Performance Factors

The design of message and device formats usually has only a minor effect on the time or resources required to edit a message. It can, however, have a considerable effect on transmission and response time. The following considerations affect performance.

All MFS-Supported Devices

The IMS `/DISPLAY POOL` command can be used to evaluate format pool operation. The objective should be to reduce the value of:

```
I/O+DI    (The sum of the numbers of fetch
 REQ1      I/O operations and directory I/O operations
           divided by the number of block requests from
           the pool.)
```

To reduce this value, do one or more of the following:

- Reduce format block I/O. The most significant and tunable portions of MFS processing cost are the CPU cycles and channel/device time required to read format blocks. To reduce format block I/O, use the following techniques:
  - Evaluate and implement $$IMSDIR, the optional MFS index directory. Index the selected MFS control blocks based on how frequently they are used. In most cases, using $$IMSDIR eliminates one read per format block during online operation.
  - Increase the size of the MFS buffer pool.
  - Increase the number of fetch request elements (FREs).
- Minimize the number of segments. Messages should be segmented for application program convenience or to meet segment size restrictions. Segment processing in MFS and DL/I requires a considerable number of CPU cycles, so do not segment unnecessarily.
- Use option 2 input. In some cases, the application input can be segmented so that no device input can be presented for segments under certain conditions. In

such cases, option 2 input messages reduce processing time slightly and reduce IMS message queue space requirements.

- Use option 3 input. Option 3 input can provide better performance than option 1 or 2 if many fields are defined, but only a few fields are received on input. Additional buffer pool space is required during editing, but message queue space requirements are reduced. When most of the defined fields are received on input, option 3 performance is not as good as 1 or 2, either in processing time or in message queue space.

  For an explanation of input message formatting options, see “Input Message Formatting Options” on page 186.
- Combine multiple DFLD literals. When multiple DFLD literals are positioned at adjacent or nearly adjacent device field locations, consideration should be given to combining the literals in fewer DFLD literal definitions. The only limitation to the number of literals combined is the maximum DFLD literal length. Combining DFLD literals reduces block size, reducing MFS processing time and, for 3270 or SLU 2 display devices, reducing transmission time.
- Do not define DFLDs that are not referred to by any MSG descriptor. Such DFLDs occupy block space and, if used extensively, could adversely affect MFS processing time.
- Combine output message fields if appropriate. Where multiple, contiguous, output message fields of a segment map to contiguous device fields of the same relative length, consider combining both the message fields and the device fields so that a single message field maps to a single device field. The greatest potential advantage is in those situations where only one blank separates the displayed fields, and message data is always present and equal to the device field length.

  Combining message fields is not recommended, however, in cases where an additional formatting burden would be placed upon the application program.
- Do not define duplicate formats.
- Do not define separate formats for simple input. Most MFS device formats should include some user input fields that allow the operator to enter any simple transaction or command, related or not related to the application for which the format was designed. Any format requires four control blocks, and formats designed specially for simple input should not be defined unnecessarily.

3270 or SLU 2 Display Devices

To enhance system performance when using 3270 or SLU 2 display devices, you can do the following:

- Use preformatted screens. This is the most significant performance consideration for MFS when 3270 or SLU 2 display devices are used. Significant amounts of data are usually required to define fields and establish literals on a screen. These field definitions and literals do not always have to be transmitted (see “3270 or SLU 2 Screen Formatting” on page 261). If the format on the device can be used, transmission time for remote terminals can be reduced up to 50 percent.
- Pad message output with nulls. The use of the FILL=NULL or PT option in the DPAGE statement reduces the amount of data transmitted to the device and the amount of processing required to format the output.
- Reduce mixed-mode operations. A mixed mode operation occurs when the selector light pen is used on an immediately detectable field and other fields on the device are modified. The mixed mode operation requires multiple I/O operations that increase response time, line utilization, and processing time. In addition, the resulting message contains the same data as would be produced by the enter key except for the indication that the selector pen was used.

- Use paging requests. Where application design permits, the PA1 (program access key 1) page advance facility should be used instead of operator entry of a logical page request. The PA1 facility requires less operator action and less communication line time, and does not require input editing before page request processing.
- Define the length of a literal DFLD followed by a nonliteral DFLD to include space between the last significant literal character and the position preceding the attribute position of the nonliteral field. This action can reduce block size and character transmission but should only be considered when the separating space is between two and five characters.
- Increase the length of DFLDs with the PROTECT attribute. When a nonliteral DFLD is defined with the PROTECT attribute, separated from the next device field by two or more blanks, and is expected to receive output data, consider increasing its length. The output data can originate from an application program, a `/FORMAT` command, or an MFLD literal. Multiple MODs can be used to map message data to the DFLD. Increasing DFLD length should reduce character transmission unless character fill (FILL=C' ') is specified. Specifying FILL=C' ' is not recommended.
- Minimize the use of the CLEAR key. Advise terminal operators not to use the CLEAR key unnecessarily. In addition, explain to terminal operators the proper use of other function keys such as the ERASE INPUT and ERASE EOF.

  Design screen formats with the objective of minimizing the use of the CLEAR key. Allow simple input from a formatted screen. To provide for this capability, establish the same device field location of all formatted screens as the standard device field for simple input. Enforce this standard for all format definitions.

  Decreasing CLEAR key usage can improve response time and use communication lines more effectively.

3270 or SLU 2 Devices with Large Screens

In addition to the performance factors listed in the previous section, the following performance factors apply only to large-screen devices:

- If pages are combined for display on large screens, operator paging is reduced proportionally to the reduction of number of pages. If the OUTBUF keyword of the IMS system definition TERMINAL macro or ETO logon descriptor cannot specify the amount of data for an entire page, more than one VTAM SEND is required to send the page.

  Related Reading: For more information on ETO, see *IMS/ESA Administration Guide: Transaction Manager*.
- For remote BTAM 3270s, IMS sends a maximum of 4 KB of data in one transmission. For local 3270s and remote VTAM 3270s, IMS sends the entire message in one transmission. These facts and the line error rate should be considered when designing support for large-screen devices.

SLU P and ISC Subsystems with DPM

If OPTIONS=PPAGE is specified in the DIV statement, the set of fields in a PPAGE (presentation page) is transmitted together in one or more records. Additional presentation pages are sent on request of the remote program or ISC subsystem for demand paging. This level of paging is the simplest for the remote program or ISC subsystem to process but imposes the most burden on IMS processing time.

If OPTIONS=DPAGE is specified, all fields within a logical page are transmitted together in one or more records. Additional logical pages are sent on request of the remote program or ISC subsystem for demand paging. This level of paging makes it

more difficult for the remote program or ISC subsystem to process the data if more than one presentation page is included, but imposes less burden on IMS processing time.

If OPTIONS=MSG is specified, all the data within a message is sent together and no paging is performed. This technique might require more processing and logic in the remote program or ISC subsystem but is the best for IMS performance if all pages are actually used by the remote program or ISC subsystem. If many pages are not used by the remote program or ISC subsystem, this option results in unnecessary line traffic and IMS processing.

If autopage is specified (SCA byte 1, bit 5) and option PPAGE or DPAGE is desired for DPM-Bn, all data within the message is sent and no demand paging is performed.

The RCD statement can be used to influence the placement of fields within records. The DFLD that follows the RCD statement begins in the first user data location of a new record. Fields can be placed in records so that no field spans a record boundary, or so that logically related fields appear together in the same record.

Restriction: For ISC subsystems, fields cannot span records.

Use of the RCD statement to set record boundaries can reduce transmission time and IMS processing time only if records of maximum length are created. If field placement into records is controlled using the RCDCTL specification only, the SPAN option causes the minimum number of records to be sent to the remote program. Use of SPAN requires, however, that the remote program put together the fields that have been split across records.

Loading Programmed Symbol Buffers

If programmed symbol (PS) buffers are desired and if they have not been loaded by another means (for example, a VTAM application), the buffers must be loaded.

Using an Application Program to Determine Whether Programmed Symbol Buffers Are Loaded

The buffers might have been loaded with the desired programmed symbols by a previous user of the device, and this knowledge can save resending the entire programmed symbol data stream. A handwritten log at the device is one method of maintaining the current status of the programmed symbol buffers for subsequent users.

Another method is a user-written application program that attempts to use the desired programmed symbols. If the desired programmed symbols are already loaded, the output from the application program is successfully displayed at the device. If the programmed symbols are not loaded, the output message is returned to the IMS message queue, the terminal is made inoperable, and a message is sent to the master terminal operator (MTO). The MTO should have a procedure to correct this condition. For example, the MTO could do one of the following:

- Reassign the LTERM, assign an LTERM that has the correct PS load message, restart the terminal, and then reassign the first LTERM back to the terminal.
- If the terminal does not have PS capability, reassign the LTERM to one that does.
- If the terminal does not have PS capability, dequeue the rejected message.

Exception: For an SLU 2 terminal, the output rejected was not a response mode reply. In this case, the MTO receives the error message and can try to enter a transaction that would cause the buffers to be loaded.

How to Load the Programmed Symbol Buffers

If the operator knows the programmed symbol buffers need loading (because the device was just turned on, from one of the methods described above, or some other method), the operator should enter a response mode transaction that loads the programmed symbols.

Make available, to all users at the installation, a user-written application program to load the programmed symbols. The first part of the message sent by this application program should be the programmed symbol data stream, and the remainder should be some user data displayed at the device (such as THE PROGRAMMED SYMBOL LOAD FOR *programmed-symbol-name* COMPLETE). The user data displayed at the device informs the terminal operator when the programmed symbols have been loaded. This application program should use the MFS bypass option, because the write structured field (WSF) 3270 command used to send the programmed symbol message is only supported by IMS through the MFS bypass option.

When the programmed symbol buffers that are to be loaded include a printer or a different display, other techniques must be used. Programmed symbol buffer loads are restricted to 3 KB for BSC-attached devices.

Example: The following shows the loading of a programmed symbol buffer using an automated operator interface (AOI) application program.

1. The operator at display A enters a transaction (response or conversational) requesting programmed symbol loads for display A, printer B, and display C.
2. Another AOI transaction assigns LTERMs for printer B and display C, temporarily, to a special PTERM. The AOI program assigns dummy LTERMs to printer B and display C.
3. The AOI program inserts a programmed symbol message to the dummy LTERMs of printer B and display C.
4. The AOI program sends programmed symbol messages to display A.
5. The operator visually verifies messages on both displays and the printer and confirms that the transaction executed correctly.
6. Another AOI transaction reassigns LTERMs to their original status.

Solving Programmed Symbol Load Problems

If a hardware error occurs while a programmed symbol buffer is being loaded, then the following actions occur:

1. The programmed symbol load message is returned to the IMS message queue.
2. The terminal is taken out of service, except for SLU 2 devices when programmed symbols are not available.
3. The error is logged to the IMS log.
4. A message is sent to the IMS master terminal.

Once the hardware error is corrected and the terminal is in service, the programmed symbol load message is re-sent.

If the programmed symbol load failed because of an error in the programmed symbol load message, the operator must:

1. Dequeue (/DEQ) the message (the master terminal operator might have to issue the /DEQ command).
2. Correct the error.
3. Reenter the transaction to send the programmed symbol load message again.

If a method is available for informing the next user of the programmed symbol buffer status, then the terminals with loaded programmed symbol buffers should not be turned off. When a power failure occurs, or a terminal is turned off, the contents of the programmed symbol buffers are lost.

When a terminal is turned on and no IMS messages are waiting to be sent to the display, load all required programmed symbol buffers using an IMS transaction (or some non-IMS method). However, if IMS messages are waiting to be sent, and these messages require the use of one or more programmed symbol buffers, the sending of the queued messages must be delayed until the programmed symbol buffers can be reloaded. This can be accomplished using response mode transactions to load the programmed symbol buffers.

If the programmed symbol buffers are not loaded and a message that requires a programmed symbol buffer is sent to the terminal, the following actions take place:

- For non-SLU 2 devices, IMS takes the terminal out of service, sends a message to the master terminal, and returns the output message to the message queue.
- For SLU 2 devices, the message is rejected and a sense code is returned to IMS. IMS then:
  - Returns the invalid message to the IMS queue.
  - Logs the error to the IMS log.
  - Sends an error message to the IMS master terminal if the output was a response mode reply, and takes the terminal out of service. If it is not in response mode, the error message is sent to the terminal and it is left in protected mode.

If the user-written application program is designed to queue an unsolicited message requiring a particular programmed symbol load buffer to an LTERM, the first part of the message could include a load programmed symbol data stream; however, this message could not be processed by MFS.

When a message is waiting on the IMS queue for a terminal and requires a programmed symbol that is not loaded, perform one of the following:

- If the terminal is attached by VTAM, load the programmed symbol buffers using a VTAM application.
- If a queued message requires a programmed symbol buffer and it is “normal” user output (for example, the output is not response mode or conversational), then the use of a response mode transaction to load the programmed symbol buffer permits the queued message to be properly displayed. If loading the buffers requires multiple messages, multiple response mode transactions can be used.
- Dequeue (/DEQ) the message (or have the master terminal operator dequeue the message) requiring use of a programmed symbol buffer; enter a transaction to load the programmed symbol buffer required; and then reenter the transaction that originally generated the queued message.
- Temporarily assign the LTERM to which the message is queued to another physical terminal. Load the programmed symbol buffer, then reassign the LTERM to the original physical terminal. The LTERM must be assigned to a terminal that

will not cause a message to be sent (as, for example, a 3270 display or SLUTYPE2 that is in protected screen mode).

Chapter 9. Application Programming Using MFS

This chapter contains information for application programmers whose programs communicate with devices using MFS. It describes general MFS items and specific device-oriented items that govern the format of input and output messages.

In this Chapter:

- "Input Message Formats"
- "Output Message Formats" on page 275

Input Message Formats

MFS edits input data from a device into an IMS application message format using the message definition that the MFS application designer writes as input to the MFS language utility program. An input message consists of all segments presented to an IMS application program when the program issues a DL/I GU or GN call.

The format of input messages is defined to the MFS Language utility. Each message consists of one or more segments; each segment consists of one or more fields:

```
MESSAGE
    SEGMENTS
        FIELDS
```

Message field format is defined specifically to the utility in terms of data source, field length, justification, truncation, and use of fill (pad) characters. How MFS actually formats the field is a function of the formatting option selected for the message. The option used is identified in the second byte of the 2-byte ZZ field (Z2) preceding the message text. An application program that depends on MFS should check this field to verify that the expected option was used; a X'00' in the Z2 field indicates MFS did not format the message. The format options are explained and illustrated with examples in "Input Message Formatting Options" on page 186.

Logical Pages

For 3270 or SLU 2, the input message is created from the currently displayed DPAGE on the device. For some other devices, if the device input format has more than one DPAGE defined, the device data entered determines which input LPAGE is selected to create an input message. However, for ISC (DPM-Bn) subsystems, OPTIONS=DNM or COND= can be used for DPAGE selection. For more information, see "Input Format Control for ISC (DPM-Bn) Subsystems" on page 201.

When LPAGEs are defined, each LPAGE is related to one or more DPAGEs.

Device-Dependent Input Information (3270 or SLU 2)

Using certain options for inputting information can make the application program device-dependent. Descriptions of the effects of various input options follow.

Cursor Location

As an option of the MFS Language utility, a field in the message can contain the location of the cursor on the device when input was transmitted to IMS. This option is only available for 3270 or SLU 2 display devices and its use can make programs device-dependent. The format of the cursor information is two 2-byte binary numbers, the first containing the line number, the second containing the column

© Copyright IBM Corp. 1974, 2000

number. The minimum value for the line or column is 1. For 3270-An device types, the maximum value for the line is the first parameter of the SIZE= operand; the maximum value for the column is the second parameter of the SIZE= operand.

Table 31 lists the valid line and column values.

Table 31. Maximum Line and Column Values for 3270 Device Types

	Maximum Value	
MFS Device Type	**Line**	**Column**
3270,1	12	40
3270,2	24	80
3270-An		
SIZE=(12,40)	12	40
SIZE=(12,80)	12	80
SIZE=(24,80)	24	80
SIZE=(32,80)	32	80
SIZE=(43,80)	43	80
SIZE=(27,132)	27	132
SIZE=(62,160)	62	160

Selector Pen

Use of the selector light pen can affect input fields in several ways:

- If the ATTR output field option is not used dynamically to create detectable fields, the following occurs:
  - A message field that refers to device fields defined with the attributes DET,STRIP is presented as a device-independent field.
  - The first data byte available for the message field is the byte beyond the designator character in the device field.
  - A message field that references device fields defined with the attributes IDET,STRIP is also presented with device-independent data.
  - The designator character is removed.
  - Data from this field is not presented if no modified fields exist on the device when the field is selected. In this case, the only device information available for the message is the value specified for *literal* on the PEN= operand of the DFLD statement.
- If the ATTR output field option is used dynamically to create detectable fields, then the following occurs:
  - Fields dynamically established as either deferred detectable or immediate detectable do not have designator characters removed from input.
  - If a field altered to immediate detectable is selected when no other fields on the device are modified, no device input data is available for the message.
- If a message field is defined to receive immediate detect selector pen literal data, one of the following occurs:
  - If device input is not the result of an immediate selector pen detect, the field is padded as requested.
  - If device input is the result of an immediate selector pen detect, but at least one other field on the device is modified, one data character of a question mark (?) is presented in the field with the requested padding.

  - If the device input is the result of an immediate selector pen detect and no other modified fields exist on the device, that literal is placed in the message as requested if the detected field is defined with a PEN=*literal*. If the detected field is not defined with a PEN=*literal*, one data byte of a question mark (?) is placed in the message field. In either case, no other device information is provided.
- If an EGCS attribute is defined for a light-pen-detectable field, you should specify ATTR=NOSTRIP on the DFLD statement and design your application program to bypass or remove the two designator characters from the input data. If ATTR=STRIP is specified or defaulted, MFS removes only the first designator character and truncates the last data character in the field.

Magnetic Stripe Reading Devices

The use of magnetic stripe reading devices is transparent to the application program. For operator identification (OID) card readers, the framing characters (SOR, EOR, EOI, LRC) are removed and parity checking is performed before editing.

Program Function Keys

Use of program function keys is transparent to the application program.

Program Access Keys

Program access key information is not available to application programs.

Output Message Formats

MFS edits output segments created by an IMS application program into a device format suitable for the device or remote program for which the message is destined. Normally, the output segments from the IMS program contain no device-related data. All information needed for output to a device or remote program is provided when the message format is defined to the MFS Language utility program. For a remote program with DPM, specific device-dependent information is provided by the remote program without interpretation by MFS.

An output message consists of all segments presented to IMS with an `ISRT` call between a `GU` call to the I/O PCB and either a `PURG` call, another `GU` call to the I/O PCB, or normal program termination.

The format of output messages is defined to the MFS utility just like the format of input messages—one or more segments, each with one or more fields.

```
MESSAGE
    SEGMENTs
        FIELDs
```

Logical Pages

Output segments can be grouped for formatting by defining logical pages (LPAGE statement).

```
MESSAGE
    LPAGEs
        SEGMENTs
            FIELDs
```

When LPAGEs are defined, each LPAGE is related to a specific DPAGE that defines the device format for the logical page. If LPAGEs are not defined, MFS

considers the defined message as one LPAGE, and the rules described below for messages with one LPAGE apply. Table 18 to Table 20 on page 206 illustrate various LPAGE definitions.

When a message has one LPAGE with one segment, each segment inserted by the application program is edited in the same manner.

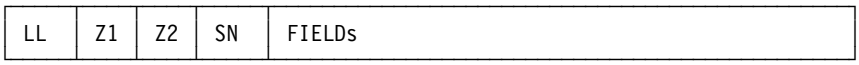
When a message has one LPAGE with multiple segments, message segments must be inserted in the defined sequence. Not all segments in an LPAGE must be presented to IMS, but be careful when segments are omitted. An option 1 or 2 segment can be omitted if all segments to the end of the LPAGE are omitted; otherwise, a null segment must be inserted to indicate segment position. Option 3 output message segments can be omitted but the segments sent must include the segment number identifier.

Multiple series of segments can be presented to IMS as an output message. If the LPAGE is defined as having N segments, segment N+1 is edited as if it were segment 1, unless a segment with the page bit (X'40') in the Z2 field is encountered prior to segment N+1. When multiple series of output segments are presented and segments are omitted, the segment which begins a series must have bit 1 (X'40') of the Z2 field turned on.

When a message has multiple LPAGEs, data in the first segment of a series determines which LPAGE the series belongs to, which determines the editing to be performed on the segments. If the LPAGE to be used cannot be determined from the first segment of a series, the last LPAGE defined is used. Rules for segment omission are the same as those described above. A bit in the Z2 field (X'80') of the message indicates structured data is present in the outbound data stream. An output message using structured data must either define the MODNAME as blanks or binary zeros, or use MFS bypass.

Segment Format

Each output segment has a 4-byte prefix defining the length of the segment and, if required, whether the segment is the first segment of an LPAGE series. Option 3 output messages must contain an additional two bytes identifying the relative segment number within the LPAGE series.



LL This is a 2-byte binary field representing the total length of the message segment, including LL, Z1, and Z2 and if present, SN. The value of LL equals the number of bytes in text (all segment fields) plus 4 (6 if option 3). The application program must fill in this count. If a size limit was defined for output segments of a transaction being processed, LL must not exceed the defined limit.

The segment length must be less than the message queue buffer data size (buffer size—prefix size) specified at IMS system definition. The segment length can be less than the length defined to the MFS Language utility. If a segment is inserted that is larger than the segment defined to the MFS utility, the segment is truncated. No error messages are issued. Fields truncated or omitted are padded as requested in the format definition to the MFS Language utility.

When PL/I is used, the LL field must be defined as a binary fullword. The value provided by the PL/I application program must represent the actual

segment length minus two bytes. For example, if an output message segment is 16 bytes, LL=14 and is the sum of: the length of LL (4 bytes − 2 bytes) + Z1 (1 byte) + Z2 (1 byte) + TEXT (10 bytes).

Z1 This is a 1-byte field containing binary zeros and is reserved for IMS.

Z2 This is a 1-byte field that can be used by the application program for control of various output device functions.

For more information on this field, see *IMS/ESA Administration Guide: Transaction Manager*.

SN For option 3 only. This is a 2-byte binary field containing the relative segment number of the segment within the LPAGE. The first segment is number 1.

A NULL segment can be used to maintain position within a series of option 1 or 2 output segments within an LPAGE. A null segment must be used if segments in the middle of an LPAGE series are to be omitted. If all segments to the end of the LPAGE series are to be omitted, null segments are not required. A null segment contains one data byte (X'3F') and has a length of 5.

Example

An example of coding a null character in COBOL is shown in Figure 56.

```
ID DIVISION.
PROGRAM-ID. SAMPLPGM.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
77  PART1 PIC 9(3) VALUE 123.
77  CUR-NAME PIC 99 COMP VALUE 0.
77  CUR-PART PIC 99 COMP VALUE 0.
01     NULLC.
    02 FILLER PIC 9 COMP-3 VALUE 3.
01  LINE-A.
    02 NAME-1.
       03 NAME-2 OCCURS 30 PIC X.
    02 PARTNUM.
       03 PARTNUM1 OCCURS 10 PIC 9.
PROCEDURE DIVISION.
    MOVE ''ONES' TO NAME-1.
    MOVE 6 TO CUR-NAME.
    MOVE NULLC TO NAME-2 (CUR-NAME).
    MOVE 4 TO CUR-PART.
    MOVE NULLC TO PARTNUM1 (CUR-PART).
```

Figure 56. Coding a Null Character in COBOL

Field Format (Options 1 and 2)

All fields in option 1 and 2 output segments are defined as fixed length and fixed position. The data in the fields can be truncated or omitted by two methods:

- Inserting a short segment
- Placing a NULL character (X'3F') in the field

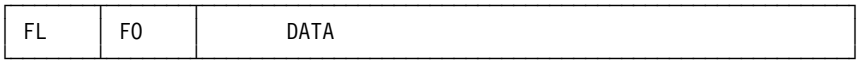
Fields are scanned left to right for a null character. The first null encountered terminates the field. If the first character of a field is a null character, the field is omitted (depending on the fill character used). Positioning of all fields in the segment remains the same regardless of null characters. Fields truncated or omitted are padded as defined to the MFS Language utility.

If ATTR=YES is specified in the MFLD definition, and if X'3F' is the first or second byte of the attribute portion of the field, the field is omitted and the attributes specified on the DFLD statement are used.

For an example of field truncation and omission, see “Output Message Formatting Options” on page 204.

Field Format (Option 3)

Under option 3 output, fields can be placed in their segments in any order and with any length that conforms to the segment size restriction. Short fields or omitted fields are padded as defined to the MFS Language utility. Each field must be preceded by a 4-byte field prefix of the same format provided by MFS for option 3 input fields:



FL The length of the field, including the 4-byte field prefix. FL consists of 2 binary bytes, which require no alignment.

FO The relative offset of the field in the segment, based on the definition of the message to the MFS Language utility. FO consists of 2 binary bytes, which require no alignment. The relative offset of the first field defined in the segment is 4. The relative offset of the second field is 4 plus the length of the first field as defined to the MFS Language utility.

Errors in the contents of FL and FO cause unpredictable results.

Option 3 fields do not need to be in sequence in the output segment, but all fields must be contiguous in the segment; that is, the field prefix of the second field must begin in the byte beyond the first field’s data. Null characters in option 3 fields have no effect on the data transmitted to the device. They are treated as any other nongraphic characters; that is, replaced with blanks.

Device control characters are invalid in output message fields. For 3270 display and SLU 2 terminals, the control characters HT, CR, LF, NL, and BS are changed to null characters. For all other devices, these control characters are changed to blanks. All other nongraphic characters (X'00' through X'3F', and X'FF') are changed to blanks before transmission. For DPM devices, control characters are permitted if GRAPHIC=NO has been specified.

Examples of field formats are shown under “Output Message Formatting Options” on page 204.

Device-Dependent Output Information

Using certain options for outputting information can make the application program device-dependent. Some options allow the application program to control certain features of devices receiving output. Descriptions of the effects of various output options follow.

System Control Area (SCA)

An option of the MFS Language utility allows the creation of an SCA field in the first segment of a message or, if LPAGEs are defined, in the first segment of any or all LPAGEs. This field allows application program control of specific device features when the features apply to the device for which the message is destined. The first 2 bytes of the SCA field are defined as shown in Table 32 on page 279 and Table 33 on page 279

on page 277.

Table 32. Valid Bytes and Bits for TYPE=3270, SLU 2, DPM-An, or DPM-Bn

Byte	Bit	Description
0	0-7	Should be 0.
1	0	Should be 1.
	1	Force format write (erase device buffer and write all required data).
	2	Erase unprotected fields before write.
	3	Sound device alarm.
	4	Copy output to candidate printer.
	5	B'0'—For 3270, protect the screen when output is sent. For DPM, demand paging can be performed.
		B'1'—For 3270, do not protect the screen when output is sent. For DPM-B, autopaging can be performed.
	6	For the partition formatted 3290: B'0'—do not erase existing partitions. B'1'—erase all partitions before sending message. For others, should be 0.
	7	Should be 0.

Notes:

1. For the 3290 in partition format mode, the DOF on the current message is checked to see if it is the same DOF used last. If it is, bit 6 in the SCA and DSCA operands is checked for the erase/do not erase partitions option before the output message is sent.
2. The default for bit 6 is B'0', “do not erase”. If this bit is not specified, the output is sent according to the partition paging option specified, and partitions that do not receive output remain unchanged.
3. If bit 6 is set to B'1', then existing partitions will be erased and the output is sent according to the partition paging option specified. See “Partition Initialization Options and Paging” on page 244 for more information.
4. The SCA bit settings are “OR’d” to the DSCA bit settings. For example, if byte 1 bit 5 in the DSCA for DPM-B is set to B'0' in the DSCA for DPM-B, the application program can request autopaged output by setting the SCA value to B'1'. (This request is honored only if present in the first segment of the first LPAGE of the output message.)
5. SCA information is sent to the remote program or ISC subsystem in a DFLD identified by the parameter SCA (see “Chapter 6. Introduction to MFS” on page 169). Any invalid bits for the device type are reset. The valid bits are used.

Table 33. Valid Bytes and Bits for TYPE=FIDS, FIDS3, FIDS4, FIDS7, FIJP, FIPB, or FIFP

Byte	Bit	Description
0	0-7	Should be 0.
1	0	Should be 1.
	1-2	Not applicable for FIN output devices.
	3	Set “device alarm” in output message header.
	4	Not applicable for FIN output devices.
	5-7	Should be 0.

Table 33. Valid Bytes and Bits for TYPE=FIDS, FIDS3, FIDS4, FIDS7, FIJP, FIPB, or FIFP (continued)

Byte	Bit	Description

Notes:

1. Bits 1, 2, and 4 function only for 3270 and are not applicable to finance workstations. If set on by the program, and the message is edited for a finance workstation, they are ignored.
2. For TYPE=274X, SCS1, or SCS2, the SCA parameter is ignored.
3. For TYPE=3270P, all bits except “set device alarm” are ignored.

Cursor Location

An application program can set the cursor location on the screen either by setting a cursor attribute for a field or by using a special cursor positioning field in the output message.

Recommendation: Use the cursor attribute method because the application program does not need to know the position of fields on a device.

Cursor positioning using the cursor attribute method is described in “Dynamic Attribute Modification” on page 281.

Using an option of the MFS Language utility, you can define a field in an output segment to allow the application program to request cursor positioning to a specific line and column on the device. Depending on the device output format used, there can be one or more such fields per LPAGE. If the field contains an invalid number it is ignored and the cursor is positioned as requested in the device output format.

The cursor field should contain two 2-byte binary numbers (no alignment required), the first containing the line number, the second containing the column number. The minimum value for the line or column is 1. For 3270-An device types, the maximum value for the line is the first parameter of the SIZE= operand; the maximum value for the column is the second parameter of the SIZE= operand. Table 34 lists the valid line and column values.

Table 34. Maximum Line and Column Values for MFS Device Types

	Maximum Value	
MFS Device Type	**Line**	**Column**
FIDS (240 characters)	6	40
FIDS3 (480 characters)	12	40
FIDS4 (1024 characters)	16	64
FIDS7 (1920 characters)	24	80
3270,1 (480 characters)	12	40
3270,2 (1920 characters)	24	80
3270-An		
SIZE=(12,40) (480 characters)	12	40
SIZE=(12,80) (960 characters)	12	80
SIZE=(24,80) (1920 characters)	24	80
SIZE=(32,80) (2560 characters)	32	80
SIZE=(43,80) (3440 characters)	43	80

Table 34. Maximum Line and Column Values for MFS Device Types (continued)

MFS Device Type	Maximum Value	
	Line	Column
SIZE=(27,132) (3564 characters)	27	132
SIZE=(62,160) (9920 characters)	62	160

Dynamic Attribute Modification

An option of the MFS Language utility allows an IMS application program to dynamically modify, replace, or simulate the attributes of a device field. This dynamic attribute modification is requested in an output message definition by specifying ATTR=YES in an MFLD statement. MFS then reserves the first two data bytes of the output message field for attribute definition. Errors detected in the data of the 2-byte specification or X'3F' in the first or second attribute byte produce the results shown in Table 35.

Table 35. Results of Data Errors

Message data from application (first 3 bytes)	Output message to device			
	Video, attribute source	Nonvideo, attribute used		
		SIM	NOSIM2	User data
3Fxxxx	DFLD	b	0000	Fill character used
xx3Fxx	DFLD	b	xx00	Fill character used
xxxx3F	Message	y	xxxx	Fill character used
IIII3F	DFLD	b	0000	Fill character used
xxxxxx	Message	y	xxxx	Sent
IIIIxx	DFLD	b	0000	Sent

Notes:

b	Blank
xx	Valid attributes or data
y	Simulated attribute
II	Invalid attribute

Attributes are always sent, even if no data is sent.

When dynamic attribute modification is specified for a device field with predefined attributes, an attribute is sent to the device for that field in every output operation, even if the data for this device field is not included in the output message. These attributes are used in the following ways:

- If the output message field has an attribute and the attribute is valid, then the dynamic attribute modification is performed.
- If the message field is not included in the LPAGE being used or the attribute is not valid, the predefined attribute for the device field is used.

When attribute simulation is defined, the first byte of the device field is reserved for attribute data. The following attributes can be simulated:

- Cursor position (3604 display only)

- Nondisplayable
- High-intensity displayable
- Modified attributes

The two attribute bytes are defined in Table 36.

Table 36. Definitions of the Two Attribute Bytes

Byte	Bit	Definition
0	0-1	If both bits are on, requests that the cursor be placed on the first position of this field on the device. The first cursor-positioning request encountered in an LPAGE series (first MFLD with cursor attribute or cursor line/column value) that applies to a physical page is honored; these bits must be 00 or 11.
	2-7	Must be off.
1	0	Must be on.
	1	1. If on, these attribute specifications are to replace the attribute byte defined for the field. 2. If off, these attribute specifications are to be added to the attribute byte defined for the field logical “OR” operation. A zero in a bit position indicates that the defined attribute is to be used (that is, if bit 2 is 0 then the field will be protected or unprotected depending on the DFLD definition. A 1 in a bit position indicates that the corresponding attribute is to be used (that is, if bit 3 is 1 then the field will have the numeric attribute.)
	2	Protected
	3	Numeric
	4	High-intensity (forces detectable and displayable); if simulated, an * appears in the first byte of the device field.
	5	Nondisplayable (forces nondetectable); if simulated, no data is sent regardless of other attributes.
	6	Detectable (forces normal intensity).
	7	Premodified; if simulated, an underscore (_) appears in the first byte of the device field.

Notes:

1. Bits 4, 5, and 6 are incompatible. If more than one is set, bit 4 takes precedence over bits 5 and 6. Bit 5 takes precedence over bit 6.
2. If both bits 4 and 7 are simulated, an ! appears in the first byte of the device field.

Dynamic modification of attributes to detectable requires other action by the IMS application program to make the device function properly. Detectable fields must have a designator character and certain padding characters.

For DPM, field attribute information can be passed from the IMS application program to the remote program, but cannot be specified, unless ATTR=(YES,*nn*) appears in the MFS DFLD definitions.

See the appropriate component description manual to determine which extended attributes are available to a given terminal type.

Dynamic Modification of Extended Field Attributes

For an application program to modify extended attribute data, the MFLD statement must specify ATTR=*nn*. Any error causes the DFLD EATTR= specification for that extended attribute byte to be used.

For modification of the extended attributes, two additional bytes per attribute must be reserved. The values that can be specified in these extended attribute modification bytes and the resulting values that are used are:

Specification	Value Used
X'00'	Device default
Valid value	Your specification
Invalid or omitted	From EATTR= on DFLD statement
Duplicate	Last (rightmost) specification

During online execution, if ATTR=PROT is specified as a dynamic modification, any field validation attributes defined on the DFLD statement or specified as a dynamic modification are reset.

Restriction: Trigger fields are not supported by MFS.

Table 37 shows the format of the extended attribute modification bytes.

Table 37. Format of Extended Attribute Modification Bytes

ATTR 1 type	ATTR 1 value	ATTR 2 type	ATTR 2 value	ATTR n type	ATTR n value
	1	2	3	2xn_2	2xn_1

Types

Hexadecimal specifications:

01 Validation replacement

02 Validation addition

03 Field outlining replacement

04 Field outlining addition

05 Input control replacement

06 Input control addition

Field outlining applies to 3270 display devices, and to printers defined as 3270P or SCS1 that support the 3270 Structured Field and Attribute Processing option, and support the Extended Graphics Character Set (EGCS).

Character specifications (the letter C indicates character):

C1 Highlighting

C2 Color

C3 Programmed Symbols

Values

Field validation in hexadecimal:

Bit **Meaning**

0 to 4	Reserved
5	Mandatory fill
6	Mandatory field
7	Reserved

For field highlighting as shown below:

Character	**Meaning**
X'00'	Device default
X'F1'	Blink
X'F2'	Reverse video
X'F4'	Underline

Field color (seven-color models only):

Character	**Meaning**
X'00'	Device default
X'F1'	Blue
X'F2'	Red
X'F3'	Pink
X'F4'	Green
X'F5'	Turquoise
X'F6'	Yellow
X'F7'	Neutral

Field outlining in hexadecimal:

Bit	**Meaning**
0 to 3	Reserved
4	Left line
5	Over line
6	Right line
7	Under line
X'00'	Default (no outline)

Input control (of DBCS/EBCDIC mixed fields) in hexadecimal:

Bit	**Meaning**
0 to 6	Reserved
7	SO/SI creation
X'00'	Default (no SO/SI creation)

For the programmed symbols, valid local ID values are in the range X'40'—X'FE', or X'00' for the device default.

Ways to specify the binary validation attribute type and value in COBOL are shown in Figure 57.

```
VAL_REP_MFILL  PIC 9(3)  COMP VALUE 260   (replace-mandatory fill)
*
VAL_REP_MFLD   PIC 9(3)  COMP VALUE 258   (replace-mandatory field)
*
VAL_ADD_MFILL  PIC 9(3)  COMP VALUE 516   (add-mandatory fill)
*
VAL_ADD_MFLD   PIC 9(3)  COMP VALUE 514   (add-mandatory field)
*
```

Figure 57. Binary Validation Attribute Type and Value Specification in COBOL

Ways to specify field outlining attributes, input control types, and values in COBOL are shown in Figure 58.

```
 01  BINVALUE.
     02  VAL0000                PIC S999 COMP   VALUE   +0.
     02  VAL0000X    REDEFINES  VAL0000.
         03  FILLER             PIC X.
         03  VAL00              PIC X.
*                                      (NO FIELD OUTLINE)
```

Figure 58. Various Ways to Specify Field Outlining (Part 1 of 16)

```
     02  VAL0001                PIC S999 COMP   VALUE   +1.
     02  VAL0001X    REDEFINES  VAL0001.
         03  FILLER             PIC X.
         03  VAL01              PIC X.
*                                      (UNDERLINE)
```

Figure 58. Various Ways to Specify Field Outlining (Part 2 of 16)

```
     02  VAL0002                PIC S999 COMP   VALUE   +2.
     02  VAL0002X    REDEFINES  VAL0002.
         03  FILLER             PIC X.
         03  VAL02              PIC X.
*                                      (RIGHTLINE)
```

Figure 58. Various Ways to Specify Field Outlining (Part 3 of 16)

```
     02  VAL0003                PIC S999 COMP   VALUE   +3.
     02  VAL0003X    REDEFINES  VAL0003.
         03  FILLER             PIC X.
         03  VAL03              PIC X.
*                                      (RIGHTLINE & UNDERLINE)
```

Figure 58. Various Ways to Specify Field Outlining (Part 4 of 16)

```
           02  VAL0004                PIC S999 COMP   VALUE   +4.
           02  VAL0004X    REDEFINES  VAL0004.
               03  FILLER              PIC X.
               03  VAL04               PIC X.
      *                                      (OVERLINE)
```

Figure 58. Various Ways to Specify Field Outlining (Part 5 of 16)

```
           02  VAL0005                PIC S999 COMP   VALUE   +5.
           02  VAL0005X    REDEFINES  VAL0005.
               03  FILLER              PIC X.
               03  VAL05               PIC X.
      *                                      (OVERLINE & UNDERLINE)
```

Figure 58. Various Ways to Specify Field Outlining (Part 6 of 16)

```
           02  VAL0006                PIC S999 COMP   VALUE   +6.
           02  VAL0006X    REDEFINES  VAL0006.
               03  FILLER              PIC X.
               03  VAL06               PIC X.
      *                                      (OVERLINE & RIGHTLINE)
```

Figure 58. Various Ways to Specify Field Outlining (Part 7 of 16)

```
           02  VAL0007                PIC S999 COMP   VALUE   +7.
           02  VAL0007X    REDEFINES  VAL0007.
               03  FILLER              PIC X.
               03  VAL07               PIC X.
      *                                      (OVERLINE & RIGHTLINE
      *                                       & UNDERLINE)
```

Figure 58. Various Ways to Specify Field Outlining (Part 8 of 16)

```
           02  VAL0008                PIC S999 COMP   VALUE   +8.
           02  VAL0008X    REDEFINES  VAL0008.
               03  FILLER              PIC X.
               03  VAL08               PIC X.
      *                                      (LEFTLINE)
```

Figure 58. Various Ways to Specify Field Outlining (Part 9 of 16)

```
           02  VAL0009                PIC S999 COMP   VALUE   +9.
           02  VAL0009X    REDEFINES  VAL0009.
               03  FILLER              PIC X.
               03  VAL09               PIC X.
      *                                      (LEFTLINE & UNDERLINE)
```

Figure 58. Various Ways to Specify Field Outlining (Part 10 of 16)

```
           02  VAL000A                PIC S999 COMP   VALUE  +10.
           02  VAL000AX    REDEFINES  VAL000A.
               03  FILLER              PIC X.
               03  VAL0A               PIC X.
      *                                      (LEFTLINE & RIGHTLINE)
```

Figure 58. Various Ways to Specify Field Outlining (Part 11 of 16)

```
           02  VAL000B                PIC S999 COMP   VALUE  +11.
           02  VAL000BX    REDEFINES  VAL000B.
               03  FILLER              PIC X.
               03  VAL0B               PIC X.
      *                                      (LEFTLINE & RIGHTLINE
      *                                       & UNDERLINE)
```

Figure 58. Various Ways to Specify Field Outlining (Part 12 of 16)

```
           02  VAL000C                PIC S999 COMP   VALUE  +12.
           02  VAL000CX    REDEFINES  VAL000C.
               03  FILLER              PIC X.
               03  VAL0C               PIC X.
      *                                      (LEFTLINE & OVERLINE)
```

Figure 58. Various Ways to Specify Field Outlining (Part 13 of 16)

```
           02  VAL000D                PIC S999 COMP   VALUE  +13.
           02  VAL000DX    REDEFINES  VAL000D.
               03  FILLER              PIC X.
               03  VAL0D               PIC X.
      *                                      (LEFTLINE & OVERLINE
      *                                       & UNDERLINE)
```

Figure 58. Various Ways to Specify Field Outlining (Part 14 of 16)

```
           02  VAL000E                PIC S999 COMP   VALUE  +14.
           02  VAL000EX    REDEFINES  VAL000E.
               03  FILLER              PIC X.
               03  VAL0E               PIC X.
      *                                      (LEFTLINE & OVERLINE
      *                                       & RIGHTLINE)
```

Figure 58. Various Ways to Specify Field Outlining (Part 15 of 16)

```
           02  VAL000F                PIC S999 COMP   VALUE  +15.
           02  VAL000FX    REDEFINES  VAL000F.
               03  FILLER              PIC X.
               03  VAL0F               PIC X.
      *                                      (BOX)
```

Figure 58. Various Ways to Specify Field Outlining (Part 16 of 16)

Examples: The following examples show the use of the EATTR= and ATTR=(,*nn*) operands:

```
AX     DFLD  EATTR=(VMFILL,HUL),ATTR=(NUM,HI)
AY     MFLD  AX,ATTR=(,2)
```

The EATTR= operand of the DFLD statement requests that the specified field must be completely filled with data, high intensity, and underlined. The ATTR= operand of the DFLD statement requests that the specified field be numeric and high intensity.

Specifying the ATTR=(,2) operand indicates the application program can dynamically modify the two extended attributes specified in the EATTR= operand. If this is specified, the LTH= value on the MFLD statement must be increased by 4 bytes for the modified attribute bytes. The application program can dynamically modify the validation and the extended highlighting attributes. The extended attributes of color and programmed symbols cannot be dynamically modified, because they were not specified in the EATTR= operand. The existing 3270 attributes cannot be dynamically modified, because ATTR=YES was not specified on the MFLD statement.

To dynamically modify the extended highlighting to blinking, and add mandatory field validation when data is entered into the field, the information shown in Table 38 must be placed in the field referenced by the MFLD “AY” in the preceding example.

Table 38. Extended Attribute Types and Values for COBOL

ATTR 1 type	ATTR 1 value	ATTR 2 type	ATTR 2 value	Field data
C1	F1	02	02	data
0	1	2	3	4–n

Specification of color and programmed symbols, if present, is ignored. Regardless of the number of attribute modification bytes specified, MFS sends the number of extended attributes specified in the EATTR=operand of the DFLD.

Because the validation addition type (X'02') is specified, rather than the validation replacement type (X'01'), the change to the validation attribute byte is an addition rather than a replacement.

```
BX    DFLD    EATTR=(CD,HD,PC'Z'),ATTR=(PROT)
BY    MFLD    BX,ATTR=(YES,3)
```

The EATTR= operand of the DFLD statement requests a field with a programmed symbol buffer local ID of “Z” and the protected attribute. If no dynamic modification by an IMS application program occurs, the color and highlighting device defaults are used. Because of the specification of ATTR=(YES,3) in this example, the color, extended highlighting, programmed symbol buffer local ID, and existing 3270 attributes can be dynamically modified.

You can dynamically modify the color, extended highlighting, and the 3270 attribute bytes, while keeping the programmed symbol local ID (PC'Z') as specified on the DFLD statement. For example, to dynamically modify the color to pink, the extended highlighting to reverse video, and the 3270 attribute bytes to numeric and unprotected, use the attribute modification bytes for fields referenced by MFLD “BY” as shown in Table 39 on page 289.

Table 39. Example of Dynamically Modified Attribute Bytes

Existing 3270 ATTR mods		ATTR 1 type	ATTR 1 value	ATTR 2 type	ATTR 2 value	ATTR 3 type	ATTR 3 value	Field data
00	D0	C2	F3	C1	F2	40	40	data
0	1	2	3	4	5	6	7	8–n

With byte 1, bit 1 of the existing 3270 attribute modification bytes on, IMS replaces the existing 3270 attribute byte rather than adding to it. This changes the field to unprotected and specifies the numeric attribute. The third attribute has a type of X'40' (an invalid type) specified, which causes IMS to use the DFLD specification for programmed symbols.

Dynamic Modification of EGCS Data

EGCS data can also be dynamically modified to permit EBCDIC or EGCS data to be mapped to a particular field on the 3270 display. With this function:

- You can enter EBCDIC or EGCS data.
- The application program can receive EBCDIC or EGCS data.
- EBCDIC or EGCS data can be passed to an SLU P remote program or to an ISC subsystem.

If ATTR=(,*nn*) is specified in the MFLD statement and a programmed symbol attribute is specified in the corresponding DFLD statement, the application program can modify the field programmed symbol attribute. Dynamic modification of the programmed symbol attribute for EGCS requires two additional bytes. These additional bytes precede the MFLD data and must be included in the MFLD LTH= specification.

The IMS application program can modify the DFLD programmed symbol attribute if all the following conditions are met:

- The DFLD specifies EATTR=PX'hh', PC'c', EGCS'hh' or EGCS.
- The corresponding MFLD statement specifies ATTR=(,*nn*), where *nn* is a value from 1 through 4.
- The application program includes 2 × *nn* additional bytes preceding the data field.
- One set of two attribute bytes has an X'C3' as its first byte and a valid value (X'00' or X'40'—X'FE') as its second byte.

Table 40 illustrates what MFS transmits in the value byte of the programmed symbol attribute type, if the DFLD statement does or does not specify a programmed symbol attribute, and the IMS application program does or does not modify it.

Table 40. Attribute Type Value Byte Contents

Application Program Programmed Symbol Attribute Bytes of X and:	C3 EATTR=	ATTR=	EATTR=
	Programmed symbol specified	Programmed symbol default	Not specified
X'40_FE'[1]	Send X'40_FE'	Send X'40_FE'	Send no attribute
Default X'00'[1]	Send X'00'	Send X'00'	Send no attribute

Table 40. Attribute Type Value Byte Contents (continued)

Application Program Programmed Symbol Attribute Bytes of X and:	C3 EATTR=	ATTR=	EATTR=
Not specified [2]	Send programmed symbol DFLD specification	Send no attribute	N/A
Omitted or Invalid [3]	Send programmed symbol DFLD specification	Send X'00'	Send no attribute

Notes:

1. ATTR=*nn* is specified on at least one MFLD statement that maps to this DFLD statement. The IMS application program specifies a programmed symbol attribute of X'40' to X'FE'.
2. ATTR=*nn* is not specified on any MFLD statement that maps to this DFLD statement.
3. ATTR=*nn* is specified on at least one MFLD statement that maps to this DFLD statement. The application program omits specifying this attribute, or the specified attribute is not X'00' or X'40' to X'FE'.

Dynamic Modification of DBCS/EBCDIC Mixed Data

Programmed symbols and input control attribute bytes can be dynamically modified to permit EBCDIC or EGCS data to be mapped to a particular field on the 3270 display. DBCS/EBCDIC mixed data can also be dynamically modified. DBCS is a subset of EGCS, so the EGCS field can contain DBCS data, as shown in Figure 59.

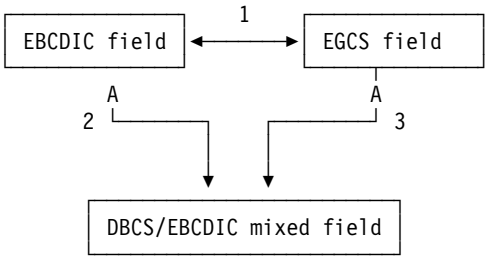


Figure 59. Dynamic Modification of a DBCS/EBCDIC Mixed Field

The IMS application program can make a field EBCDIC, EGCS, or DBCS/EBCDIC mixed when all of the following conditions are satisfied:

- One of the following is specified on the DFLD statement:

```
EATTR=(EGCS,MIXD)
EATTR=(EGCS'00',MIX)
EATTR=(EGCS'00',MIXD)
```

  A DBCS keyword does not exist; DBCS fields are specified using the EGCS keyword. The initial attribute must specify an EGCS field, a DBCS/EBCDIC mixed field, or an EBCDIC field.
- The corresponding MFLD statement specifies ATTR=(,*nn*) where *nn* is 2 or greater.
- The application program contains 2 × *nn* additional bytes preceding the data field.

When *nn*=2, the initial attribute is changed as shown in Table 41 according to the value of the two attribute byte sets (4 bytes) specified in front of the data field by the application program.

Table 41. Dynamic Modification of a DBCS/EBCDIC Mixed Field

Attribute Byte	EBCDIC	EGCS	Mixed
40404040	EBCDIC	EGCS	Mixed
05014040	Mixed	Mixed	Mixed
0501C3F8	EGCS	EGCS	EGCS
C3F84040	EGCS	EGCS	EGCS
C3F80501	Mixed	Mixed	Mixed
0500C3F8	EGCS	EGCS	EGCS
C3000501	Mixed	Mixed	Mixed
C3000500	EBCDIC	EBCDIC	EBCDIC

When the initial attribute specifies an EGCS field and the application program specifies dynamic modification of the input control attribute to a DBCS/EBCDIC mixed field, MFS replaces the value of the programmed symbol for which the EGCS field is specified with the device default. For more information, refer to “Dynamic Modification of Extended Field Attributes” on page 283.

Specification of Message Output Descriptor Name

Output messages destined for MFS terminals are formatted using a message output descriptor (MOD). Which MOD IMS uses can be specified within the output call, either insert (ISRT) or purge (PURG). Both `ISRT` and `PURG` allow you to specify an output MOD name parameter on the call that provides a segment of an output message.

When the output MOD name parameter is specified, IMS uses the name supplied to select the message output descriptor. If the call is directed to the I/O PCB or alternate response PCB, IMS updates the MESSAGE OUTPUT DESCRIPTOR NAME field of the I/O PCB with the name supplied in the output call. The MOD name of all output messages inserted on an alternate PCB that did not explicitly specify a MOD name is set to the previous MOD name.

Which MOD IMS uses to format the message depends on the name specified:

Name Specified	Descriptor Used
Valid output MOD name	Message output descriptor named by output MOD name
Eight blanks	IMS default message output descriptor (3270 or SLU 2 only—other devices use IMS basic edit for output)
Invalid output MOD name	IMS error default message output descriptor

If the output MOD name parameter is not specified, IMS formats the message using the MOD named in the MESSAGE OUTPUT DESCRIPTOR NAME field of the I/O PCB.

MFS Bypass for the 3270 or SLU 2

IMS MFS allows the IMS application program to bypass MFS formatting of input and output messages. With this option, the IMS application program can load programmed symbol buffers, or send a device-dependent data stream to format and update the 3270 display, or write a message to a 3270 printer. The bypass can be used only on the SLU 2, and 3270 devices (except the 3275 dial-up BTAM terminal). Optionally, the IMS application program can examine an input message with the attention identification (AID) byte, cursor address, SBA orders, and buffer addresses as received from the display. For BTAM and non-SNA VTAM transmissions, the data to be sent must be equal to or less than the value specified in the system definition OUTBUF parameter. Data sent to a printer using the MFS bypass is restricted to 4 KB.

MFS recognizes two special message output descriptor (MOD) names: DFS.EDT and DFS.EDTN.

Output messages bypass MFS formatting only if DFS.EDT or DFS.EDTN is supplied as the MOD name parameter of the application program CALL statement (for more information, see “Specifying Input Forms for MFS Bypass”). IMS system messages, IMS error messages, application program messages with no MOD name, and message switches are always formatted by MFS (using the IMS-supplied formats).

When MFS is bypassed on output, the application program is responsible for constructing the entire 3270 data stream, beginning with the command code and ending with the last data byte. An exception to this could be 3270 output using the MFS bypass and destined to a printer. The following table shows the hexadecimal EBCDIC command codes for use with the 3271/3274 controllers:

Command	3271/3274
Erase All Unprotected	6F
Erase/Write	F5
Erase/Write Alternate	7E
Read Buffer	F2
Read Modified	F6
Read Modified All	6E
Write	F1
Write Structured Field	F3

The user-written application program has two ways to send output to printers:

- By providing the command code and WCC character in the application program and by setting bit 0 to 1 (X'80') in the Z2 field of the message segment to show that the appropriate command is provided.
- By allowing IMS to provide the command code and other characters. However, to print less than the maximum line length, insert new line (NL) characters at the appropriate places in the data stream. This method is the default.

Specifying Input Forms for MFS Bypass

After using the MFS bypass, the IMS application program must accept the input in one of two forms depending on the MOD name specified for the output message:

- MODNAME=DFS.EDT edits the input data.

- MODNAME=DFS.EDTN performs no editing on the input data.

MODNAME=DFS.EDT: The AID and the cursor address are removed from the data stream and any SBA or start field sequences are replaced with blanks. In addition, the basic input edit routine performs the editing. If the AID code received is a CLEAR, PA2, PA3, PFK12, or selector pen attention, existing IMS functions are performed. If a PA1 is received, IMS performs the same function as for PA2 (that is, the next output message is sent if one is available).

MODNAME=DFS.EDTN: If the transaction is in conversational mode, all input is passed to the application as received from the terminal. If the transaction is not in conversational mode, the transaction code must be positioned to precede the AID character of the data stream received from the terminal.

The password should never be passed to the IMS application program. The basic editing functions are performed on the destination and password fields only. If the password appears within parentheses immediately after the transaction code, basic edit removes the password. No editing is performed on the remainder of the data. Existing IMS functions are bypassed for AID codes resulting from a CLEAR, PA1, PA2, PA3, or selector pen attention. PFK12 causes a copy to be performed if it is allowed.

Position the transaction code using the physical terminal input edit exit, or cause IMS to supply it using the conversational or preset destination mode.

If the terminal is in conversational mode, the message is sent to the application program in the conversation. If the terminal is in preset mode, the transaction code is added to the beginning of the message and the message is sent to the destination established by the `/SET` command. Therefore, while in preset mode, a slash (/) as the first character of the input data is not considered an IMS command. To be recognized as a command, `/RESET` must immediately follow the cursor address in the input data stream. To do this, enter the `/RESET` command from an unformatted screen (no fields defined for the screen). If the screen is formatted (fields defined for the screen), press the clear key to unformat the screen. However, an application program must receive the clear AID byte and write a data stream that does not format the screen.

Example:

```
Data stream = F5C3, erases the 3270 buffer.
Data stream = F5C3114040, erases the 3275 buffer.
              Entering: The /RESET command
              resets preset mode.
```

If `/RESET` is received from an unformatted screen, while bypassing MFS and basic edit (MOD name is DFS.EDTN) and in preset mode, the input is treated as a command, and the terminal is taken out of preset mode. You are responsible for sending a data stream that leaves the screen unformatted.

If the transaction code and password (if required) are entered with the input message and the terminal is not in conversational or preset mode, your physical terminal input edit exit routine must be included in the IMS system definition. The physical terminal input edit routine gains control before IMS destination and security checking and must modify the input to place the transaction code and password (if required) in front of the AID code.

If the OPTIONS keyword of the IMS system definition TERMINAL or TYPE macro specifies that the keyboard is to remain locked, and the MFS bypass with MOD name DFS.EDTN is used, the application program must assume responsibility for unlocking the 3270 keyboard and resetting the MDT flags.

After use of the MFS bypass, the next output message is formatted by MFS if the MOD name is not supplied or the MOD name supplied is not DFS.EDT or DFS.EDTN.

MFS bypass is intended primarily for subsystems executing under IMS and is not recommended for normal application usage. If IMS application programs deal with 3270 data streams, they become device-dependent, which complicates the application development process.

When a read command is executing in the MFS bypass, the output message containing the read command is dequeued or re-enqueued when the input is received, depending on the option (PAGDEL/NPGDEL) specified on the TERMINAL macro during system definition.

MFS Bypass for the SLU 2 (3290) with Partitioning

When the MOD specified in an application is either DFS.EDT or DFS.EDTN, the output message generated can cause an SLU 2 terminal to function in partitioned mode. Using DFS.EDTN, a conversational application can send a Query and receive a Query reply.

For output, the application program must supply the Create Partition data stream within the output message, along with the data for the partitions. Also, the SLU 2 Device-Dependent Module sets Change Direction (CD) on non-last conversational output messages. This allows Reads and Queries to be sent in Write Structured Fields data streams.

A Query Reply input can be processed only if the previous MOD specified is DFS.EDTN. A Query Reply input can be received but does not have a transaction code in the data stream.

For partitions 01 through 0F, the X'88' byte is followed by a 2-byte field that is not used. If a X'80' byte follows this field, then the next byte is the PID byte (X'01' through X'0F'). For partition 00, the input will have the same format as input data from a non-partitioned SLU 2.

For input with DFS.EDT or DFS.EDTN, the first AID byte, X'88', causes the proper decoding of the second AID byte. Depending on the second AID byte, one of the following occurs:

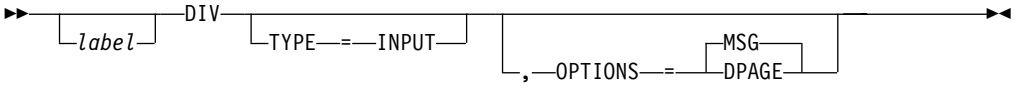
- If the second AID byte decoded is X'80', a third AID byte is decoded. The data stream following that AID byte is passed to the application program as follows:
  - Using basic edit, if DFS.EDT is specified
  - As a complete data stream, if DFS.EDTN is specified
- If the second AID byte is not X'80', input is passed only if the MOD specified in the application is DFS.EDTN. When DFS.EDTN is specified, the complete data stream starting with the X'88' AID byte is passed to the application program.

DIV Statement

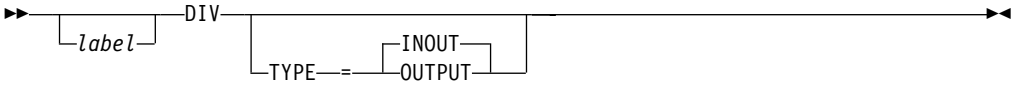
The DIV statement defines device formats within a DIF or DOF. The formats are identified as input, output, or both input and output, and can consist of multiple physical pages. For DEV TYPE=274X, SCS1, SCS2, or DPM-AN, two DIV

statements can be defined: DIV TYPE=OUTPUT and DIV TYPE=INPUT. For all other device types, only one DIV statement per DEV is allowed.

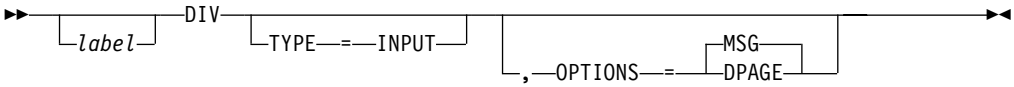
Format for DEV TYPE=274X, SCS1, or SCS2 and DIV TYPE=INPUT:



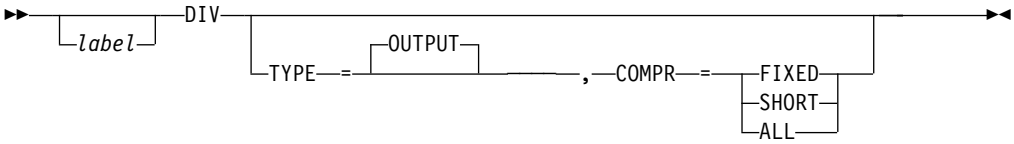
Format for DEV TYPE=3270 or 3270-An:



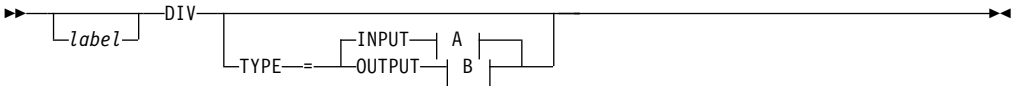
Format for DEV TYPE=FIN:



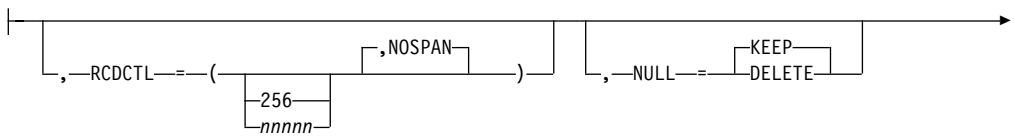
Format for DEV TYPE=274X, SCS1, SCS2, 3270P, FIDS, FIDS3, FIDS4, FIDS7, FIJP, FIPB, or FIFP and DIV TYPE=OUTPUT:

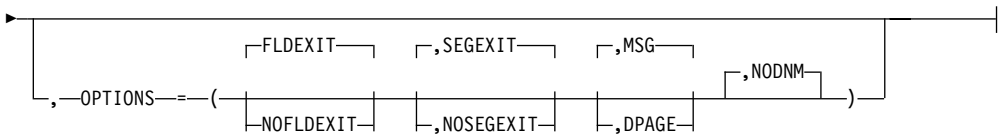


Format for DEV TYPE=DPM-An:

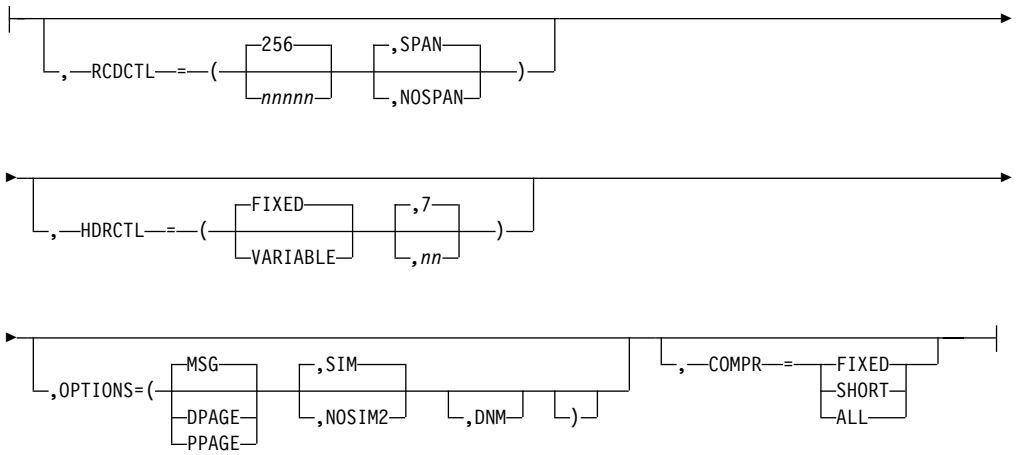


A:

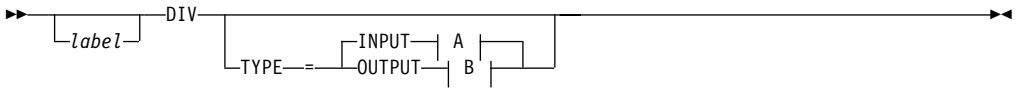




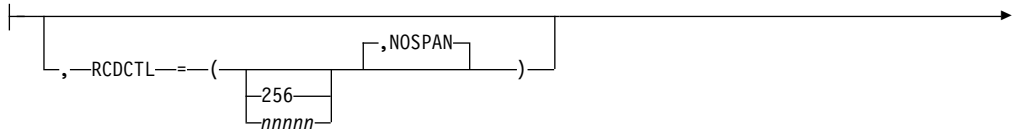
B:

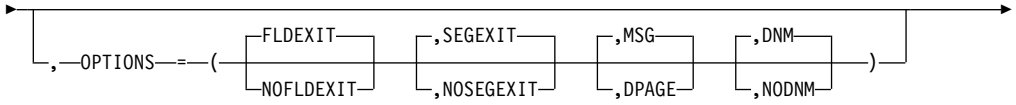


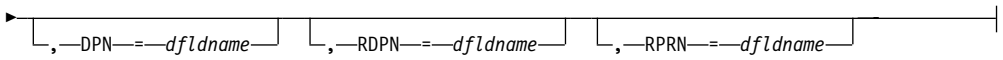
Format for DEV TYPE=DPM-Bn:



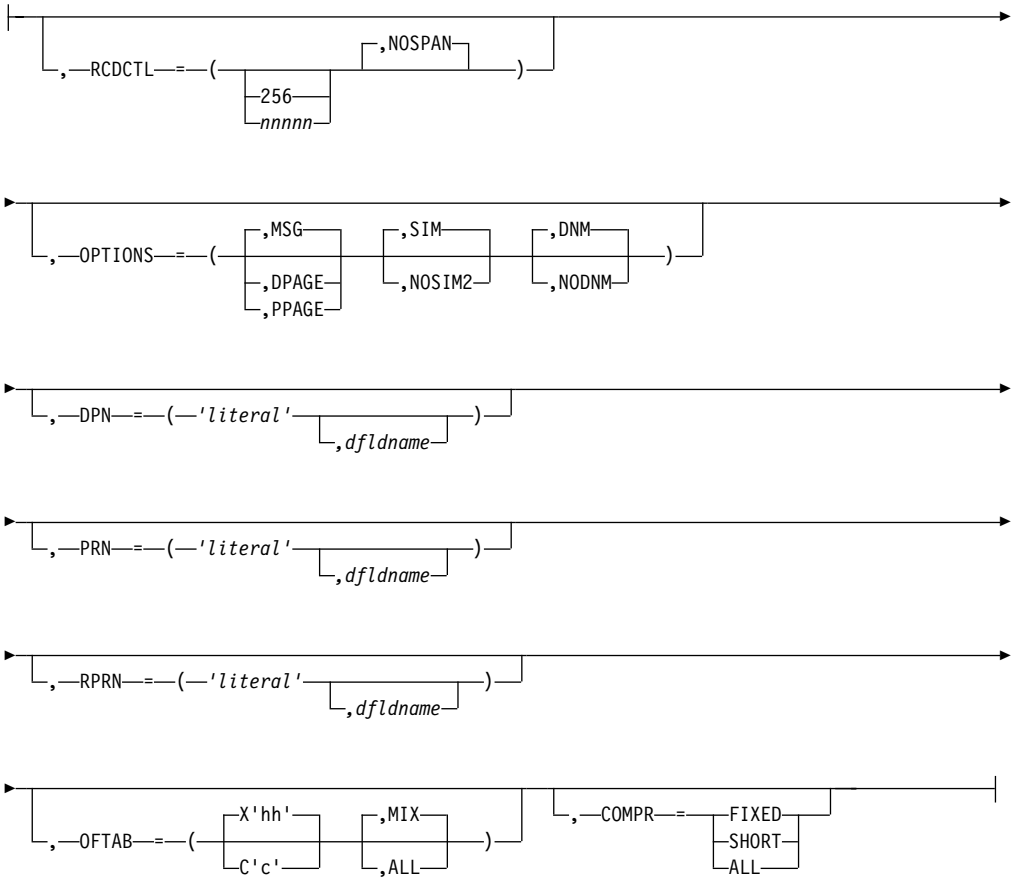
A:







B:



Parameters:

label
A one- to eight-character alphanumeric name that is specified to uniquely identify this statement.

TYPE=
This describes the format as input, output, or both.

INOUT
Describes an input and output format.

INPUT|OUTPUT
Describes an input-only format (INPUT) or an output-only format (OUTPUT). Certain DEV statement keywords can be used. For example:

- Specifying WIDTH=80 for DEV TYPE=SCS1 indicates that fields can be printed in columns 1 through 80 on output and received from columns 1 through 80 on input.
- Specifying WIDTH=80 for DEV TYPE=SCS2 indicates that both the card reader and card punch have the same number of punch positions.
- Specifying WIDTH=80 and HTAB=(SET,5) for DEV TYPE=SCS1 indicates that fields can be printed in columns 5 through 80 on output and received from columns 5 through 80 on input. In this case DFLD POS=(1,5) or POS=5 on input is the same as if you specified column 1 and a left margin position at 1.

  You enter data the same way, regardless of where the left margin is currently set.

RCDCTL=

Creates record definitions even if RCD statements are used in the same format definition. RCDCTL is valid only if MODE=RECORD is specified on the DEV statement.

The first data field is the first field of the message for OPTIONS=MSG. The first data field is the first field of the DPAGE or PPAGE for OPTIONS=DPAGE and PPAGE, respectively. If the first data field does not fit in the same record as the output message header, and if OPTIONS=DPAGE or PPAGE has been specified, the first data record will be sent in the next transmission. The output message header will be transmitted by itself (as is always the case for OPTIONS=MSG).

256

The maximum length of an input or output transmission. The value 256 is valid only for DEV TYPE=DPM-An or DPM-Bn.

nnnnn

The maximum length of an input or output transmission. A value is valid only for DEV TYPE=DPM-An or DPM-Bn. The length cannot be greater than 32000 or less than the length of the message output header. For information about the DPM-An message output header, see the “HDRCTL parameter” on page 301.

When TYPE=OUTPUT is specified, *nnnnn* is less than or equal to the output buffer size specified in the OUTBUF= macro at IMS system definition. If *nnnnn* is greater than the OUTBUF= value specified, one record can require multiple output transmissions and can produce undesirable results in the remote program. If fields do not exactly fit in the defined records, and NOSPAN has been specified, records might not be completely filled.

SPAN

Specifies that fields can span records.

When TYPE=OUTPUT is specified you can specify SPAN only with DEV TYPE=DPM-An. Fields can span a record boundary but not a PPAGE boundary. The remote program must include logic to associate the partial fields or deal with them separately.

NOSPAN

Specifies that fields cannot span records. Every field is contained within a record and no field has a length greater than the value specified. NOSPAN is the default.

NULL=

Specifies how MFS is to handle trailing nulls. NULL= is valid only for DEV TYPE=DPM-An and TYPE=INPUT.

KEEP

Directs MFS to ignore trailing nulls.

DELETE

Directs MFS to search for and replace trailing nulls. MFS searches input message fields for trailing nulls or for fields that are all nulls, and replaces the nulls with the fill character specified in the message definition. See “Optional Deletion of Null Characters for DPM-An” on page 196 for a discussion of the effects of NULL=DELETE.

OPTIONS=

Specifies formatting and mapping of data.

DNM

Specifies the data name.

- For TYPE=INPUT:

  DNM can be specified only for DEV TYPE=DPM-Bn. A specific DPAGE is selected to map the current or only data transmission when the DPAGE data name is supplied as the DSN parameter in the message header, and the DPAGE data name matches a defined DPAGE data name. If these conditions are not met, the last defined DPAGE name is used to map the data, unless the DPAGE is defined as conditional.

- For TYPE=OUTPUT:

  DNM can be specified for DEV TYPE=DPM-An or DPM-Bn.

  For DEV TYPE=DPM-An, use DNM with the FORS keyword on the DEV statement to specify a literal in the message header. See the discussion of the FORS= keyword and output message headers with the forms literal in “Output Message Header” on page 225 and “Chapter 5. More about Message Processing” on page 123. This parameter is optional.

  For DEV TYPE=DPM-Bn, MFS includes the following in the DD header:

  - The FMT name if OPTIONS=MSG
  - The DPAGE name if OPTIONS=DPAGE
  - The PPAGE name if OPTIONS=PPAGE

NODNM

Specifies that there is no data name.

- For TYPE=INPUT:

  NODNM can be specified for either DEV TYPE=DPM-An or DPM-Bn. MFS selects a specific DPAGE by performing a conditional test on the data received and the COND= parameter.

- For TYPE=OUTPUT:

  NODNM can be specified only for DEV TYPE=DPM-Bn. If NODNM is specified, no data structure name (DSN) is supplied in the DD header.

DPAGE

Specifies different ways of receiving and transmitting data, depending on the device type and whether TYPE=INPUT or TYPE=OUTPUT:

- For TYPE=INPUT:

  For 274x, SCS1, SCS2, or FIN, or for DEV TYPE=DPM-An or DPM-Bn, DPAGE specifies that an input message can be created from multiple DPAGEs.

  If multiple DPAGE input is not requested in MFS definitions, messages cannot be created from more than one DPAGE.

  If a single DPAGE is transmitted and contains more data than defined for the DPAGE selected, or multiple pages are transmitted, the input message is rejected and an error message is sent to the other subsystem.

- For TYPE=OUTPUT:

For DEV TYPE=DPM-An or DPM-Bn, DPAGE specifies that IMS transmits all DFLDs that are grouped in one page together. The logical page is transmitted in one or more records. If PPAGE statements are defined with the DPAGE, each PPAGE statement begins a new record. An additional logical page is sent when a paging request is received from the remote program. Each logical page is preceded by an output message header, and the label on the DPAGE is placed in the header. For DEV TYPE=DPM-Bn, the data structure name is optional in the DD header and depends on the specification of DNM or NODNM.

FLDEXIT
Specifies that the exit routine in the MSG definition MFLD is to be called for DEV TYPE=DPM-An or DPM-Bn and TYPE=INPUT.

FLDEXIT is the default.

This parameter is valid only when DEV TYPE=DPM-An or DPM-Bn and TYPE=INPUT.

NOFLDEXIT
Specifies that the exit routine in the MSG definition MFLD is to be bypassed.

MSG
Specifies different ways of creating and transmitting messages, depending on the device and whether TYPE=INPUT or TYPE=OUTPUT:

- For TYPE=INPUT:

  For DEV TYPE=274x, SCS1, SCS2, or FIN, or for DEV TYPE=DPM-An or DPM-Bn, MSG specifies that an input message can be created from a single DPAGE.

- For TYPE=OUTPUT:

  For DEV TYPE=DPM-An or DPM-Bn and TYPE=OUTPUT, MSG is the default and specifies that IMS transmits all the DFLDs within a message together as a single message group. The message is preceded by an output message header. All DFLDs are transmitted. For DEV TYPE=DPM-Bn, the data structure name is optional in the header.

PPAGE
Specifies that IMS transmits the DFLDs that are grouped in one presentation page (PPAGE) together in one chain. PPAGE is valid only when DEV TYPE=DPM-An or DPM-Bn and TYPE=OUTPUT. The presentation page is transmitted in a group of one or more records. An additional presentation page is sent when a paging request is sent to IMS from the remote program. Each presentation page is preceded by an output message header, and the label on the PPAGE statement is placed in the header. For DEV TYPE=DPM-Bn, the data structure name is optional in the DD header and depends on the specification of DNM or NODNM.

SEGEXIT
Specifies that the exit routine in the MSG definition SEG is to be called for DEV TYPE=DPM-An or DPM-Bn and TYPE=INPUT. SEGEXIT is the default.

This parameter is valid only when DEV TYPE=DPM-An or DPM-Bn and TYPE=INPUT.

NOSEGEXIT
Specifies that the exit routine in the MSG definition SEG is to be bypassed.

SIM
Specifies that MFS is to simulate attributes. This is valid only when DEV TYPE=DPM-An or DPM-Bn and TYPE=OUTPUT. SIM indicates that MFS is to simulate the attributes specified by the IMS application program and place the simulated attributes in corresponding DFLDs that are defined with ATTR=YES or YES,*nn*. The first byte of the field is used for the simulated attributes.

If the MFLD does not supply 3270 attribute information (by means of the ATTR=YES or YES,*nn* operand) for the corresponding DFLD specifying ATTR=YES or YES,*nn*, a blank is sent in the first byte of the field. The application designer of the remote program or ISC subsystem is responsible for interpreting the simulated attribute within the remote program or ISC subsystem.

SIM is the default of SIM/NOSIM2.

NOSIM2
Specifies that MFS sends a bit string that is 2 bytes long to the remote program or subsystem. This bit string is sent exactly as received from the IMS application program. 3270 extended bytes, if any (ATTR=YES,*nn*), are always sent as received from the application program and follow the 2-byte string of 3270 attributes.

If the MFLD does not supply attribute information, binary zeros are sent in the 2 bytes preceding the data for the field.

See ATTR=YES on the MFLD statement on page 277 for additional information.

HDRCTL=
Specifies, for DEV TYPE=DPM-An and DIV TYPE=OUTPUT only, the characteristics of the output message header.

FIXED
Specifies that a fully padded output message header is to be sent to the remote program. The structure of the fixed output message header is the same for all DPM output messages that are built using this FMT definition. The content of the output message header is shown in an example under “Output Format Control for SLU P DPM-An” on page 224. The base DPM output message header has a length of 7, and includes the version ID.

VARIABLE
Specifies that MIDNAME and DATANAME have trailing blanks omitted and their length fields adjusted accordingly. If MIDNAME is not used, neither the MIDNAME field nor its length is present.

nn Specifies the minimum length of the header, that is, the base header without MFS fields, as shown in the example under “Output Format Control for SLU P DPM-An” on page 224. The default is 7, which is the length of the base message header for DPM. Specifying other than 7 can cause erroneous results in the remote program.

The parameters RDPN=, DPN=, PRN=, and RPRN= refer to both the ISC ATTACH function management header and the equivalent ISC SCHEDULER function management header.

RDPN=
For DIV TYPE=INPUT, the *dfldname* specification permits the suggested return destination process name (RDPN) to be supplied in the input message MFLD referencing this *dfldname*. If *dfldname* is not specified, no RDPN is supplied in the input message.

DPN=
For DIV TYPE=OUTPUT, the '*literal*' specification requests MFS to use this literal as the DPN in the output ATTACH message header. *literal* cannot exceed eight characters, and must be enclosed in single quotes. If the *dfldname* is also specified, the data supplied in the MFLD referencing this *dfldname* is used as the DPN in the output ATTACH message header. If no output message MFLD reference to the *dfldname* exists, *literal* is used. If the data in the MFLD referencing the *dfldname* is greater than eight characters, the first eight characters are used.

PRN=
For DIV TYPE=INPUT, the *dfldname* specification permits the suggested primary resource name (PRN) to be supplied in the input message MFLD referencing this *dfldname*. If the *dfldname* is not specified, no PRN is supplied in the input message to the application program.

For DIV TYPE=OUTPUT, the '*literal*' specification requests MFS to use *literal* as the PRN in the output ATTACH message header. *literal* cannot exceed eight characters and must be enclosed in single quotes. If the *dfldname* is also specified, the data supplied in the MFLD referencing this *dfldname* is used as the PRN in the output ATTACH message header. If no output message MFLD reference to the *dfldname* exists, '*literal*' is used. If the data in the MFLD referencing the *dfldname* is greater than eight characters, the first eight characters are used.

RPRN=
For DIV TYPE=INPUT, the *dfldname* specification permits the suggested return primary resource name (RPRN) to be supplied in the input message MFLD referencing this *dfldname*. If *dfldname* is not specified, no RPRN is supplied in the input message to the application program.

For DIV TYPE=OUTPUT, '*literal*' specification requests MFS to use *literal* as the suggested return primary resource name (RPRN) in the output ATTACH message header. *literal* cannot exceed 8 characters and must be enclosed in single quotes. If the *dfldname* is also specified, the data supplied in the MFLD referencing this *dfldname* is used as the RPRN in the output ATTACH message header. If no output message MFLD reference to the *dfldname* exists, '*literal*' is used. If the data in the MFLD referencing the *dfldname* is greater than 8 characters, the first 8 characters are used.

OFTAB=
Directs MFS to insert output field tab separator characters in the output data stream for the message. If OPTIONS=DNM and OFTAB, then the OFTAB character is placed in the DD header and an indicator is set to MIX or ALL. If OPTIONS=NODNM, then no DD header is sent.

X'*hh***'**
Specifies a hexadecimal character (*hh*) to be used as the output field tab separator character. X'3F' and X'40' are invalid.

C'*c***'**
Specifies a character (*c*) to be used as the output field tab separator character. You cannot specify a blank for the character (**C' '**).

The character specified cannot be present in the data stream from the IMS application program. If it is present, it is changed to a blank (X'40').

If an output field tab separator character is defined, either MIX or ALL can also be specified. The default is MIX.

MIX
Specifies that the output field tab separator character is inserted into each individual field with no data or with data that is less than the defined DFLD length.

ALL
Specifies that the output field tab separator character is inserted into all fields, regardless of data length.

COMPR=
Directs MFS to remove trailing blanks from short fields, fixed-length fields, or all fields presented by the application program.

For DPM-An devices, trailing blanks are removed from the end of a segment if all of the following are specified:

- FILL=NULL or FILL=PT
- GRAPHIC=YES for the current segment being mapped
- OPT=1 or OPT=2, in the MSG segment

If these conditions are met, trailing blanks are replaced as follows:

FIXED
Specifies that trailing blanks from fixed-length fields are replaced by nulls.

SHORT
Specifies that trailing blanks from fields shortened by the application are replaced by nulls.

ALL
Specifies that trailing blanks from all fields are replaced by nulls.

The trailing nulls are compressed at the end of the record. See the FILL= operand on page 307 for additional information.

For DPM-Bn devices, trailing blanks are removed if all of the following are specified:

- OFTAB (on the current DIV statement), FILL=NULL, or FILL=PT
- GRAPHIC=YES for the current segment being mapped
- OPT=1 or OPT=2 in the MSG segment

If these conditions are met, trailing blanks are removed as follows:

FIXED
Specifies that trailing blanks are to be removed from fixed-length fields.

SHORT
Specifies that trailing blanks are to be removed from fields shortened by the application.

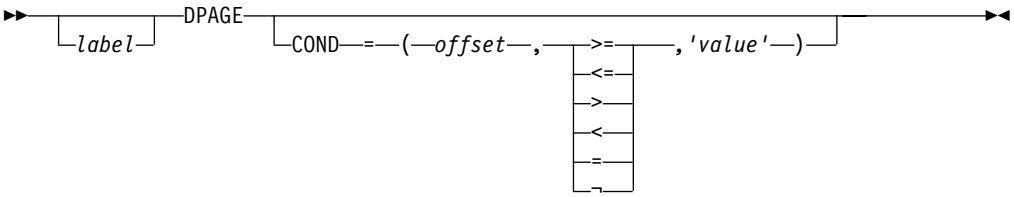
ALL
Specifies that trailing blanks are to be removed from all fields.

For additional information on blank compression for DPN-BN devices, see “Trailing Blank Compression” on page 232.

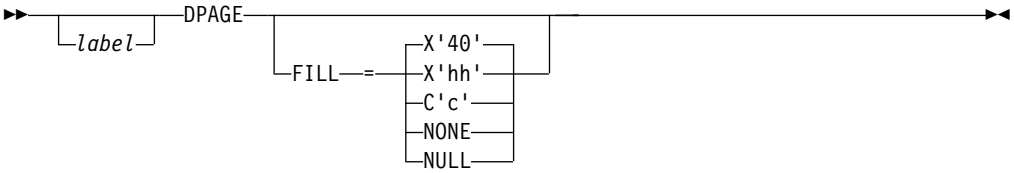
DPAGE Statement

The DPAGE statement defines a logical page of a device format. This statement can be omitted if none of the message descriptors referring to this device format (FMT) contain LPAGE statements and no specific device option is required.

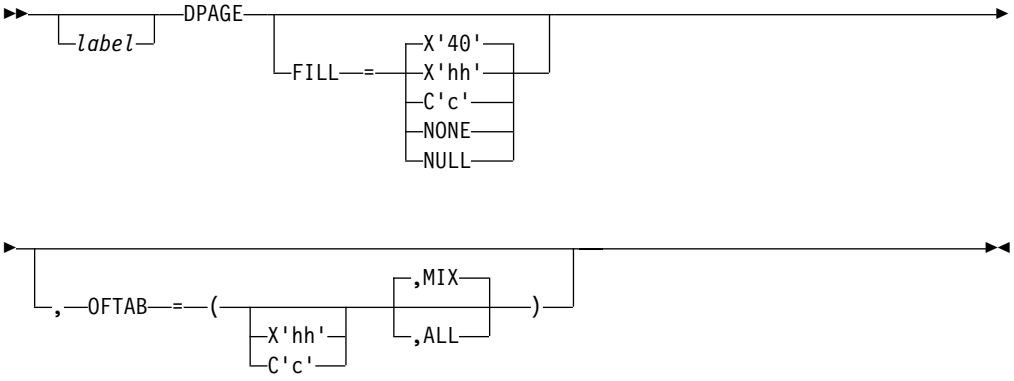
Format for DEV TYPE=274X, DPM-An, or DPM-Bn AND DIV TYPE=INPUT:



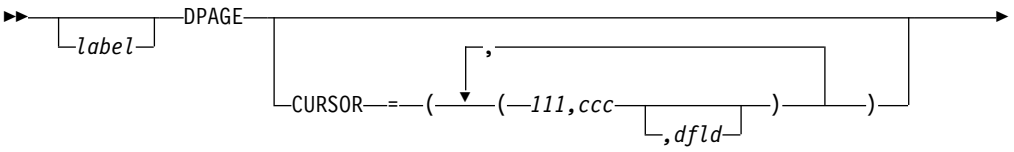
Format for DEV TYPE=274X or DPM-An AND DIV TYPE=OUTPUT:

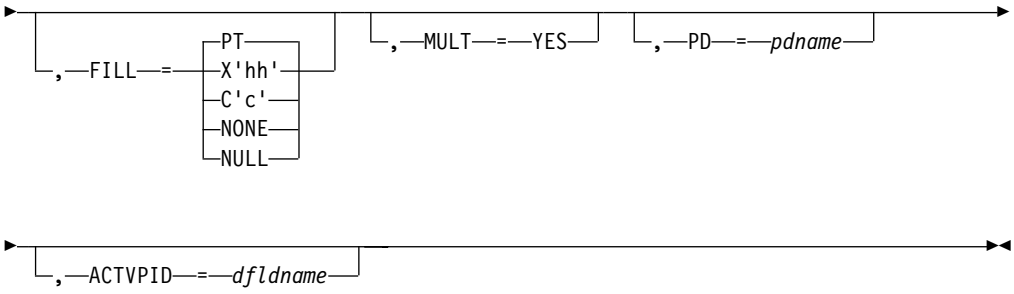


Format for DEV TYPE=DPM-Bn AND DIV TYPE=OUTPUT:

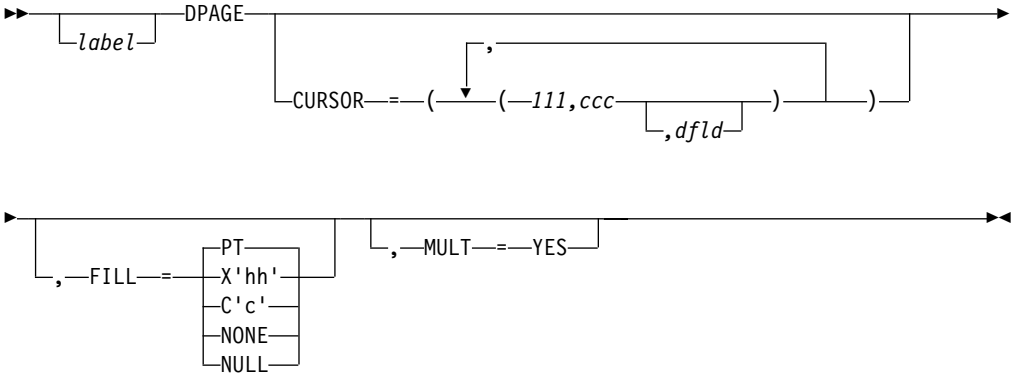


Format for DEV TYPE=3270-An:

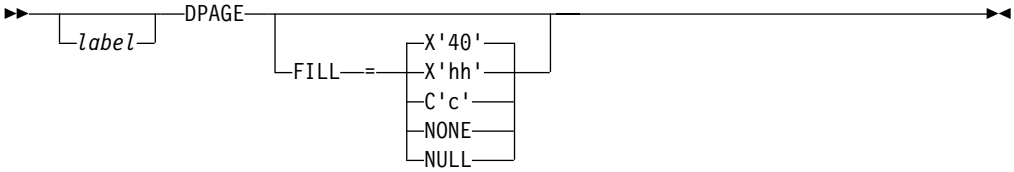




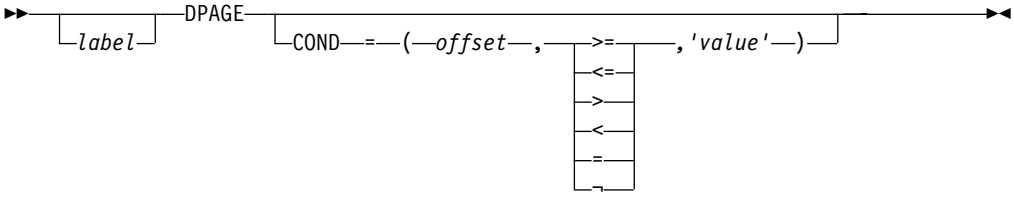
Format for DEV TYPE=3270:



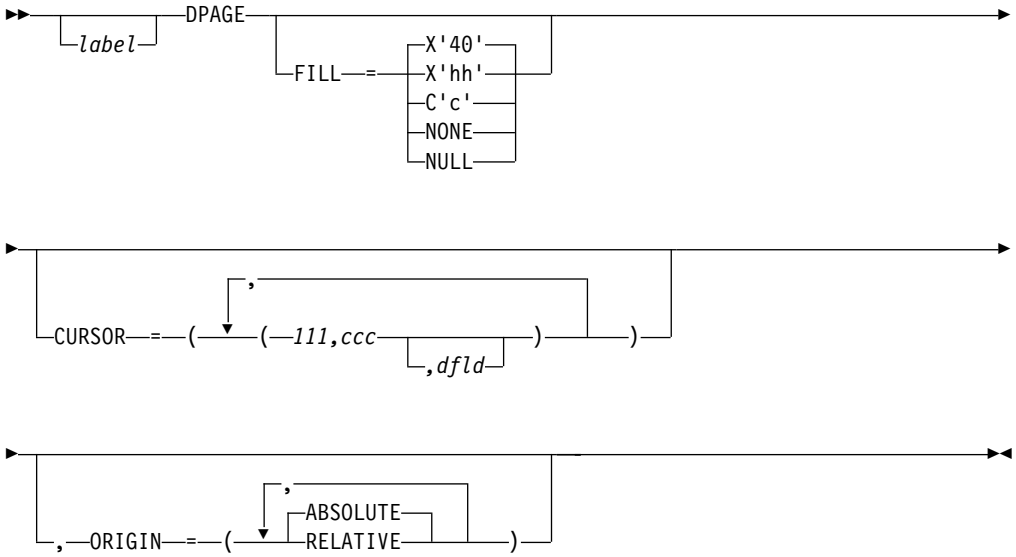
Format for DEV TYPE=3270P:



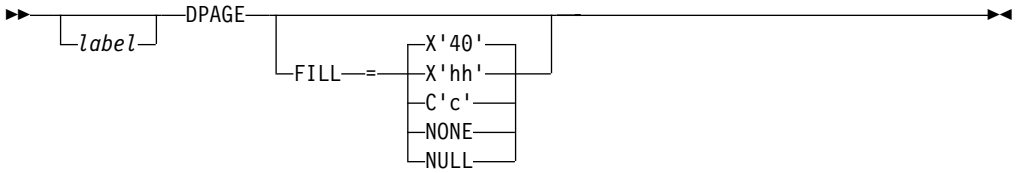
Format for DEV TYPE=FIN:



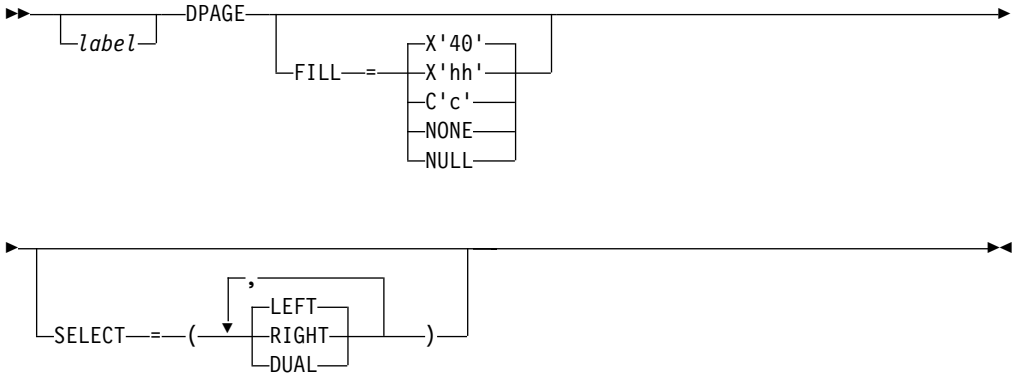
Format for DEV TYPE=FIDS, FIDS3, FIDS4, or FIDS7:



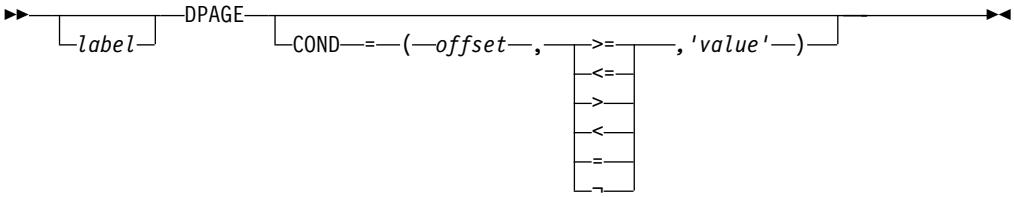
Format for DEV TYPE=FIJP or FIPB:



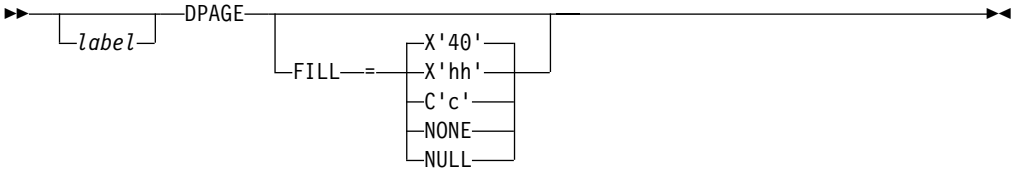
Format for DEV TYPE=FIFP:



Format for DEV TYPE=SCS1 or SCS2 AND DIV TYPE=INPUT:



Format for DEV TYPE=SCS1 or SCS2 AND DIV TYPE=OUTPUT:



Parameters:

label
A 1- to 8-byte alphanumeric name can be specified for this device format that contains LPAGE SOR= references, or if only one DPAGE statement is defined for the device. If multiple DEV statements are defined in the same FMT definition, each must contain DPAGE statements with the same label.

For device type DPM-An and DIV statement OPTIONS=DPAGE, this name is sent to the remote program as the data name in the output message header. If *label* is omitted, MFS generates a diagnostic name and sends it to the remote program in the header. If the DPAGE statement is omitted, the label on the FMT statement is sent in the output message header. If OPTIONS=DNM, the label on the FMT statement is sent as the DSN in the DD header.

COND=
Specifies a conditional test to be performed on the first input record. The offset specified is relative to zero. The specification of the offset must allow for the LLZZ field of the input record (for example, the first data byte is at offset 4). If the condition is satisfied, the DFLDs defined following this DPAGE are used to format the input. When no conditions are satisfied, the last defined DPAGE will be used only if the last defined DPAGE does not specify COND=. If the COND= parameter is specified for the last DPAGE defined and the last defined DPAGE condition is not satisfied, the input message will be rejected. Multiple LPAGE definitions are allowed in message input definitions.

If this keyword is specified, and OPTIONS=NODNM is specified on the DIV statement, this specification is used for DPAGE selection. If this keyword is specified and OPTIONS=DNM is specified on the DIV statement, the COND= specification is ignored and the data structure name from the DD header is used for DPAGE selection.

Lowercase data entered from 274X, Finance, SCS1, or SCS2 keyboards is not translated to uppercase when the COND= comparison is made. Therefore, the literal operand must also be in lowercase.

FILL=
Specifies a fill character for output device fields. Default value for all device

types except the 3270 display is X'40'; default for the 3270 display is PT. For 3270 output when EGCS fields are present, only FILL=PT or FILL=NULL should be specified. A FILL=PT erases an output field (either a 1- or 2-byte field) only when data is sent to the field, and thus does not erase the DFLD if the application program message omits the MFLD. For DPM-Bn, if OFTAB is specified, FILL= is ignored and FILL=NULL is assumed.

NONE
Must be specified if the fill character from the message output descriptor is to be used to fill the device fields.

X'*hh***'**
Specifies a hexadecimal character (*hh*) that is used to fill the device fields.

C'*c***'**
Specifies a character (*c*) that is used to fill the device fields.

NULL
Specifies that fields are not to be filled. For devices other than the 3270 display, *compacted lines* are produced when message data does not fill the device fields.

For DPM-An devices, trailing nulls (X'3F') are removed from all records transmitted to the remote program or subsystem. Trailing nulls are removed up to the first non-null character. Null characters between non-null characters are transmitted. If the entire record is null, but more data records follow, a record containing a single null is transmitted to the remote program. If the entire record is null and more records follow, if OPTIONS=MSG or DPAGE, or in a PPAGE, if OPTIONS=PPAGE, then all null records are deleted to the end of that DPAGE or PPAGE.

PT
Is identical to NULL except for the 3270 display. For the 3270 display, specifies that output fields that do not fill the device field (DFLD) are followed by a program tab character to erase data previously in the field; otherwise, this operation is identical to FILL=NULL.

For 3270 display devices, any specification with a value less than X'3F' is changed to X'00' for control characters or to X'40' for other nongraphic characters. For all other devices, any FILL=X'hh' or FILL=C'c' specification with a value less than X'3F' is ignored and defaulted to X'3F' (which is equivalent to a specification of FILL=NULL).

MULT=YES
Specifies that multiple physical page input messages are allowed for this DPAGE.

CURSOR=
Specifies the position of the cursor on a physical page. Multiple cursor positions might be required if a logical page or message consists of multiple physical pages. The value *lll* specifies line number and *ccc* specifies column. Both *lll* and *ccc* must be greater than or equal to 1. The cursor position must either be on a defined field or defaulted. The default *lll*,*ccc* value for 3270 displays is 1,2. For Finance display components, if no cursor position is specified, MFS does not position the cursor—the cursor is normally placed at the end of the output data on the device. For Finance display components, all cursor positioning is absolute, regardless of the ORIGIN= parameter specified.

The *dfld* parameter provides a method for supplying the application program with cursor information on input and allowing the application program to specify cursor position on output.

Recommendation: Use the cursor attribute facility (specify ATTR=YES in the MFLD statement) for output cursor positioning.

The *dfld* parameter specifies the name of a field containing the cursor position. This name can be referenced by an MFLD statement and must *not* be used as the label of a DFLD statement in this DEV definition. The format of this field is two binary halfwords containing line and column number, respectively. When this field is referred to by a message input descriptor, it contains the cursor position at message entry. If referred to by a message output descriptor, the application program places the desired cursor position into this field as two binary halfwords containing line and column, respectively. Binary zeros in the named field cause the values specified for *lll*,*ccc* to be used for cursor positioning during output. During input, binary zeros in this field indicate that the cursor position is not defined. The input MFLD referring to this *dfld* should be defined within a segment with GRAPHIC=NO specified or should use EXIT=(0,2) to convert the binary numbers to decimal.

ORIGIN=

Specifies page positioning on the Finance display for each physical page defined. Default value is ABSOLUTE.

ABSOLUTE

Erases the previous screen and positions the page at line 1 column 1. The line and column specified in the DFLD statement become the actual line and column of the data on the screen.

RELATIVE

Positions the page starting on column 1 of the line following the line where the cursor is positioned at time of output. Results might be undesirable unless all output to the device is planned in a consistent manner.

OFTAB=

Directs MFS to insert the output field tab separator character specified on this DPAGE statement for the output data stream of the DPAGE being described.

X'*hh***'**

Specifies a hexadecimal character (*hh*) to be used as the output field tab separator character. X'3F' and X'40' are invalid.

C'*c***'**

Specifies a character (*c*) to be used as the output field tab separator character. You cannot specify a blank for the character (**C' '**).

The character specified cannot be present in data streams from the IMS application program. If it is present, it is changed to a blank (X'40').

If the output field tab separator character is defined, either MIX or ALL can also be specified. Default value is MIX.

MIX

Specifies that an output field tab separator character is to be inserted into each individual field with no data or with data less than the defined DFLD length.

ALL
Specifies that an output field tab separator character is to be inserted into all fields, regardless of data length.

SELECT=
Specifies carriage selection for a FIFP device with FEAT=DUAL specified in the previous DEV statement. It is your responsibility to ensure that proper forms are mounted and that left margins are set properly. Default value is LEFT.

LEFT
Causes the corresponding physical page defined in this DPAGE to be directed to the left platen.

RIGHT
Causes the corresponding physical page defined in this DPAGE to be directed to the right platen.

DUAL
Causes the corresponding physical page defined in this DPAGE to be directed to both the left and right platens.

PD=
(for the 3180 and 3290 in partition formatted mode) Specifies the name of the partition descriptor of the partition associated with the DPAGE statement. This is the parameter that maps a logical page of a message to or from the appropriate partition. The name of the PD must be contained within the PDB statement specified in the DEV statement.

ACTVPID=
(for the 3290 in partition formatted mode) Specifies the name of an output field in the message containing the partition identification number (PID) of the partition to be activated. This *dfldname* must be referenced by an MFLD statement and must not be used as the label of a DFLD statement in the DEV definition. The application program places the PID of the partition to be activated in this field. The PID must be in the format of a two byte binary number ranging from X'0000' to X'000F'.

Restriction: Do not specify this operand for the 3180. Because only one partition is allowed for this device, you do not need to specify an active partition.

Chapter 10. MFS Language Utility

This chapter describes the control statements used by the MFS Language utility. There are two major categories of control statements:

- *Definition statements* are used to define message formats, device formats, partition sets, and operator control tables.
- *Compilation statements* are those used to control the compilation and SYSPRINT listings of the definition statements.

Use the definition and compilation control statements to identify a particular function performed by the utility and to specify various options.

In this Chapter:

- "Utility Control Statements"

Utility Control Statements

Listed below are the definition and compilation control functions.

- SYSPRINT LISTING CONTROL

  The following parameters are provided to format the compilation listing: XREF, SUBS, COMP, DIAG, and LINECNT.

- SYSIN and SYSLIB RECORD STACKING and UNSTACKING

  Control statements are provided to allow one or more SYSIN or SYSLIB records to be processed and kept in processor storage for reuse later in the compilation. These statements are an alternative to the COPY facility for groups of statements that are repeated.

  MFLD and DFLD statements can be repetitively generated if preceded by a DO statement and followed by an ENDDO statement. Repetitive DFLD generation supports increments to line and column position information.

- ALPHA CHARACTER GENERATION

  The ALPHA statement allows specification of additions to the set of characters as alphabetic.

- COPY

  The COPY statement allows members of partitioned data sets to be copied into the input stream of the utility preprocessor.

Control Statement Syntax

The control statements are written in assembler-like language with the following standard format:

label *operation* *operand* *comments*

Figure 60. Control Statement Syntax for MFS Language Utility

label

Identifies the statement; if it is shown as optional, it can be omitted. When included, the name must begin in the first position of the statement (column 1) and must be followed by one or more blanks. It can contain from one to eight alphanumeric characters (one to six, for the FMT label), the first of which must be alphabetic.

© Copyright IBM Corp. 1974, 2000

operation
Identifies the type of control statement. It normally begins in column 10 and must be preceded and followed by one or more blanks.

operand
Is made up of one or more parameters, which can be positional or keyword parameters. A positional parameter in MFS control statements always appears in the first position of the operand, normally starting in column 16. The position of a keyword parameter is not important. The parameters within one operand are separated by commas. In the syntactical description of the control statements below, parameters preceded by commas are thus identified as keyword parameters. The operand field itself must be preceded and followed by one or more blanks.

comments
Can be written in a utility control statement, but they must be separated from the last parameter of the operand field by one or more blanks. (If the statement does not include an operand, the comment should be separated from the statement by at least one blank.) A comment line begins with an asterisk in column 1.

Continuation is accomplished by entering a nonblank character in column 72. If the current line is a comment, then the continuation line can begin in any column.

Other considerations are as follows:

- There is no limit on the number of continuation lines.
- There is no limit on the number of characters in the operand field. Individual operand items cannot exceed 256 characters, excluding trailing and embedded second quote characters.
- If a nonstandard character (for definition, see “ALPHA Statement” on page 391) is detected in a literal, a severity 4 warning message is issued. The nonstandard character is retained in the literal.
- If the current line is a control statement, the continuation line must begin in column 16.
- A single ampersand is needed to generate one ampersand character in the literal.

In addition to the definition and compiler statement specifications, several parameters can be specified in the EXEC statement PARM keyword to control the current compilation for the preprocessor and phase 1; one parameter can be specified for phase 2.

Five Special Rules

The five special rules that follow use actual MFS code as examples.

1. If you code a statement such that an equal sign or a left parenthesis immediately precedes a comma, you can omit the comma.

   ,FTAB=(,FORCE) could be coded as ,FTAB=(FORCE)
2. If you code a statement such that an equal sign immediately precedes a single item enclosed in parentheses, you can omit the parentheses.

   ,FTAB=(,FORCE) could be coded as ,FTAB=,FORCE
3. You can apply both Rule 1 and Rule 2, in either order, to a single item.

   ,FTAB=(,FORCE) could be coded as ,FTAB=FORCE
4. Under no condition can you specify a keyword without specifying at least one parameter immediately after that keyword.

Neither ,FTAB= nor ,FTAB=,LDEL='**' is permitted.

5. Blanks are required between labels and statement type names, and between statement type names and their parameters; they are not permitted elsewhere unless explicitly represented by the symbol ƀ.

   DEV ,PAGE is correct, but DEV,PAGE and ,FTAB= (,MIX) are incorrect.

Syntax Errors

The MFS Language utility attempts to recover from syntax errors in source statements. No guarantee exists for the correctness of the assumptions made in the recovery, and these assumptions can differ in different releases of IMS. Assumptions made during recovery are based on (1) what is expected when the incorrect item is encountered; (2) what could appear to the right of the item preceding the incorrect item; and (3) what could appear to the left of the incorrect item.

During the process of error recovery, the following notation can be used in the diagnostic messages:

; indicates that the end of the source statement was encountered. The position marker points to the position immediately following the last source item scanned.

L refers to a literal operand item.

V refers to an identifier operand item (alphabetic character optionally followed by alphanumeric characters).

I refers to a numeric operand item.

A refers to an alphanumeric operand item (numeric character optionally followed by alphanumeric characters).

D refers to a delimiter operand item.

Most error recovery messages have a severity code of 4, indicating a warning level error. When an item is deleted, or the syntax scan is aborted, the statement cannot be validly processed and a severity code of 8 is generated.

Invalid Sequence of Statements

The language utility preprocessor routines that process MSG, FMT, PDB, or TABLE definition statements are organized hierarchically. A routine for a given level processes a statement at that level, reads the next statement, then determines which routine will next receive control.

If the statement just read is the next lower level statement (for example, a DIV statement following a DEV statement), the next lower level routine (for example, the DIV statement processor) is called.

If the statement just read is not the next lower level statement, control can be passed to one of the following three routines:

1. The next lower level routine to assume the missing statement (for example, the DIV processor if a DEV statement is followed by a DPAGE statement)
2. The same level routine if the statement just read is of the same level as the processor (for example, a series of DFLD statements)
3. The next higher level routine (the calling routine) if the statement just read is not the same or the next lower level (for example, a DEV statement following a DFLD statement, an invalid statement, or a statement out of sequence)

Thus, if the hierarchic structure of a MSG, FMT, PDB, or TABLE definition is invalid or a statement operator is misspelled, case (3) will result in control being returned to successively higher level routines. At the highest level, only a FMT, MSG, TABLE, PDB, or END statement will be accepted by the preprocessor. Therefore, all statements before the next FMT, PDB, MSG, TABLE or END statement will be flushed (that is, not processed) and flagged with the appropriate error message.

Summary of Control Statements

The definition of message formats, device formats, partition sets, and operator control tables is accomplished with separate hierarchic sets of definition statements.

MSG Statement Set
is used to define message formats. It includes the following statements:

MSG	Identifies the beginning of a message definition.
LPAGE	Identifies a related group of segment/field definitions.
PASSWORD	Identifies a field or fields to be used as an IMS password.
SEG	Identifies a message segment.
DO	Requests iterative processing of the subsequent MFLD statements.
MFLD	Defines a message field. Iterative processing of MFLD statements can be invoked by specifying DO and ENDDO statements. To accomplish iterative processing, the DO statement is placed before the MFLD statements and the ENDDO after the MFLD statements.
ENDDO	Terminates iterative processing of the preceding MFLD statements.
MSGEND	Identifies the end of a message definition.

FMT Statement Set
is used to define device formats. It consists of the following statements:

FMT	Identifies the beginning of a format definition.
DEV	Identifies the device type and operational options.
DIV	Identifies the format as input, output, or both.
DPAGE	Identifies a group of device fields corresponding to an LPAGE group of message fields.
PPAGE	Identifies a group of logically related records that can be sent to a remote application program at one time.
DO	Requests iterative processing of the subsequent RCD or DFLD statements.
RCD	Identifies a group of related device fields that are sent to a remote application program as a single record.
DFLD	Defines a device field. Iterative processing of DFLD statements can be invoked by specifying DO and ENDDO statements. To accomplish iterative processing, the DO statement is placed before the DFLD statements and the ENDDO after the DFLD statements.
ENDDO	Terminates iterative processing of the previous RCD or DFLD statements.

FMTEND Identifies the end of a format definition.

PDB Statement Set
is used to define partition sets (Partition Descriptor Blocks). It consists of the following statements:

PDB Identifies the beginning of a partition set definition and allows the specification of several parameters that describe it.

PD Defines a Partition Descriptor, which contains the parameters necessary to describe a partition.

PDBEND Identifies the end of a partition set definition.

TABLE Statement Set
is used to define operator control tables. It includes the following statements:

TABLE Identifies the beginning of a table definition.

IF Defines a conditional test and resulting action.

TABLEEND Identifies the end of a table definition.

Compilation Statements
are used for variable functions. Compilation statements that are supported by the MFS Language utility are listed below in alphabetic order:

ALPHA Defines a set of characters to be considered alphabetic for the purpose of defining field names and literals.

COPY Copies a member of the partitioned data set represented by the SYSLIB DD statement into the input stream of the preprocessor.

DO Requests iterative processing of MFLD or DFLD definition statements.

EJECT Ejects SYSPRINT listing to the next page.

END Defines the end of data for SYSIN processing.

ENDDO Terminates iterative processing of MFLD, RCD, or DFLD definition statements.

EQU Equates a symbol with a number, alphanumeric identifier, or literal.

PRINT Controls SYSPRINT options.

RESCAN Controls EQU processing.

SPACE Skips lines on the SYSPRINT listing.

STACK Delineates one or more SYSIN or SYSLIB records that are to be kept in processor storage for reuse.

TITLE Provides a title for the SYSPRINT listing.

UNSTACK Retrieves previously stacked SYSIN or SYSLIB records.

Compilation statements are inserted at logical points in the sequence of control statements. For example, TITLE could be first, and EJECT could be placed before each MSG, FMT, or TABLE statement.

EXEC Statement Parameters

EXEC statement parameters supported by the MFS Language utility have variable compilation control functions. Parameters can be specified on the EXEC statement for the preprocessor and phase 1 to:

- Control the printed output
- Compress the reference library (IMS.REFERAL)
- Request diagnostic information
- Indicate which MFS device characteristics table is to be used
- Prevent control blocks with a specified level of error from being written in IMS.REFERAL

Parameters can also be specified on the EXEC statement for phase 2 to specify whether IMS.FORMAT and IMS.REFERAL should be compressed and whether $$IMSDIR should be automatically updated after deletions.

The DEVCHAR parameter specifies the suffix of the MFS device characteristics table to be used. The device characteristics table is accessed only if DEV TYPE=3270-An (where n is 1 to 15) is coded as input to the MFS Language utility. For a description of the MFS device characteristics table, see “MFS Device Characteristics Table” on page 249.

The EXEC statement parameters supported by the MFS Language utility have variable compilation control functions. The parameters that can be specified are:

NOXREF|XREF
Specifies whether (XREF) or not (NOXREF) a sorted cross-reference listing should be provided. A sorted cross-reference listing includes a list of all the labels and related references. The default is NOXREF.

NOCOMP|COMP
Specifies whether (COMP or COMPOSITE) or not (NOCOMP) the composite or final version of the statement, after error recovery or substitution has modified it, will be printed. A composite statement reflects syntactic assumptions made during error recovery. Semantic assumptions do not appear in composite statements but are reflected in the intermediate text blocks. The default is NOCOMP.

NOSUBS|SUBS
Specifies whether (SUBS or SUBSTITUTE) or not (NOSUBS) any statement containing a substitution variable (EQU operand) is printed. The default is NOSUBS.

NODIAG|DIAG
Specifies whether (DIAG or DIAGNOSTIC) or not (NODIAG) the XREF, COMP, and SUBS options should be set on and diagnostic information be printed. The default is NODIAG, which has no effect on the setting of the XREF, COMP, and SUBS options but suppresses printing of the diagnostic information.

NOCOMPRESS|COMPRESS
Specifies whether (COMPRESS) or not (NOCOMPRESS) the IMS.REFERAL library is to be compressed before new ITBs are added. The default is NOCOMPRESS.

DIRUPDT= UPDATE|NOUPDATE
Specifies whether (UPDATE) or not (NOUPDATE) the special index directory ($$IMSDIR) will be automatically updated after one or more blocks have been deleted from a format library. You can bypass the $$IMSDIR update by specifying NOUPDATE. The default is UPDATE.

LINECNT=*nn*
Specifies how many lines per page should be printed. The default is 55.

STOPRC=*nn*
Specifies the severity code compare value. MSG, FMT, and TABLE blocks whose error severity equals or exceeds this value will not be written to the IMS.REFERAL library. The default is 08.

DEVCHAR=*n*
Specifies the alphanumeric suffix character (x) used as the final character of the name of the device characteristics table DFSUDT0x loaded when DEV TYPE=3270-An is encountered. The default is zero (DFSUDT00).

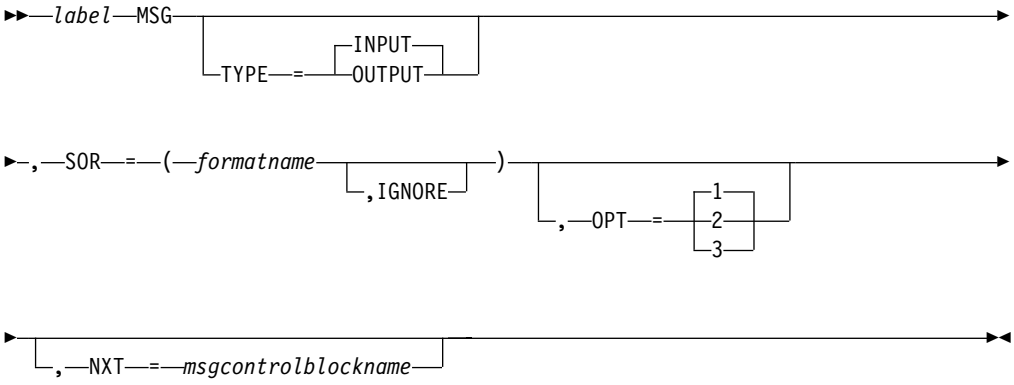
The remainder of this chapter describes, in detail, the utility control statements. The definition statements are described in the sequence shown above, with the DO and ENDDO compilation statements where they would normally be coded—before and after the MFLD or DFLD statements. The compilation statement formats are sequenced according to related function (if any)—ALPHA; COPY; EQU and RESCAN (equate processing); STACK and UNSTACK (stacking SYSIN/SYSLIB records); TITLE, PRINT, SPACE, and EJECT (SYSPRINT listing control); and END.

Message Definition Statements

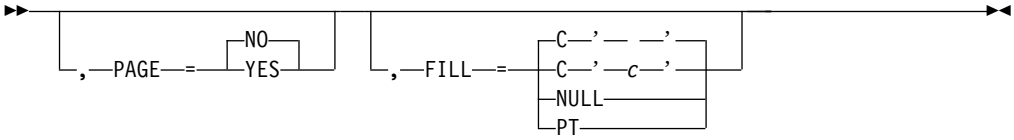
MSG Statement

The MSG statement initiates and names a message input or output definition.

Format for MSG TYPE=INPUT or OUTPUT:



Format for MSG TYPE=OUTPUT Only:



Parameters:

label
A one- to eight-character alphanumeric name must be specified. This label can be referred to in the NXT operand of another message descriptor.

TYPE=
Defines this definition as a message INPUT or OUTPUT control block. The default is INPUT.

SOR=
Specifies the source name of the FMT statement which, with the DEV statement, defines the terminal or remote program data fields processed by this message descriptor. Specifying IGNORE for TYPE=OUTPUT causes MFS to use data fields specified for the device whose FEAT= operand specifies IGNORE in the device format definition (see “DEV Statement” on page 330). For TYPE=INPUT, IGNORE should be specified only if the corresponding message output descriptor specified IGNORE. If you use SOR=IGNORE, you must specify IGNORE on both the message input descriptor and the message output descriptor.

OPT=
Specifies the message formatting option used by MFS to edit messages. The default is 1. Options 1, 2, and 3 are described in “Output Message Formatting Options” on page 204 and “Input Message Formatting Options” on page 186.

NXT=
Specifies the name of a message descriptor to be used to map the next expected message as a result of processing a message using this message descriptor. If TYPE=INPUT, NXT= specifies a message output descriptor. If TYPE=OUTPUT, NXT= specifies a message input descriptor. For ISC output, NXT= becomes the RDPN in the ATTACH FM header.

If TYPE=OUTPUT and the *formatname* specified in the SOR= operand contains formats for 3270 or 3270P device types, the *msgcontrolblockname* referred to by NXT= must use the same *formatname*.

PAGE=
Specifies whether (YES) or not (NO) operator logical paging (forward and backward paging) is to be provided for messages edited using this control block. This operand is valid only if TYPE=OUTPUT. The default is NO, which means that only forward paging of physical pages is provided.

FILL=
Specifies a fill character for output device fields. This operand is valid only if TYPE=OUTPUT. The default is C' '. The fill specification is ignored unless FILL=NONE is specified on the DPAGE statement in the FMT definition. For 3270 output when EGCS fields are present, only FILL=PT or FILL=NULL should be specified. A FILL=PT erases an output field (either a 1- or 2-byte field) only when data is sent to the field, and thus does not erase the DFLD if the application program message omits the MFLD. For DPM-Bn, if OFTAB is specified, FILL= is ignored and FILL=NULL is assumed.

C'*c***'**
Character '*c*' is used to fill device fields. For 3270 display devices, any specification with a value less than X'3F' is changed to X'00' for control characters or to X'40' for other nongraphic characters. For all other devices, any FILL=C'*c*' specification with a value less than X'3F' is ignored and defaulted to X'3F' (which is equivalent to a specification of FILL=NULL).

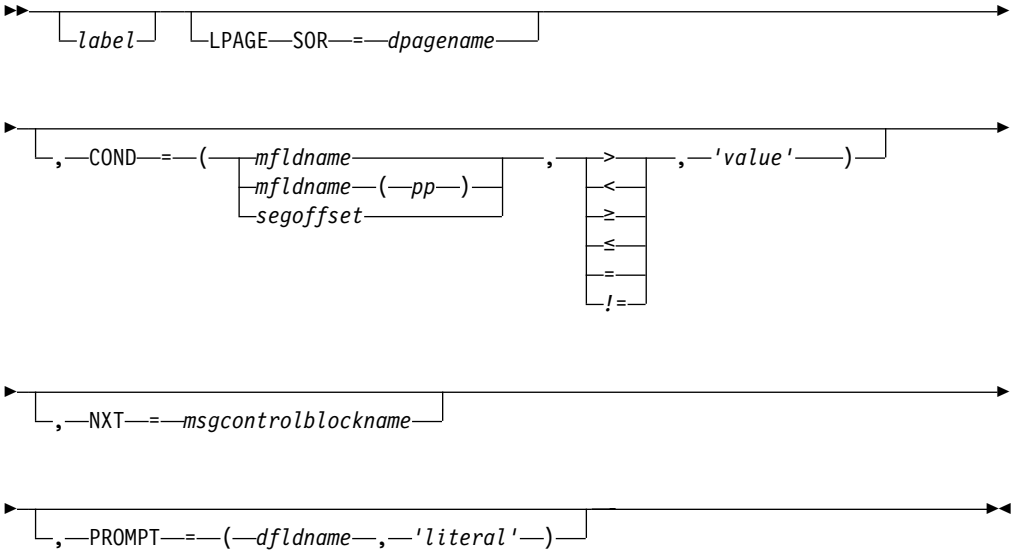
NULL
Specifies that fields are not to be filled. For devices other than 3270 and SLU 2 display, 'compacted lines' are produced when message data does not fill device fields.

PT
Is identical to NULL except for 3270 and SLU 2 display. For 3270 and SLU 2 display, PT specifies that output fields that do not fill the device field (DFLD) are followed by a program tab character to erase data previously in the field.

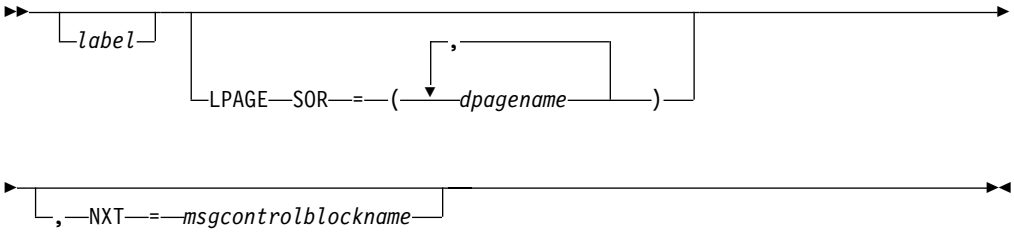
LPAGE Statement

The optional LPAGE statement defines a group of segments comprising a logical page.

Format for MSG TYPE=OUTPUT:



Format for MSG TYPE=INPUT:



Parameters:

label
A one- to eight-character alphanumeric name can be specified to uniquely identify this statement.

SOR=
Specifies the name of the DPAGE statement that defines the device format for this logical page. If TYPE=INPUT and more than one DPAGE can be used as a source of data to create an input message, more than one *dpagename* can be specified.

COND=

Describes a conditional test that, if successful, specifies that the segment and field definitions following this LPAGE are to be used for output editing of this logical page. The specified portion of the first segment of a logical page is examined to determine if it is greater than (>), less than (<), greater than or equal to (≥), less than or equal to (≤), equal to (=), or not equal to (≠) the specified literal value to determine if this LPAGE is to be used for editing. COND= is not required for the last LPAGE statement in the MSG definition.

The area examined can be defined by a field name (*mfldname*), an offset in a field (*mfldname(pp)* where *pp* is the offset in the named field), or an offset in the segment (*segoffset*). If the *mfldname(pp)* form is used, *pp* must be greater than or equal to 1. The length of the compare is the length of the specified literal. If OPT=3 is specified on the previous MSG statement, the area to be examined must be within one field as defined on an MFLD statement.

If *segoffset* is used, it is relative to zero, and the specification of that offset must allow for LLZZ of the segment (that is, the first data byte is at offset 4).

If *pp* is used, the offset is relative to 1 with respect to the named field (that is, the first byte of data in the field is at offset 1, not zero).

If the *mfldname* specified is defined with ATTR=YES, the *pp* offset must be used. The minimum offset specified must be 3. That is, the first byte of data in the field is at offset 3, following the two bytes of attributes.

If ATTR=*nn* is specified, the minimum offset must be one plus twice *nn*. Thus, if ATTR=2 is specified, *pp* must be at least 5, and, if ATTR=(YES,2) is specified, *pp* must be at least 7.

If the conditional tests for all LPAGEs fail, the last LPAGE in this MSG definition is used for editing.

If LPAGE selection is to be specified using the command data field, that is, `/FORMAT`*modname...(data)*, the MFLD specified in the LPAGE COND=*mfldname* parameter should be within the first 8 bytes of the associated LPAGEs of the MOD.

NXT=

Specifies the name of the message descriptor to be used to map the next message if this logical page is processed. This name overrides any NXT=*msgcontrolblockname* specified on the preceding MSG statement.

PROMPT=

Specifies the name of the DFLD into which MFS should insert the specified literal when formatting the last logical page of an output message. If FILL=NULL is specified once the prompt literal is displayed, it can remain on the screen if your response does not cause the screen to be reformatted.

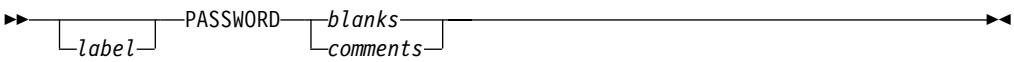
PASSWORD Statement

The PASSWORD statement identifies one or more fields to be used as an IMS password. When used, the PASSWORD statement and its associated MFLDs must precede the first SEG statement in an input LPAGE or MSG definition. Up to 8 MFLD statements can be specified after the PASSWORD statement but the total password length must not exceed 8 characters. The fill character must be X'40'. For

option 1 and 2 messages, the first 8 characters of data after editing are used for the IMS password. For option 3 messages, the data content of the first field after editing is used for the IMS password.

A password for 3270 input can also be defined in a DFLD statement. If both password methods are used, the password specified in the MSG definition is used.

Format:



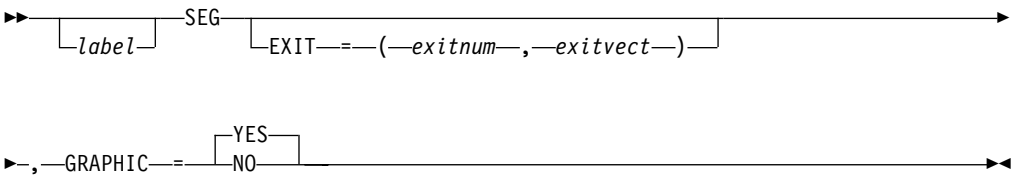
Parameters:

label
A one- to eight-character alphanumeric name can be specified to uniquely identify this statement.

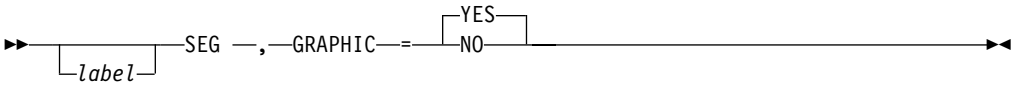
SEG Statement

The SEG statement delineates message segments and is required only if multisegment message processing is required by the application program. Output message segments cannot exceed your specified queue buffer length. Only one segment should be defined for TYPE=INPUT MSGs when the input message destination is defined as a single segment command or transaction. If more than one segment is defined, and the definition is used to input a single segment command or transaction, care must be used to ensure that your input produces only one segment after editing.

Format for MSG TYPE=INPUT:



Format for MSG TYPE=OUTPUT:



Parameters:

label
A 1- to 8-character name can be specified to uniquely identify this statement.

EXIT=
Describes the segment edit exit routine interface for this message segment. *exitnum* is the exit routine number and *exitvect* is a value to be passed to the exit routine when it is invoked for this segment. *exitnum* can range from 0 to 127. *exitvect* can range from 0 to 255. Unless NOSEGEXIT is specified on the

DIV statement (for DPM devices only), the SEG exit is invoked when processing completes for the input segment.

GRAPHIC=

Specifies for MSG TYPE=INPUT whether (YES) or not (NO) IMS should perform upper case translation on this segment if the destination definition requests it (see the EDIT= parameter of the TRANSACT or NAME macro). The default is YES. If input segment data is in nongraphic format (packed decimal, EGCS, binary, and so forth), GRAPHIC=NO should be specified. When GRAPHIC=NO is specified, FILL=NULL is invalid for MFLDs within this segment.

The list below shows the translation that occurs when GRAPHIC=YES is specified and the input message destination is defined as requesting upper case translation:

Before Translation	After Translation
a through z	A through Z
X'81' through X'89'	X'C1' through X'C9'
X'91' through X'99'	X'D1' through X'D9'
X'A2' through X'A9'	X'E2' through X'E9'

If FILL=NULL is specified for any MFLD in a segment defined as GRAPHIC=YES, the hexadecimal character X'3F' is compressed out of the segment. If GRAPHIC=NO and FILL=NULL are specified in the SEG statement, any X'3F' in the non-graphic data stream is compressed out of the segment and undesirable results might be produced. Non-graphic data should be sent on output as fixed length output fields and the use of FILL=NULL is not recommended in this case.

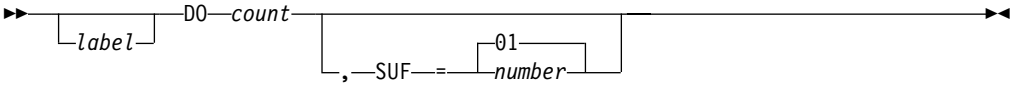
For MSG TYPE=OUTPUT, the GRAPHIC= keyword applies only for DPM. It specifies whether (YES) or not (NO) nongraphic control characters (X'00' to X'3F') in the data from the IMS application program are to be replaced by blanks. The default value is YES. If NO is specified, MFS allows any bit string received from an IMS application program to flow unmodified through MFS to the remote program.

Restriction: When GRAPHIC=NO is specified, IMS application programs using Options 1 and 2 cannot omit segments in the middle of an LPAGE, or truncate or omit fields in the segment using the null character (X'3F').

DO Statement

The DO statement causes repetitive generation of MFLD statements between the DO and ENDDO statements. DO is optional, but a message that includes a DO must include a subsequent ENDDO.

Format:



Parameters:

label
A one- to eight-character alphanumeric name can be specified. It is not used.

count
Specifies how many times to generate the following MFLD statements. The maximum *count* that can be specified is 99; if more than 99 is specified, the 2 rightmost digits of the specified *count* are used (for example, 03 would be used if 103 were specified) and an error message is issued.

SUF=
Specifies the 1- or 2-digit suffix to be appended to the MFLD *label* and *dfldname* of the first group of generated MFLD statements. The default is 01. MFS increases the suffix by 1 on each subsequent generation of statements.

If the specified suffix exceeds 2 digits, MFS uses the rightmost 2 digits.

If the specified *count* is such that the generated suffix eventually exceeds 2 digits, MFS reduces the *count* to the largest legitimate value. For example, if *count* equals 8 and SUF=95, invalid suffixes of 100, 101, and 102 would result. In this instance, MFS reduces *count* to 5, processes the statement, and issues an error message.

Printing Generated MFLD Statements

The generated MFLD statements can be printed in a symbolic source format by specifying COMP in the parameter list of the EXEC statement. This provides a means of seeing the results of the MFLD statement generation without having to interpret the intermediate text blocks.

The following items are printed for each generated MFLD statement:

- The generated statement sequence number followed by a + (plus sign) to indicate that the MFLD statement was generated as a result of DO statement processing.
- The MFLD statement label, if present, including the appended suffix.
- The statement operator, MFLD.
- *dfldname*, if present, including the appended suffix.
- For ECGS literals, the G, SO, and SI is not present. Literals are truncated if there is insufficient room to print all specifications. Truncation is indicated by a portion of the literal followed by an ellipsis (...) representing the truncated portion.
- The system literal name, if present.

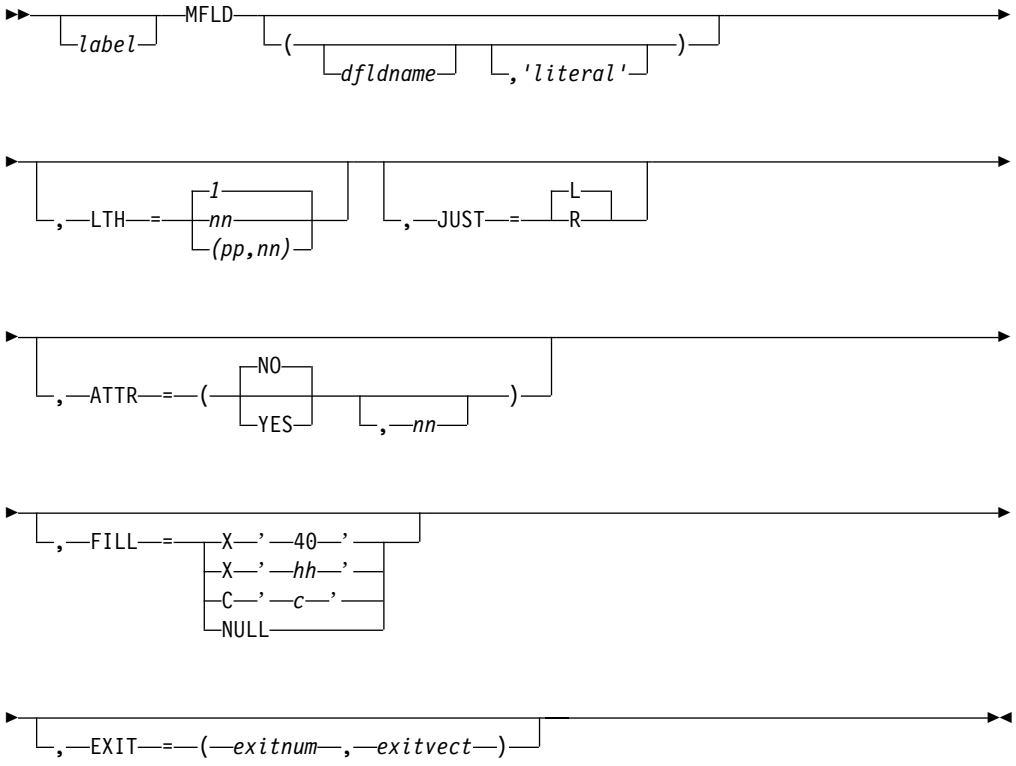
  If both *dfldname* and a literal are present, they are enclosed in parentheses.
- (,SCA), if present.
- The field length, in the form LTH=*nnnn* (or LTH=(*pppp,nnnn*), if present).
- JUST=L or R, if present.
- ATTR=YES, if present.
- ATTR=*nn*, if present.

No other operands are printed, even if specified on the source MFLD statement.

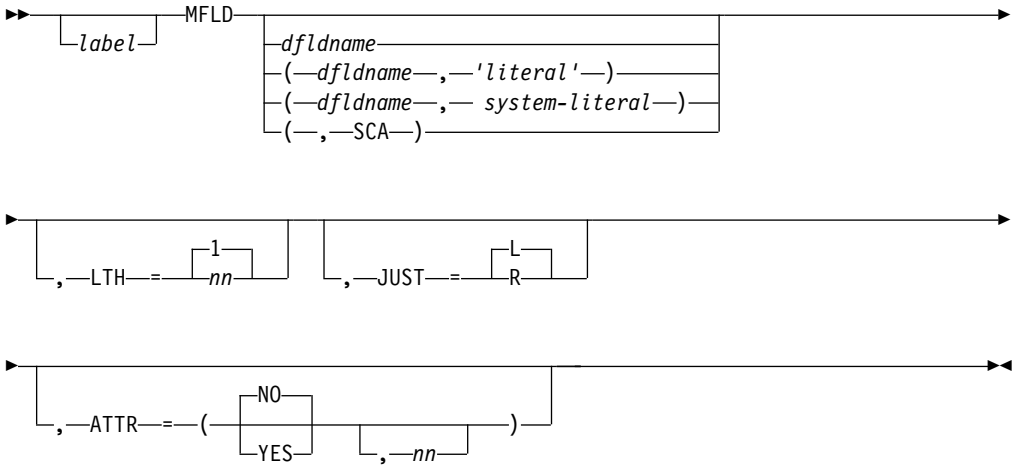
MFLD Statement

The MFLD statement defines a message field as it will be presented to an application program as part of a message output segment. At least one MFLD statement must be specified for each MSG definition.

Format for MSG TYPE=INPUT:



Format for MSG TYPE=OUTPUT:



Parameters:

label

A one-to eight-character alphanumeric name can be specified. *label* is required if it is referred to in the COND operand of the previous LPAGE statement. It can be used to uniquely identify this statement. If the MFLD is between the DO and ENDDO statements, *label* is restricted to 6 characters or less. DO statement processing appends a 2-digit suffix (a sequence number, 01 to 99) to the label and prints the label as part of the generated MFLD statement. If *label* is more than 6 characters and iterative generation is used, the label is truncated at 6

characters, and the 2-digit sequence number is added to make the 8-character name. No error message is issued if this occurs.

dfldname
Specifies the device field name (defined via the DEV or DFLD statement) from which input data is extracted or into which output data is placed. If this parameter is omitted when defining a message output control block, the data supplied by the application program is not displayed on the output device. If the repetitive generation function of MFS is used (DO and ENDDO statements), *dfldname* should be restricted to 6 characters maximum length. When each repetition of the statement is generated, a 2-character sequence number (01 to 99) is appended to *dfldname*. If the *dfldname* specified here is greater than 6 bytes and repetitive generation is used, *dfldname* is truncated at 6 characters and a 2-character sequence number is appended to form an 8-character name. No error message is provided if this occurs. This parameter can be specified in one of the following formats:

dfldname
Identifies the device field name from which input data is extracted or into which output data is placed.

'*literal***'**
Can be specified if a literal value is to be inserted in an input message.

(*dfldname***,'***literal***')**
If TYPE=OUTPUT, this describes the literal data to be placed in the named DFLD. When this form is specified, space for the literal must not be allocated in the output message segment supplied by the application program.

If TYPE=INPUT, this describes the literal data to be placed in the message field when no data for this field is received from the device. If this *dfldname* is used in the PFK parameter of a DEV statement, this literal is always replaced by the PF key literal or control function. However, when this *dfldname* is specified in the PFK parameter, but the PF key is not used, the literal specified in the MFLD statement is moved into the message field. When physical paging is used, the literal is inserted in the field but is not processed until after the last physical page of the logical page has been displayed.

In both cases, if the LTH= operand is specified, the length of the literal is truncated or padded as necessary to the length of the LTH= specification. If the length of the specified literal is less than the defined field length, the literal is padded with blanks if TYPE=OUTPUT and with the specified fill character (FILL=) if TYPE=INPUT. If no fill character is specified for input, the literal is padded with blanks (the default). The length of the literal value cannot exceed 256 bytes.

(*dfldname***,***system-literal***)**
Specifies a name from a list of system literals. A system literal functions like a normal literal except that the literal value is created during formatting prior to transmission to the device. The LTH=, ATTR=, and JUST= operands cannot be specified. When this form is specified, space for the literal must not be allocated in the output message segment supplied by the application program.

The system literals and their associated lengths and formats are shown in Table 42 on page 326.

Table 42. Lengths and Formats of System Literals

	Produces Literal of:		
System Literal Name	**Length**	**Format**	**Notes**
LTSEQ	5	*nnnnn*	1
LTNAME	8	*aaaaaaaa*	1
TIME	8	HH:MM:SS	
DATE1 or YYDDD	6	YY.DDD	
DATE2 or MMDDYY	8	MM/DD/YY	
DATE3 or DDMMYY	8	DD/MM/YY	
DATE4 or YYMMDD	8	YY/MM/DD	
DATE1Y4 or YYYYDDD or DATEJUL	8	YYYY.DDD	
DATE2Y4 or MMDDYYYY or DATEUSA	10	MM/DD/YYYY	
DATE3Y4 or DDMMYYYY or DATEEUR	10	DD/MM/YYYY	
DATE4Y4 or YYYYMMDD or DATEISO	10	YYYY/MM/DD	
LPAGENO	4	*nnnn*	2
LTMSG	14	MSG WAITING Q*x*	3

Table 42. Lengths and Formats of System Literals (continued)

	Produces Literal of:		
System Literal Name	**Length**	**Format**	**Notes**

Notes:

1. LTSEQ is the output message sequence number for the logical terminal. The value created is the logical terminal dequeue count plus 1. The first output message after an IMS cold start or `/NRESTART BUILDQ` has a sequence number of 00001. Certain IMS-created messages do not change this number.

   LTNAME is the logical terminal (LTERM) name of the LTERM for which this message is being formatted.

   Messages generated by the IMS control region in response to terminal input (error messages, most command responses) do not have an LTSEQ or an LTNAME. These messages use the IMS message output descriptor DFSMO1. In these instances, the values provided are 00000 and blanks, respectively.
2. LPAGENO specifies that the current logical page number of the message be provided as a system literal. This number corresponds to the page number you entered for an operator logical page request. The literal produced is a 4-digit number with leading zeros converted to blanks.
3. LTMSG specifies that when this output message is sent to the terminal, the literal 'MSG Waiting Q*x*' (where *x* is message queue number 1, 2, 3, or 4) is sent in the LTMSG field if there are messages in the queue for the terminal. If there are no messages in the queues, other than the current queue, blanks are sent in the LTMSG field.

   Usually the message waiting is sent when the current message is dequeued. If the message is waiting in Q1, it is sent. If the message is in Q2 and the terminal is in exclusive mode, it is sent (when any other messages from Q1 are sent). If the message is in Q2 and conversational status does not prevent it from being sent or if the message is in Q3 or Q4 and the exclusive or conversational status does not prevent it from being sent, it is sent. If a message is waiting to be sent on another queue and the terminal is in conversation, the conversation can be held to view the message; if the terminal is in exclusive mode, the message can be viewed when the terminal is taken out of exclusive mode. If you are entering response mode transactions, the message can be viewed before entering response mode transaction input from the terminal.

   This system literal is recommended for conversational mode. It is not recommended for ISC subsystems.

(,SCA)
Defines this output field as the system control area which is not displayed on the output device. There can be only one such field in a logical page (LPAGE) and it must be in the first message segment of that page. If no logical pages are defined, only one SCA field can be defined and it must be in the first segment of the output message. This specification is valid only if TYPE=OUTPUT was specified on the previous MSG statement.

LTH=
Specifies the length of the field to be presented to an application program on input or received from an application program on output. Default or minimum value is 1. Maximum value is 8000. (The maximum message length must not exceed 32767.)

The form (*pp*,*nn*) can be used when defining an input field; however, a field name must be specified in the first positional parameter if the (*pp*,*nn*) form is used. The value supplied for *pp* specifies which byte in the input data field is to be considered the first byte of data for the message field. For example, a *pp* of 2 specifies that the first byte of input data is to be ignored, and the second byte

becomes the first byte of this field. The value of *pp* must be greater than or equal to 1. The value supplied for *nn* specifies the length of the field to be presented to an application program.

If (,SCA) is specified in the positional parameter, the specified LTH= value must be at least 2.

LTH= can be omitted if a literal is specified in the positional operand (TYPE=INPUT), in which case, length specified for *literal* is used. If LTH= is specified for a literal field, the specified literal is either truncated or padded with blanks to the specified length. If the MFLD statement appears between a DO and an ENDDO statement, a length value is printed on the generated MFLD statement, regardless of whether LTH= is specified in the MFLD source statement.

JUST=
Specifies that the data field is to be left-justified (L) or right-justified (R) and right- or left- truncated as required, depending upon the amount of data expected or presented by the device format control block. The default is L.

ATTR=
Specifies whether (YES) or not (NO) the application program can modify the 3270 attributes and the extended attributes (*nn*).

If YES, 2 bytes must be reserved for the 3270 attribute data to be filled in by the application program on output and to be initialized to blanks on input. These 2 bytes must be included in the LTH= specification.

The value supplied for *nn* is the number of extended attributes that can be dynamically modified. The value of *nn* can be a number from 1 to 6. An invalid specification will default to 1. Two additional bytes per attribute must be reserved for the extended attribute data to be filled in by the application program on output and to be initialized to blanks on input. These attribute bytes must be included in the MFLD LTH= specification.

Example: Shown below are valid specifications for ATTR= and the number of bytes that must be reserved for each different specification:

```
MFLD    ,ATTR=(YES,nn)
        2 + (2 × nn)

MFLD    ,ATTR=(NO,nn)
        2 × nn

MFLD    ,ATTR=(nn)
        2 × nn

MFLD    ,ATTR=YES
        2

MFLD    ,ATTR=NO
        0
```

ATTR=YES and *nn* are invalid if a literal value has been specified through the positional parameter in an output message.

The attributes in a field sent to another IMS ISC subsystem are treated as input data by MFS regardless of any ATTR= specifications in the format of the

receiving subsystem. For example, a message field (MFLD) defined as ATTR=(YES,1),LTH=5 would contain the following:

```
00A0C2F1C8C5D3D3D6
```

If the MFLD in the receiving subsystem is defined as LTH=9 and without ATTR=, the application program receives:

```
00A0C2F1C8C5D3D3D6
```

If the MFLD in the receiving subsystem is defined as LTH=13 and ATTR=(YES,1), the application program receives:

```
4040404000A0C2F1C8C5D3D3D6
```

If the MFLD in the receiving subsystem is defined as LTH=5 and ATTR=(YES,1), the application program receives:

```
4040404000A0C2F1C8
```

The input SEG statement should be specified as GRAPHIC=NO to prevent translation of the attribute data to uppercase.

FILL=

Specifies a character to be used to pad this field when the length of the data received from the device is less than the length of this field. This character is also used to pad when no data is received for this field (except when MSG statement specifies option 3.) This operand is only valid if TYPE=INPUT. The default is X'40'.

X'*hh***'**

Character whose hexadecimal representation is *hh* is used to fill fields. FILL=X'3F' is the same as FILL=NULL.

C'*c***'**

Character *c* is used to fill fields.

NULL

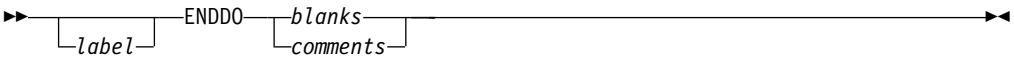
Causes compression of the message segment to the left by the amount of missing data in the field. Refer to “Cursor Position Input and FILL=NULL” on page 191 for more information.

EXIT=

Describes the field edit exit routine interface for this message field. The exit routine number is specified in *exitnum*, and *exitvect* is a value to be passed to the exit routine when it is invoked for this field. The value of *exitnum* can range from 0 to 127. The value of *exitvect* can range from 0 to 255. The address of the field as it exists after MFS editing, (but before NULL compression for option 1 and 2), is passed to the edit exit routine, along with the vector defined for the field. (If NOFLDEXIT is specified for a DPM device, the exit routine will not be invoked.) The exit routine can return a code with a value from 0 to 255. MFS maintains the highest such code returned for each segment for use by the segment edit routine. EXIT= is invalid if '*literal*' is specified on the same MFLD statement.

ENDDO Statement

The ENDDO statement terminates the group of MFLD statements that are to be repetitively generated. The generated MFLD statements are printed immediately following the ENDDO statement. ENDDO is required when a DO statement has been specified.



label
A one- to eight-character alphanumeric name can be specified. It is not used.

MSGEND Statement

The MSGEND statement terminates a message input or output definition and is required as the last statement in the definition. If this is the end of the job submitted, it must also be followed by an END compilation statement.

label
a one- to eight-character alphanumeric name can be specified. It is not used.

Format Definition Statements

FMT Statement

The FMT statement initiates and names a format definition that includes one or more device formats differing only in the device type and features specified in the DEV statement. Each device format included in the format definition specifies the layout for data sent to or received from a device or a remote program.

Format:

Parameters:

label
A required one- to six-character alphanumeric name that is referred to by message descriptors in the SOR= operand of MSG statements.

The name specified for *label* becomes part of the member name used for the resulting device output format and device input format blocks that are stored in the IMS.FORMAT library.

If DEV TYPE=DPM-An, and DIV OPTIONS=MSG, the name specified for *label* is sent to the remote program as the data name in the output message header.

If DEV TYPE=DPM-Bn, and DIV OPTIONS=(MSG,DNM), the name specified for *label* is sent to the other subsystem as the data structure name in the DD FM header.

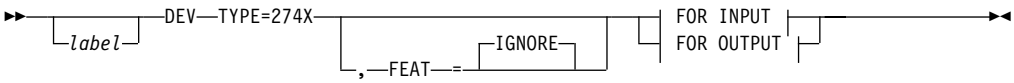
DEV Statement

The DEV statement defines device characteristics for a specific device or data formats for a specific device type. The DFLD statements following this DEV statement are mapped using the characteristics specified until the next DEV or

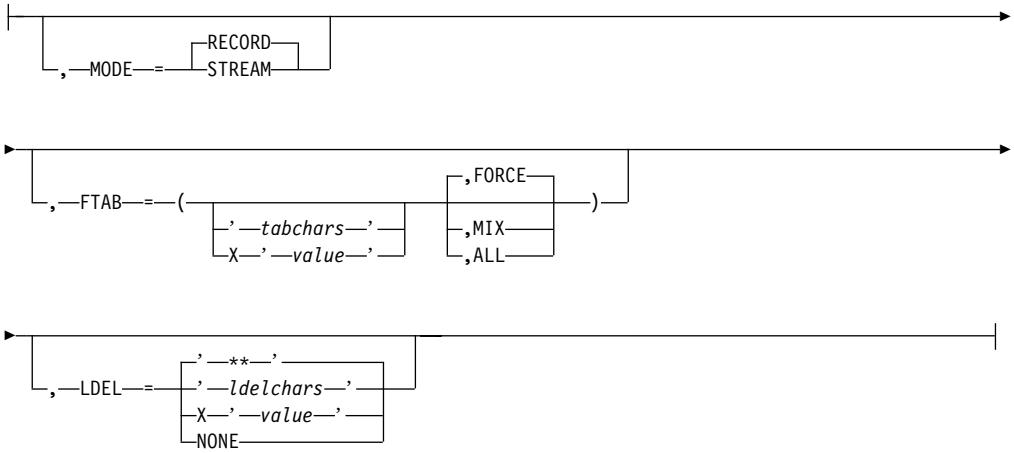
FMTEND statement is encountered. For DPM devices, the DEV statement specifies the DPM program type number and (optionally) a feature set number.

Recommendation: Read the TYPE= operand description before using the DEV statement.

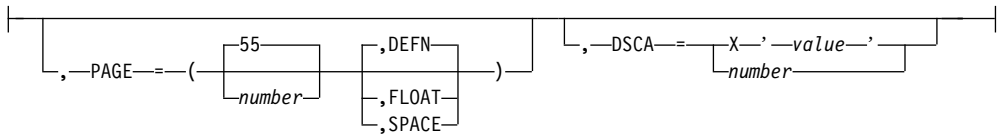
Format for 2740 or 2741:



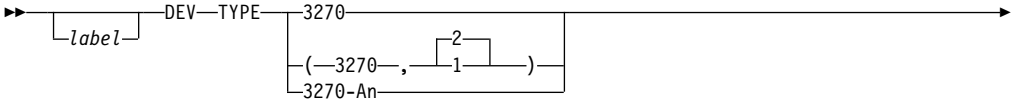
FOR INPUT:



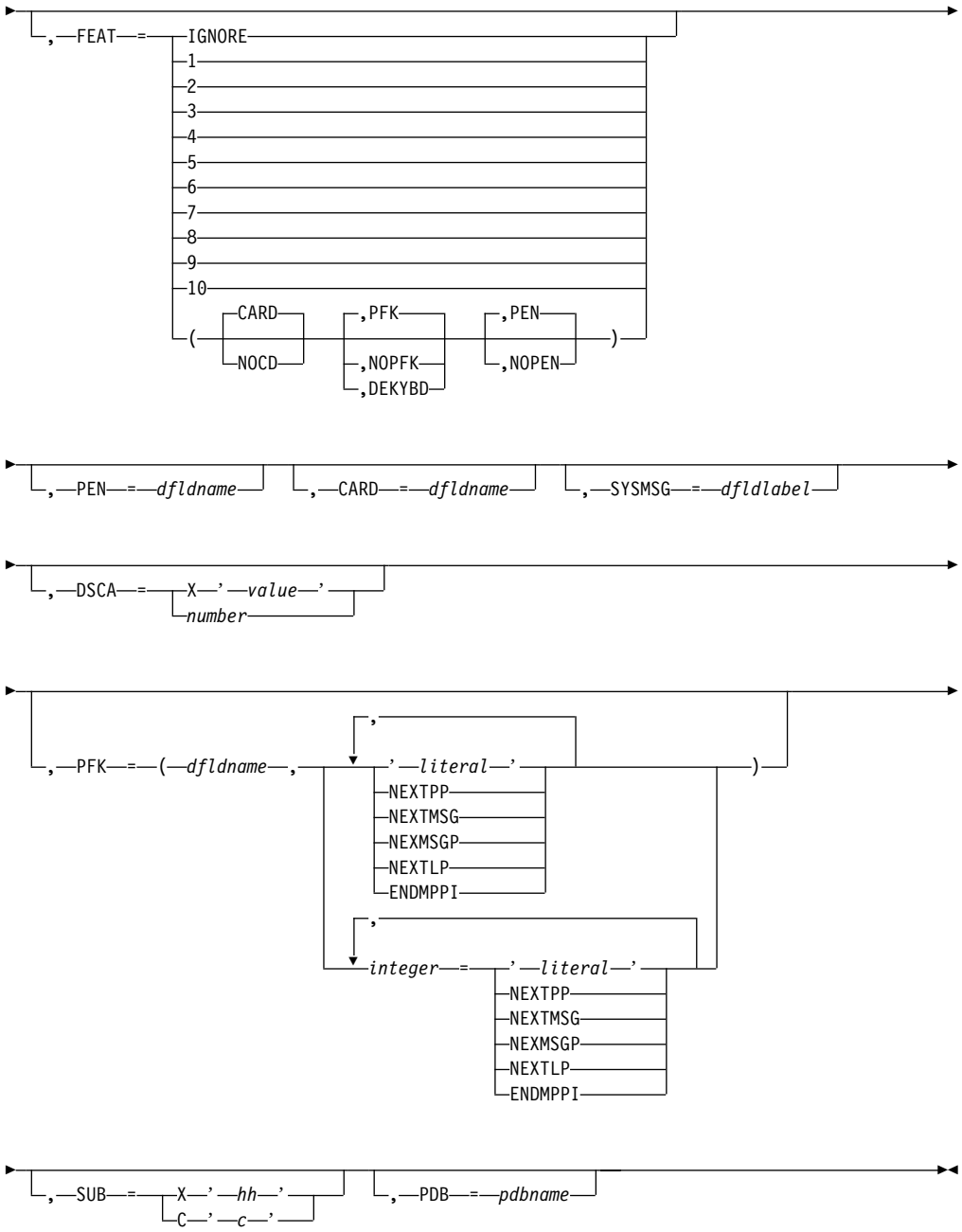
FOR OUTPUT:



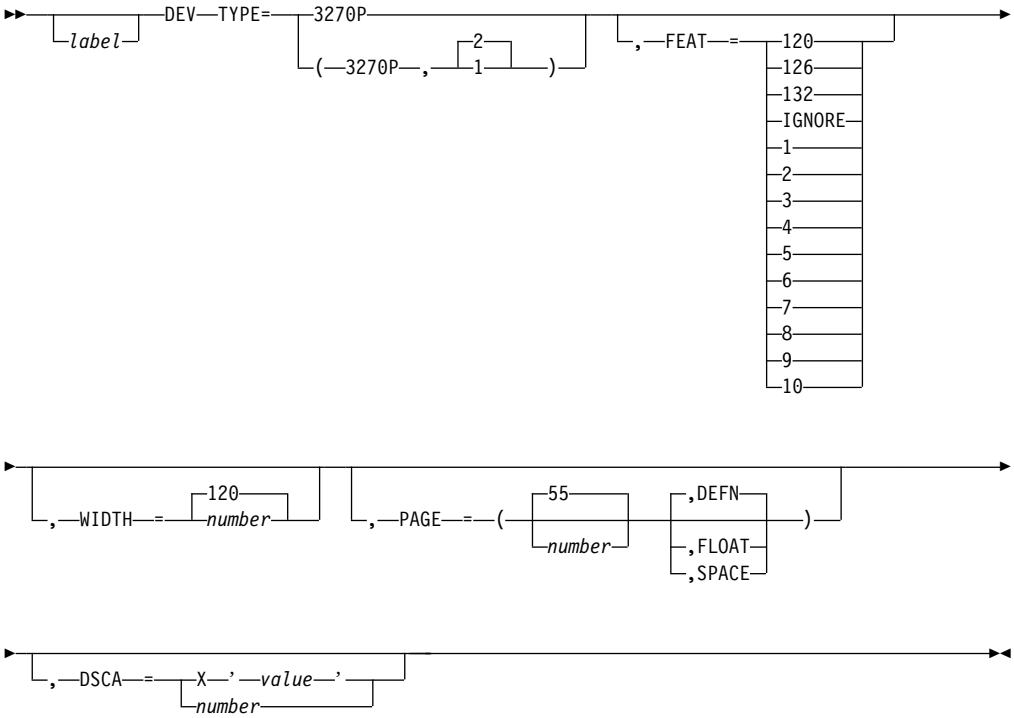
Format for 3270 Display:



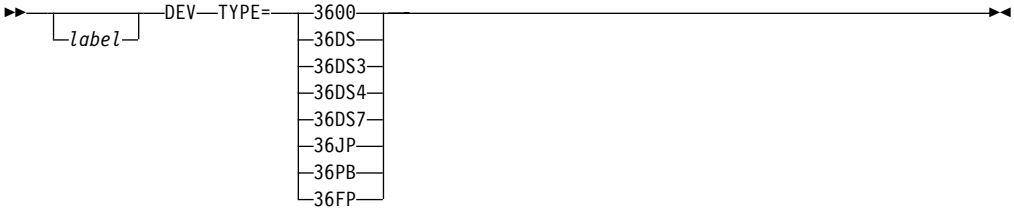
Format Definition Statements: DEV



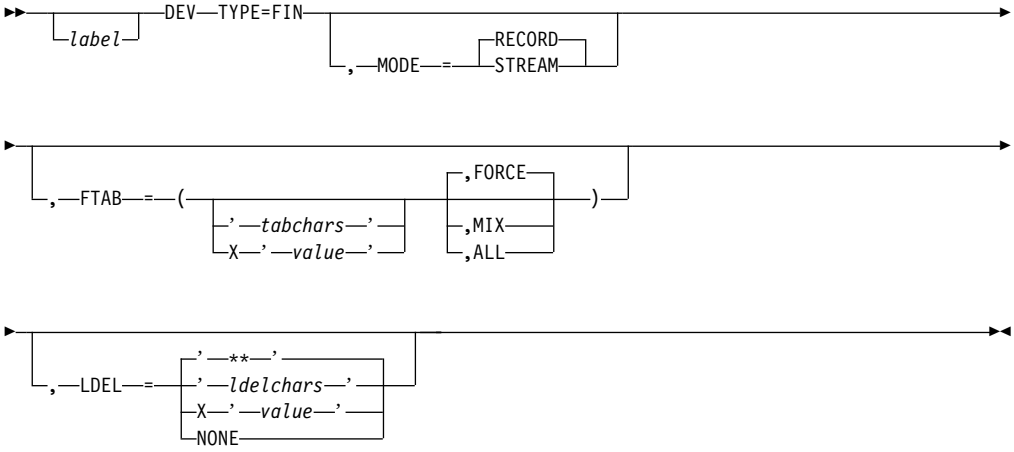
Format for 3270 Printers:



Format for Finance Workstations (3600 OR 4700):

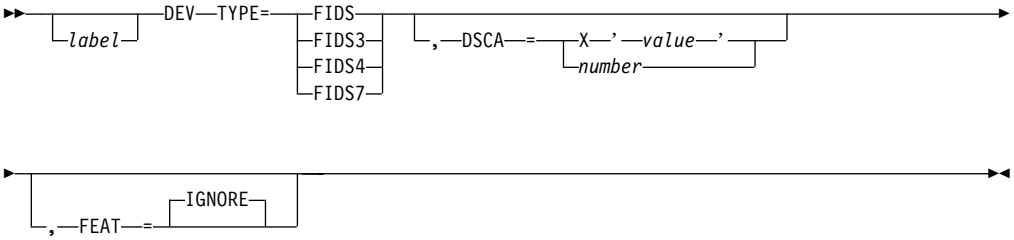


Format for DEV TYPE=FIN:

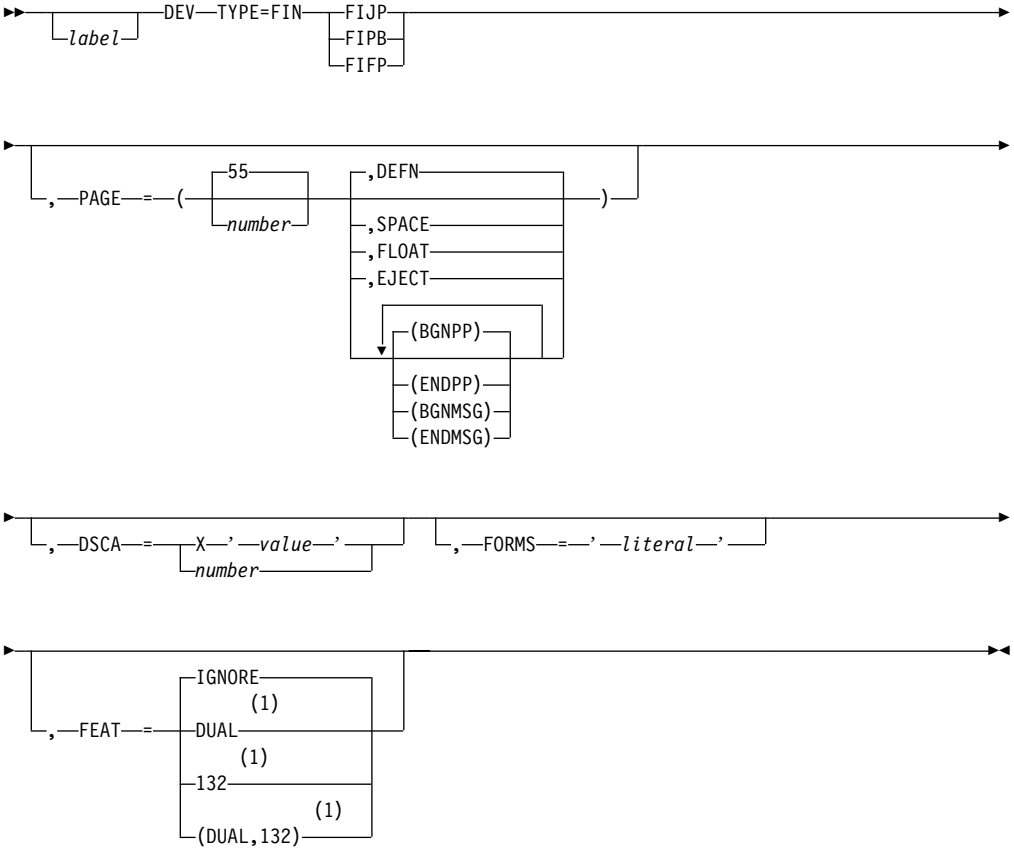


Format Definition Statements: DEV

Format for DEV TYPE=FIDS, FIDS3, FIDS4, FIDS7:



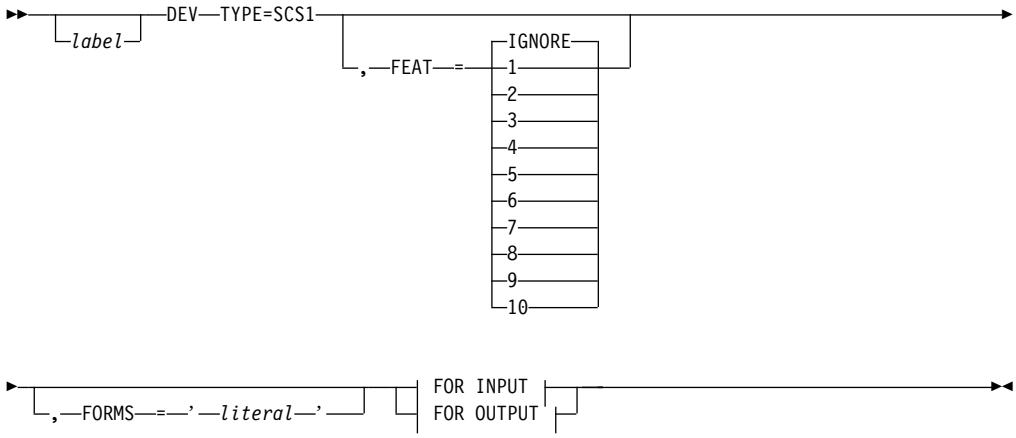
Format for DEV TYPE=FIJP, FIPB, FIFP:



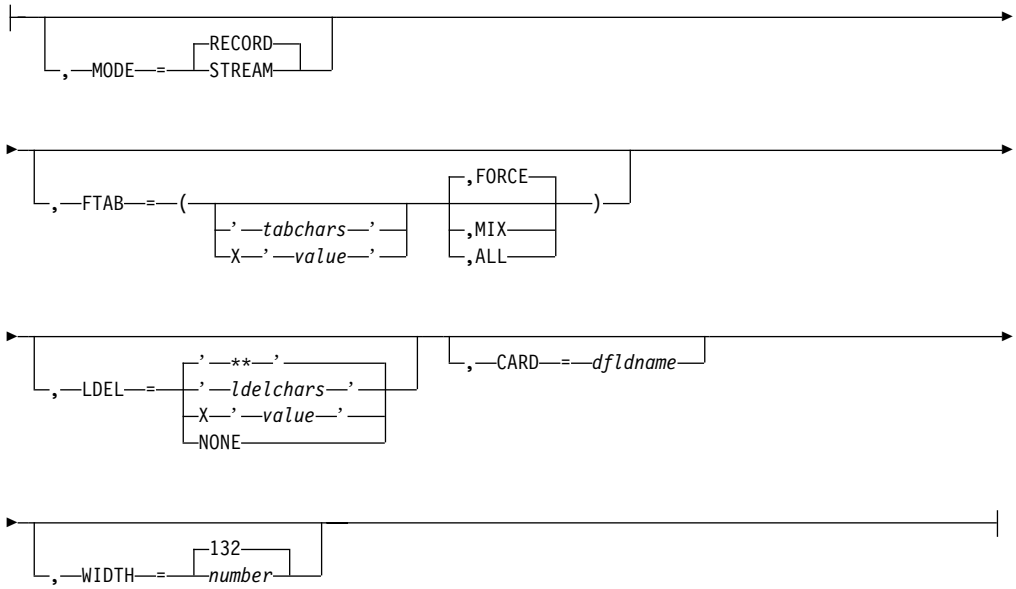
Notes:

1 FIFP only

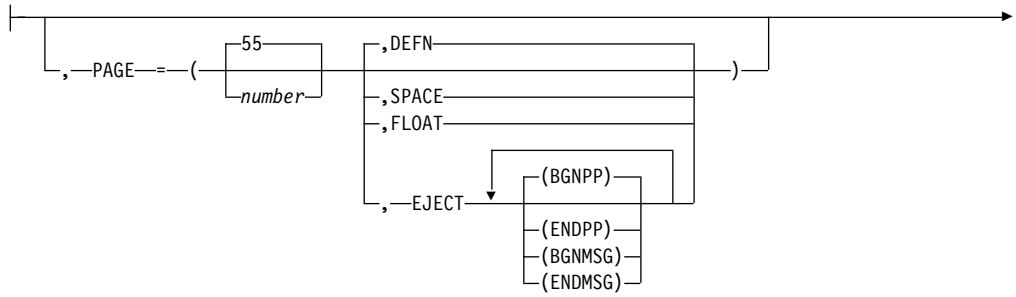
Format for SCS1:

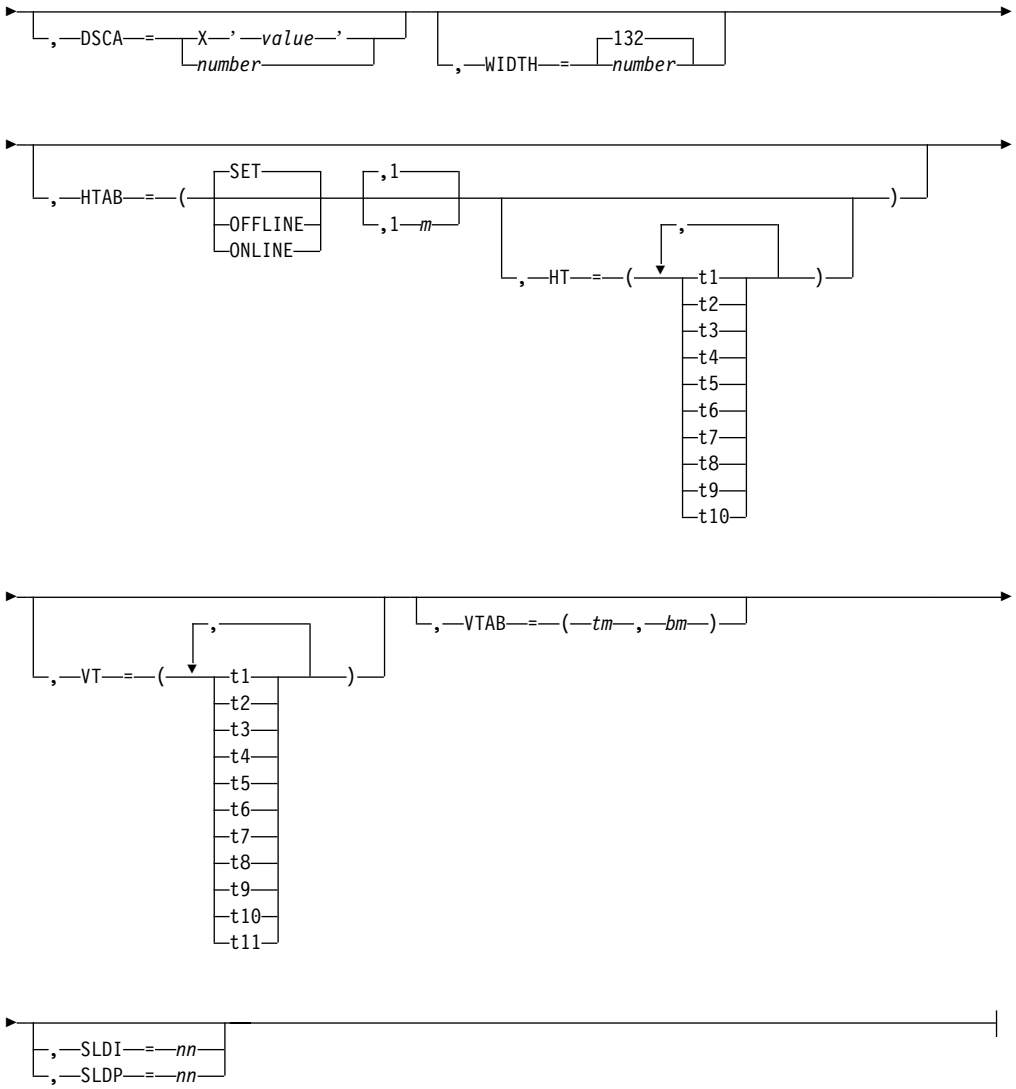


FOR INPUT:

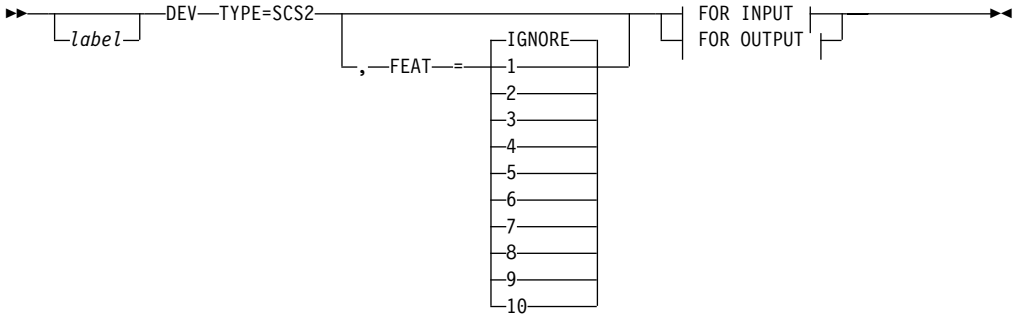


FOR OUTPUT:

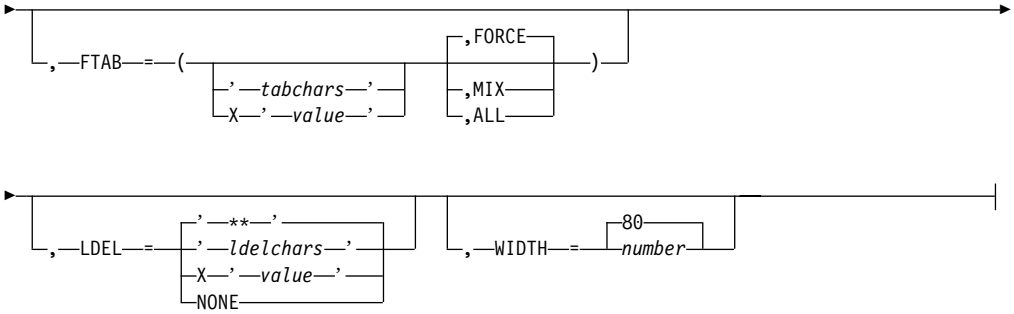




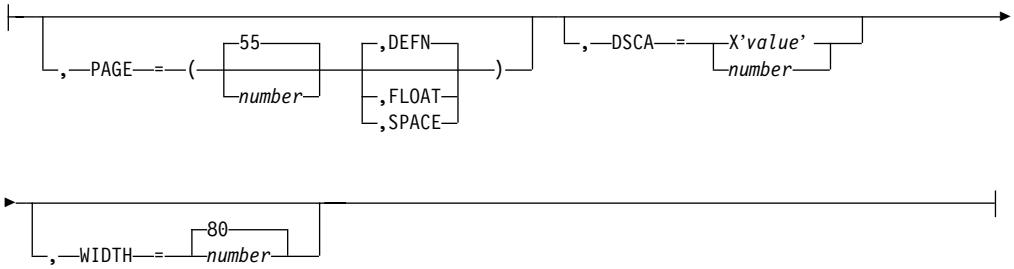
Format for SCS2:



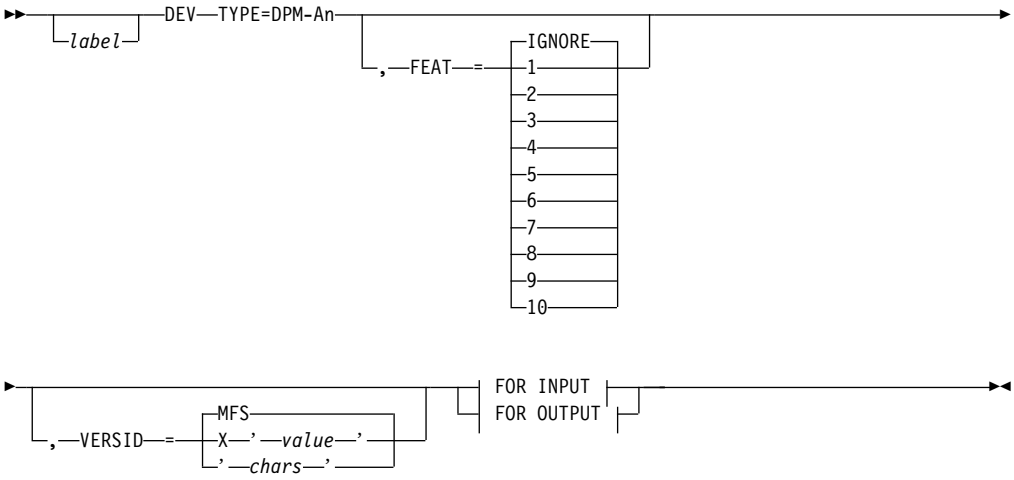
FOR INPUT:



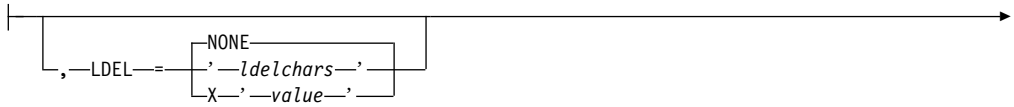
FOR OUTPUT:

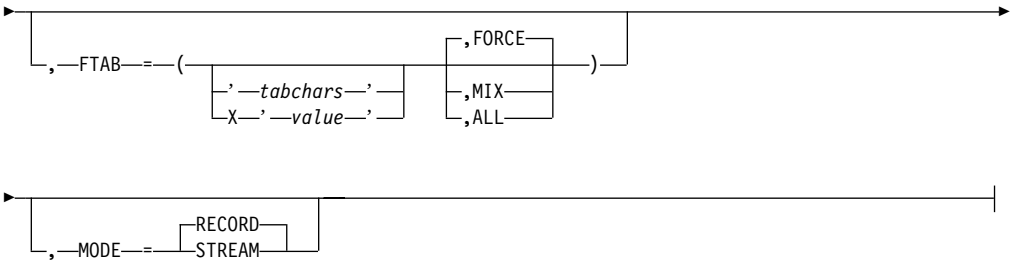


Format for DPM-An:

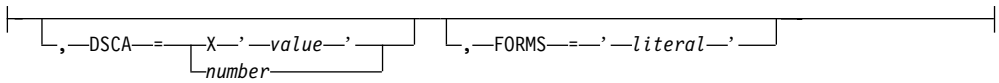


FOR INPUT:

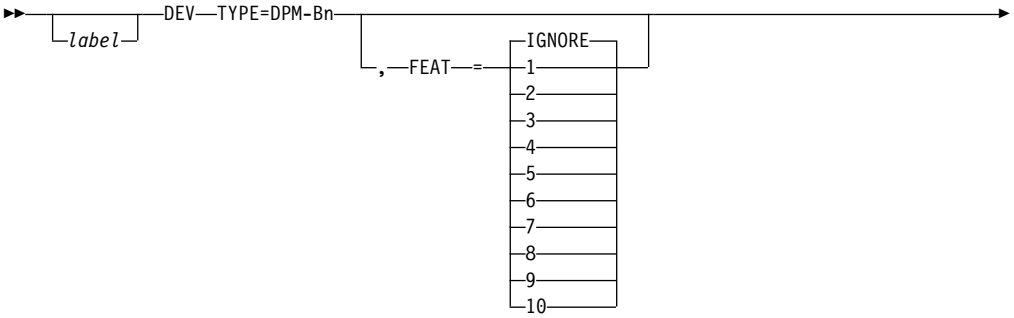




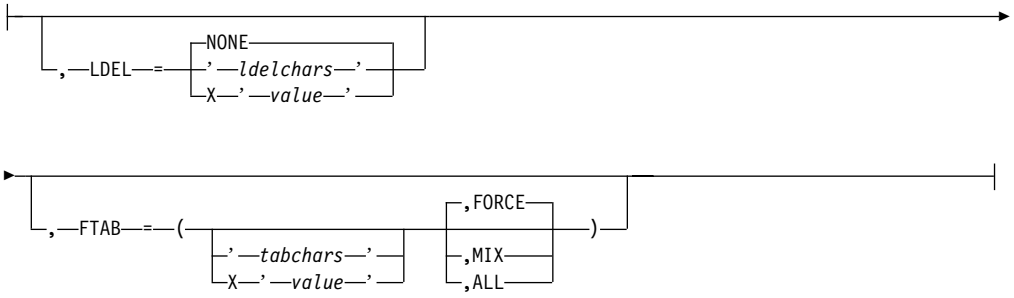
FOR OUTPUT:



Format for DPM-Bn:



FOR INPUT:



FOR OUTPUT:

Parameters:

label
An optional one- to eight-character alphanumeric name that uniquely identifies this statement.

TYPE=
Specifies the device type and model number of a device using this format description. The 3284-3 printer attached to a 3275 is supported only as TYPE=3270P. The model number specified when defining a format for a 3284-3 is the model number of the associated 3275.

TYPE=3270-An specifies a symbolic name for 3270 and SLU 2 displays with the screen size defined during IMS system definition, feature numbers *n*=1-15. This specification causes the MFS Language utility to read the MFS device characteristics table (DFSUDT0x) to extract the screen size.

TYPE=DPM-Bn specifies the device as an ISC node. The device type specified by *n* must agree with the specification of the component (COMPT=) on the system definition TERMINAL macro.

Based on the device and model used, specify:

TYPE=	Device-Model
274X	2740-1, 2741-1, or 2740-2
3270, 1	3275-1
	3276-1,11 (defined at IMS system definition as 3270 model 1)
	3277-1
	3278-1 (defined at IMS system definition as 3277 model 1)
	SLU 2 (480 characters)
3270,2	3275-2 SLU 2 (1920 characters)
	(any display defined during IMS system definition as 'mod 2' with screen area of 1920 characters)
3270-An	3270-An (applies to any 3270 or SLU 2 display defined as TYPE=3270-An during IMS system definition)

Examples of 3270 devices that can be defined as 3270-An and the recommended standard of associating screen sizes with the device type symbolic name follow:

Device	Screen size and definition
3180	24×80 screen size defined as 3270-A2
327X-1,11	12×80 screen size defined as 3270-A1
327X-2,12	24×80 screen size defined as 3270-A2
327X-3,13	32×80 screen size defined as 3270-A3

	327X-4,14	43×80 screen size defined as 3270-A4
	3278-5	27x132 screen size defined as 3270-A7
	3290	62x160 screen size defined as 3270-A8 or 24×80 screen size defined as 3270-A2
	5550	3270 Kanji Emulation or 3270 PC with 24×80 screen size defined as 3270-A2
3270P,1	3284-1 3286-1 3287 (with 480 character print feature and not attached as SLU 1 or SLU 4) 3289 (with 480 character print feature and not attached as SLU 1 or SLU 4)	
3270P,2	3284-2 3286-2 3287 (with 1920 character print feature and not attached as SLU 1 or SLU 4) 3289 (with 1920 character print feature and not attached as SLU 1 or SLU 4)	
FIN	Finance application program (input only)	
FIDS	Finance display component (6×40; for example, 3604-1 or -2)	
FIDS3	Finance display component (12×40; for example, 3604-3)	
FIDS4	Finance display component (16×64; for example, 3604-4)	
FIDS7	Finance display component (24×80; for example, 3604-7)	
FIJP	Finance journal printer	
FIPB	Finance passbook printer	
FIFP	Finance administrative printer	
SCS1	The following console keyboard printers: NTO 3771 3773 3774 3775 3776 5553 5557 SLU 1 (with a print data set or bulk printer) SLU 4 3289 and 3287 when attached to IMS as SLU 1	
SCS2	3521 card punch 3501 card reader	

2502 card reader

SLU 1 (transmit data set)

SLU 4

DPM-An SLU P (n is value 1-15)

DPM-Bn ISC (n is value 1-15)

MODE=

Specifies the manner in which field scanning is to occur. Default value is RECORD. MODE= is valid for DPM-An input only, and for DPM-Bn input and output. For DPM-Bn, if the input and output modes are not the same, each DIV statement must be preceded by a DEV statement.

RECORD

Specifies that fields are defined as occurring within specific records (a line from a device, a transmission from a remote program) that is transmitted from the device or program. For DPM-Bn, Record mode must be specified for variable length, variable blocked (VLVB) format records.

STREAM

Specifies that fields are defined as a contiguous stream of fields—record boundaries do not affect the MFS scan. Fields can be split across records and fields can be entered from any record provided they are entered in the defined sequence. For DPM-Bn, Stream mode must be specified for chained request/response units (RUs).

FTAB=

Specifies the field tab (FTAB) characters that you or a remote program can use to terminate an input field when either the length of the data entered is less than the defined field length, or no data for the field exists:

- For FIN, DPM-An, and DPM-Bn, a maximum of eight FTAB characters or 16 hexadecimal digits can be specified, and at least one character (or two hexadecimal digits) should be specified.
- For SCS1, up to four FTAB characters or eight hexadecimal digits can be specified; the characters NL, LF, HT, and VT are always FTAB characters and do not need to be specified.
- For SCS2, up to three FTAB characters or six hexadecimal digits can be specified. The characters NL, CR, LF, HT, and VT are always FTAB characters and do not have to be specified; however, they are received by MFS only if the Hollerith code is punched in the card if the input is from the card reader.

If no FTAB characters are defined, each device input field is considered to be of its defined length. In Record mode, when the end of a record is reached, the current field is terminated and all subsequent fields defined for that record are processed with no device data (message fill). In Stream mode, all transmissions that comprise the input message are treated as a stream of data fields unaffected by transmission boundaries. If FTABs are not defined or are not used for DPM input, each input field is considered to be of defined length except when NULL=DELETE is specified. With NULL=DELETE, if trailing nulls are encountered in a field or an entire field is null, the field is padded to defined length using the message fill character.

If FTAB characters are defined in this operand, either FORCE, MIX, or ALL can also be specified. The default is FORCE.

FORCE
Specifies that an FTAB is not required until you or a remote program enters an FTAB character. In record mode, if an FTAB is used for one field, the remaining fields of the current record must be terminated with an FTAB, regardless of length. In stream mode, if an FTAB is used for one field, the remaining fields in the message must be terminated with an FTAB.

MIX
Specifies that an FTAB is never required but can be used to terminate any input field when data is less than the defined field length.

ALL
Specifies that an FTAB must be used to terminate all fields, regardless of length, except for certain mode (MODE=) dependent conditions. In record mode, an FTAB is not required for the last field defined or entered in the record. In stream mode, an FTAB is not required for the last field defined or entered in the message.

LDEL=
Specifies two characters or four hexadecimal digits, which, if entered as the last two characters of a record of input data, cause the record to be discarded. A specification of NONE causes IMS to bypass record delete processing, except for the first record, which is always deleted if the last two characters are asterisks (**). NONE is the default for DPM devices. For other devices, the default is **.

PAGE=
Specifies output parameters as follows:

number
For printer devices, number defines the number of print lines on a printed page; for card devices, number defines the number of cards to be punched per DPAGE or physical page (if *pp* parameter is used in the DFLD statements). This value is used for validity checking. The number specified must be greater than or equal to 1 and less than 256. The default is 55.

If VTAB= is specified for SCS1 printers, then the minimum value for PAGE= is 3.

DEFN
Specifies that lines/cards are to be printed/punched as defined by DFLD statements (no lines/cards are to be removed or added to the output page).

SPACE
Specifies that each output page contains the exact number of lines/cards specified in the *number* parameter.

FLOAT
Specifies that lines/cards with no data (all blank or NULL) after formatting are to be deleted.

For 3270P and SCS1 devices, some lines having no data (that is, all blank or null) must not be deleted under the following circumstances:

- The line contains one or more set line density (SLDx=) specifications.
- A field specified as having extended attributes spans more than one line.

EJECT
Specifies that a forms eject operation should be performed for printer devices. EJECT is valid only when TYPE=FIJP, FIPB, FIFP, or SCS1. If

EJECT is specified for SCS1, MFS assumes the Vertical Forms Control feature is present. The default for the sublist is BGNPP.

The sublist specifies when ejects are to be performed:

BGNPP
Specifies that an eject is to be performed before each physical page of output.

ENDPP
Specifies that an eject is to be performed after each physical page is printed.

BGNMSG
Specifies that an eject is to be performed before any data in the message is printed.

ENDMSG
Specifies that an eject is to be performed after all message data is printed.

DSCA=
Specifies a default system control area (DSCA) for output messages using this device format. The DSCA supersedes any SCA specified in a message output descriptor if there are conflicting specifications. Normally, the functions specified in both SCAs are performed. If the DSCA= operand is specified for SCS1 or SCS2, it is ignored. If the DSCA= operand is specified for 3270P, it is ignored, except for the bit setting for “sound device alarm”. If this bit is specified on the DSCA/SCA option, it is sent to the device. For TYPE=DPM-An or DPM-Bn, DSCA/SCA information is sent to a remote program or ISC subsystem only if a DFLD definition requests it.

The value specified here must be a decimal number not exceeding 65535 or X'*hhhh*'. If the number is specified, the number is internally converted to X'*hhhh*'.

The two bytes of the DSCA field should be defined as shown in Table 43 or Table 45 on page 344.

Table 43 shows the DSCA bit settings for 3270 display or SLU 2 devices or TYPE=DPM-An or DPM-Bn.

Table 43. Bit Settings for DSCA Field. For 3270 Display, SLU 2 Devices, TYPE=DPM-An, or DPM-Bn

Byte	Bit	
0	0-7	Should be 0.
1	0	Should be 1.
	1	Force format write (erase device buffer and write all required data).
	2	Erase unprotected fields before write.
	3	Sound device alarm.
	4	Copy output to candidate pointer. Bits 1-4 are ignored for DPM-Bn.
	5	B'0'- For 3270, protect the screen when output is sent. For DPM, demand paging can be performed. B'1'- For 3270, do not protect the screen when output is sent. For DPM-B, autopaging can be performed.
	6-7	Should be 0, except for the 3290 in partitioned format mode.

If byte 1 bit 5 is set to B'1' (unprotect screen option) for a 3275 display, and both input and output occur simultaneously (contention), the device is disconnected. For non-3275 devices, the SCA option is ignored. If byte 1 bit 5 is set to B'0', the application program can request autopaged output by setting the SCA value to B'1'. This request is honored only if present in the first segment of the first LPAGE of the output message.

If a nonzero value is specified for byte 0, or for bit 6 or 7 in byte 1, MFS overrides the specified value with zero, except for the 3290 in partitioned format mode.

For the 3290 in partitioned format mode, byte 1 bit 6 has special significance. If the DOF of the output message is the same as the DOF of the last message, then byte 1 bit 6 of the DSCA is checked for the erase/not erase partitions option before the output message is sent. Meanings of the bit 6 settings are shown in Table 44.

Table 44. 3290 Partitioned Format Mode Bit Setting

Byte	Bit	Setting	Meaning
1	6	B'1'	Erase all partitions before sending output message.
		B'0111'	Do not erase existing partitions.

The default is B'0' (do not erase). If bit 6 is defined, all existing partitions are erased and the output is sent according to the specified partition paging option (see “Partition Initialization Options and Paging” on page 244). If bit 6 is not defined, the output is sent according to the specified partition paging option and partitions that do not receive output remain in the state they were in before output was sent.

Table 45 shows the DSCA bit settings for TYPE=FIDS, FIDS3, FIDS4, FIDS7, FIJP, FIPB, or FIFP.

Table 45. Bit Settings for DSCA Field. For TYPE=FIDS, FIDS3, FIDS4, FIDS7, FIJP, FIPB, or FIFP

Byte	Bit	
0	0-7	Should be 0.
1	0	Should be 1.
	1-2	Not applicable for FIN output devices.
	3	Set 'device alarm' in output message header.
	4	Not applicable for FIN output devices.
	5-7	Should be 0.

For FIN devices, if a nonzero value is specified for byte 0, or for bits 1, 2, 5, 6, or 7 in byte 1, MFS overrides the specified value with zero.

Bits 1, 2, and 4 in byte 1 only function for 3270 and SLU 2 and are therefore not applicable to FIN. If set on, and the message is edited for an FIN output device, they are ignored.

For 3270 and FIN devices, the function specified is performed. For DPM devices, the specification is supplied to the remote program in a user-defined device field (DFLD).

FEAT=
Specifies features for this device or program group.

IGNORE
Specifies that device features are to be ignored for this device.

120|126|132
Specifies line length for 3284, and 3286 device types (TYPE=3270P).

CARD
Specifies that the device has a 3270 operator identification card reader. NOCD specifies the absence of the CARD feature.

DEKYBD
Specifies data entry keyboard feature. This feature implies PFK feature; therefore, PFK is invalid if DEKYBD is specified. NOPFK implies the absence of PFK and DEKYBD features.

PFK
Specifies that the device has program function keys. NOPFK specifies the absence of the PFK and DEKYBD features.

PEN
Specifies the selector light pen detect feature. NOPEN specifies the absence of the PEN feature.

DUAL
Specifies that the FIFP device has the dual independent forms feed feature.

132
Specifies that the FIFP device has the expanded print line feature.

1|2|3|4|5|6|7|8|9|10
Specifies customer-defined features for the SCS1, SCS2, 3270P, DPM-An, or DPM-Bn device type.

For SCS1, SCS2, and 3270P devices, FEAT= allows grouping of devices with special device characteristics. For example, FEAT=1 could group devices with a maximum of 80 print positions and no VFC, and FEAT=2 could group devices with 132 print positions and the VFC feature. FEAT=IGNORE should be specified to group together devices with a minimum set of device capabilities. For 3270P devices, when WIDTH= is specified, FEAT=(1...10) must also be specified. If FEAT=(1...10) is specified but WIDTH= is not specified, WIDTH= defaults to 120.

For DEV TYPE=DPM-An or DPM-Bn, FEAT= specifies a user-defined group of device formats so that programs with common features and dependencies can be selected together.

When IGNORE is specified, no other values should be coded in the FEAT= operand. When FEAT=IGNORE is not specified in the TERMINAL macro during system definition, the MSG statement must specify IGNORE in the SOR= operand for the device format with the IGNORE specification. Unless FEAT=IGNORE is used, FEAT= must specify **exactly** what was specified in the TERMINAL macro during IMS system definition. If it does not, the DFS057 error message is issued. When FEAT=IGNORE or 1-10 is specified for 3270 devices,

the operands PEN=, CARD=, and PFK= can still be specified. When TYPE=3270P and FEAT=IGNORE, MFS allows a line width of 120 characters.

CARD, PFK, DEKYBD, and PEN feature values are valid only for 3270 displays. DUAL is valid only if TYPE=FIFP. If the FEAT= operand is omitted, the default features are CARD, PFK, and PEN for 3270 displays; the default line width is 120 for TYPE=3270P and 80 for TYPE=FIFP.

1, 2, 3, 4, 5, 6, 7, 8, 9, and 10 are valid values only for 3270, 3270P, 3270-An, SCS1, SCS2, DPM-An, and DPM-Bn (for DEV TYPE=). For 3270 displays, the FEAT= specifications of 1 to 5 can be used to group devices with specific features or hardware data stream dependencies.

Restriction: This keyword is optional and cannot be used with any other feature specification for 3270 displays.

When using the same format for both the 3290 and the 3180, you must specify a different value on the FEAT= operand for each device type. The FEAT parameter values selected for each device must also be specified on the TERMINAL macro in the IMS SYSGEN.

For 274X, FIN, FIDS, FIDS3, FIDS4, FIDS7, FIJP, and FIPB, FEAT is always IGNORE. For FIFP, IGNORE is used unless 132 and DUAL are specified.

Feature operand values can be specified in any order, and only those values desired need be specified. The underlined values do not have to be specified because they are defaults. Only one value in each vertical list can be specified.

Examples: Some of the uses of the FEAT= specification are:

- `TYPE=DPM-A1,FEAT=1` could group device formats with DPAGE paging option and simulated attributes.
- `TYPE=DPM-A5,FEAT=2` could group device formats with no paging option and bit string attributes (which are not interpreted by MFS).
- `TYPE=DPM-B1,FEAT=IGNORE` could identify device formats with PPAGE paging option and a minimum set of program requirements.

PFK=

Defines an input field name to contain program function key literal or control function data (first subparameter) and, in positional or keyword format, either the literal data to be placed in the specified field, or the control function to be performed when the corresponding function key is entered (remaining subparameters).

The name of the first subparameter (the input field name that will contain the program function key literal or control function data) can be referred to by an MFLD statement and must *not* be used as the label of a DFLD statement within this DEV definition. The remaining subparameters can be specified in positional or keyword format. If the subparameters are in keyword format, the integer specified must be from 1 to 36, inclusive, and not duplicated. Only one PFK= operand format (positional or keyword) can be specified on a DEV statement. This operand is valid only for 3270 displays. At the time the actual format blocks are created, each literal is padded on the right with blanks to the length of the largest literal in the list. The maximum literal length is 256 bytes.

If the device supports the IMS copy function, then PFK12 invokes the copy function and the definition of PFK12 in the DEV statement is ignored; otherwise, the definition of PFK12 is honored.

If FEAT=NOPFK is specified, it is changed to PFK. The maximum number of user-defined PFKs is 36.

Control functions that can be specified are:

NEXTPP—PAGE ADVANCE
Specifies a request for the next physical page in the current output message. If no output message is in progress, no explicit response is made.

NEXTMSG—MESSAGE ADVANCE
Specifies a request to dequeue the output message in progress (if any) and to send the next output message in the queue (if any).

NEXTMSGP—MESSAGE ADVANCE PROTECT
Specifies a request to dequeue the output message in progress (if any), and send the next output message or return an information message indicating that no next message exists.

NEXTLP—NEXT LOGICAL PAGE
Specifies a request for the next logical page of the current message.

ENDMPPI—END MULTIPLE PAGE INPUT
Specifies the end of a multiple physical page input message.

PEN=
Defines an input field name to contain literal data when an immediate light pen detection of a field with a space or null designator character occurs. The literal data is defined on the DFLD statement with the PEN= operand. (See PEN= operand on the DFLD statement.) This name can be referred to by an MFLD statement and must not be used as the label of a DFLD statement within this DEV definition. The PEN= operand is valid only for 3270 displays. If FEAT=NOPEN is specified, it is changed to PEN.

If an immediate detect occurs on a field defined with a space or null designator character, and either another field has been selected or modified or has the MOD attribute, or the PEN= operand is not defined for the DFLD, a question mark (?) is inserted in the PEN= field name.

If no immediate detection occurs or the immediate detect occurs on a field defined with an ampersand (&) designator character, the PEN= operand is padded with the fill specified in the MFLD statement.

CARD=
Defines the input field name to receive operator identification card data when that data is entered. This name can be referenced by an MFLD statement and must not be used as the label of a DFLD statement within this DEV definition. This operand is valid only if a 3270 display or SCS1 is specified. If FEAT=NOCD is specified for a 3270 display, it is changed to CARD. All control characters are removed from magnetic card input before the data is presented to the input MFLD that refers to this card field name.

For 3270 displays, an unprotected field large enough to contain the magnetic card data and control characters must be defined through a DFLD statement. Position the cursor to this field and insert the card in the reader to enter card

information. The card data is logically associated with the CARD= field name, not the name used in the DFLD statement.

For device TYPE=SCS1, only card data with the operator ID (OID) character is associated with this field name. Cards with the OID character can be entered at any time during data entry. MFS treats data without the OID character as if it were data entered from the keyboard.

SYSMSG=
Specifies the label of the DFLD statements that define the device field in which IMS system messages are to be displayed. This operand is valid only if a 3270 display is specified. A DFLD with this label should be defined for each physical page within each DPAGE defined within this DEV definition. DFLDs for SYSMSG should be at least LTH=79 to prevent message truncation. The referenced DFLD can also be referenced by an MFLD statement.

FORMS=
Specifies a 1- to 16-byte literal. For the FIN, this literal is included in the output message header for each message sent to the device using this FMT. The data can be used by the FIN application program to ensure that special forms required for a given message are mounted on the device and that page size and forms alignment are established.

For SCS1 output to SLU 1 print data set components or SLU 4, this literal names the data set to receive IMS output. For 3770 programmable models defined to IMS as SLU 1 or SLU 4, however, the literal is ignored by the terminal and all print data set output goes to the SYS.INTR data set. For all SCS1 output to 3770 (nonprogrammable), SLU 1 non-PDS components or SLU 4, the literal is ignored.

For DEV TYPE DPM-An, this literal is included in the output message header. If the DPAGE or PPAGE paging option is specified, the literal is part of the special forms output message header sent as a separate transmission, followed (after a paging request from the remote program) by the DPAGE or PPAGE output message header and data records. If the default MSG option is selected, the output message header with literal is sent as the first record, followed by data records. See “Output Message Header” on page 225 for a discussion of output headers with the forms literal.

WIDTH=
Specifies the maximum line width for this DEV type as one of:

- Number of print positions per line of input or output data
- Number of punch positions per card of input or output data
- Card width for card reader input data

The defaults are 132 for SCS1 input and output, 80 for SCS2 input and output, and 120 for 3270P output. A specified number cannot exceed 255 for SCS1 and 249 for SCS2. Line width is specified relative to column 1, regardless of whether a left margin value is specified in the HTAB= keyword (SCS1 and SCS2 only). The width specified must be greater than or equal to 1.

For 3270P devices, if WIDTH is specified, then FEAT=(1...10) must also be specified. If FEAT=(1...10) is specified, and WIDTH= is not specified, WIDTH= defaults to 120.

HTAB=
Specifies when TYPE=SCS1:

- Where on the device MFS should set horizontal tab stops
- Whether and when MFS should insert tab control characters in the output message to cause horizontal tabbing
- Where on the device MFS should position the left margin

If HTAB= is not specified, no horizontal tabbing is done and the left margin position is assumed to be column 1.

SET|ONLINE|OFFLINE
Specifies that MFS should set horizontal formatting controls for the device. When MFS sets horizontal format controls for the device, the following characteristics are established: maximum line width, left and right margins, and horizontal tab stops. The default is SET when the HTAB= keyword is present.

SET
Specifies that MFS should set horizontal tab stops but should not insert tab control characters into the output message. You can then use horizontal tabbing on subsequent input.

ONLINE
Specifies that MFS should set horizontal tab stops at the specified (HT=) locations and insert tab control characters during online processing.

OFFLINE
Specifies that MFS should set horizontal tab stops at the specified (HT=) locations and insert tab control characters during offline compilation of the format.

1|lm (left margin)
Specifies the column position of the left margin. The default is 1. The value specified must be less than the WIDTH= value.

HT=
Specifies from 1 to 10 horizontal tab stop locations. The values specified must be relative to position 1, equal to or greater than the left margin value, and less than the WIDTH= value.

VT=
Specifies that MFS should insert tab control characters at the specified locations. From 1 to 11 vertical tab stop locations can be specified. If VTAB= is specified, the VT= values specified must be relative to line 1 and equal to or less than the bottom margin specified on the VTAB= keyword. If VTAB= is not specified, the VT= values must be equal to or less than the page depth specified in the PAGE= keyword. The maximum value is 255. If a value greater than 255 is specified, 255 is assumed and no error message is generated. VT= is valid only when TYPE=SCS1. If PAGE=(n,FLOAT) is specified, VT= is invalid.

X'00' is accepted as a valid tab stop only if VTAB= is also specified.

Together with VTAB= and PAGE=, VT= comprises a data stream to set the vertical format of the page. *tm* on the VTAB= keyword must be greater than or equal to 1 and less than *t1* on the VT= keyword. *bm* on the VTAB= keyword must be greater than or equal to *t11* on the VT= keyword and less than or equal to the maximum page length specified on the PAGE= keyword.

VTAB=
For SCS1 printers, specifies top (*tm*) and bottom (*bm*) page margins. Together

with VT= and PAGE=, VTAB= comprises a data stream to set the vertical format of the page. *tm* must be greater than or equal to 1 and less than *t1* on the VT= keyword. The maximum *tm* is 253.

bm must be greater than or equal to *t11* on the VT= keyword and less than or equal to the maximum page length specified on the PAGE= keyword. *bm* must be at least two greater than *tm*. If VTAB= is specified, then the PAGE= value must be 3 or greater.

A form feed (FF) is inserted after the set vertical format (SVF) data stream if the top margin (*tm*) specified on the VTAB= keyword is not equal to 1.

If PAGE=(n,FLOAT) is specified, VTAB= is invalid.

SLDI=
For SCS1 printers, specifies the line density for an output message in lines per inch. (See also SLDP=). SLDI= can also be specified on the DFLD statement. The SLDI= value must be from 1 through 72 and consistent with the architecture of the device for which it is specified (see the appropriate device or component manual).

If SLDI= is specified both on the DEV statement and the DFLD statement, two SLD data streams are created. One is sent at the beginning of a message to set the line density. The second is sent within the message, just prior to the field on which the SLDI= specification is encountered, but after any vertical tabs and new line characters.

Restriction: You cannot specify both SLDI= and SLDP= on the DEV statement.

The SLDI= specification within the message changes the line density from that set at the beginning of the message, and this latter line density remains in effect until explicitly reset.

SLDP=
For SCS1 printers, specifies the line density for an output message in points per inch. (See also SLDI=). SLDP= can also be specified on the DFLD statement. The SLDP= value must be from 1 through 72 and consistent with the architecture of the device for which it is specified (see the appropriate device or component manual).

If SLDP= is specified both on the DEV statement and the DFLD statement, two SLD data streams are created. One is sent at the beginning of a message to set the line density. The second is sent within the message, just prior to the field on which the SLDP= specification is encountered, but after any vertical tabs and new line characters.

Restriction: You cannot specify both SLDP= and SLDI= on the DEV statement.

The SLDP= specification within the message changes the line density from that set at the beginning of the message, and this latter line density remains in effect until explicitly reset.

Recommendation: Be careful when defining set line density (SLDx) keywords to ensure that forms alignment is maintained. If SLDx= is improperly defined, loss of forms alignment can occur.

VERSID=

Specifies any two-character or 2-byte hexadecimal value as the version ID. If MFS is specified or if the VERSID keyword is not specified, MFS calculates the version ID. MFS is the default.

The version ID is calculated by MFS and is based on the date and time stamp that an FMT definition has compiled. The value is printed on the MFS Language utility output so you can refer to it in format definitions.

SUB=

Specifies the character used by MFS to replace any X'3F' characters in the input data stream. No translation occurs if this parameter is specified as X'3F' or this parameter is not specified, or the input received bypasses MFS editing. The specified SUB character should not appear elsewhere in the data stream; therefore, it should be nongraphic.

X'*hh***'**

Character whose hexadecimal representation is '*hh*' replaces all X'3F' in the input data stream.

C'*c***'**

Character '*c*' replaces all X'3F' in the input data stream.

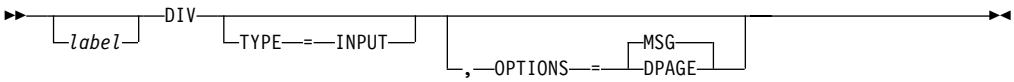
PDB=

(For the 3290 or 3180 in partitioned format mode) specifies the name of the Partition Descriptor Block that is used to describe the partition set for an output or input message. This parameter is valid only for DEV statements that specify TYPE=3270-An.

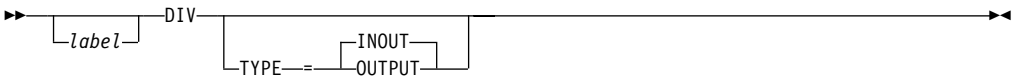
DIV Statement

The DIV statement defines device formats within a DIF or DOF. The formats are identified as input, output, or both input and output, and can consist of multiple physical pages. For DEV TYPE=274X, SCS1, SCS2, or DPM-AN, two DIV statements can be defined: DIV TYPE=OUTPUT and DIV TYPE=INPUT. For all other device types, only one DIV statement per DEV is allowed.

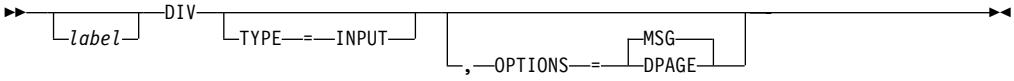
Format for DEV TYPE=274X, SCS1, or SCS2 and DIV TYPE=INPUT:



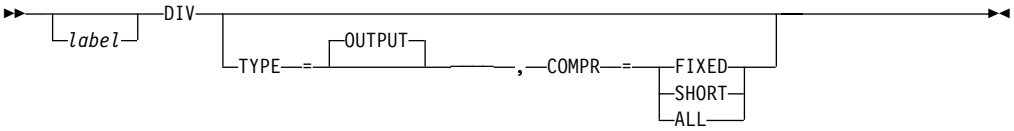
Format for DEV TYPE=3270 or 3270-An:



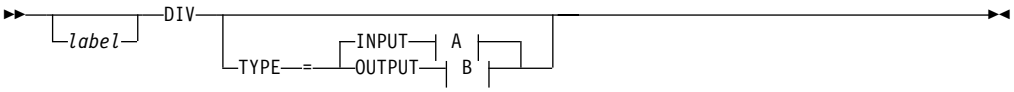
Format for DEV TYPE=FIN:



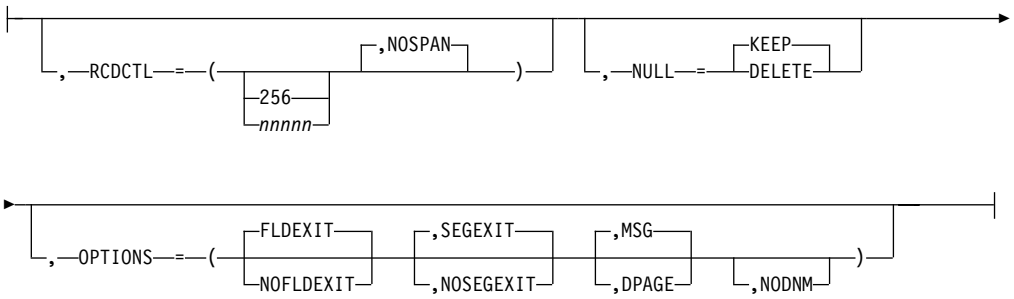
Format for DEV TYPE=274X, SCS1, SCS2, 3270P, FIDS, FIDS3, FIDS4, FIDS7, FIJP, FIPB, or FIFP and DIV TYPE=OUTPUT:



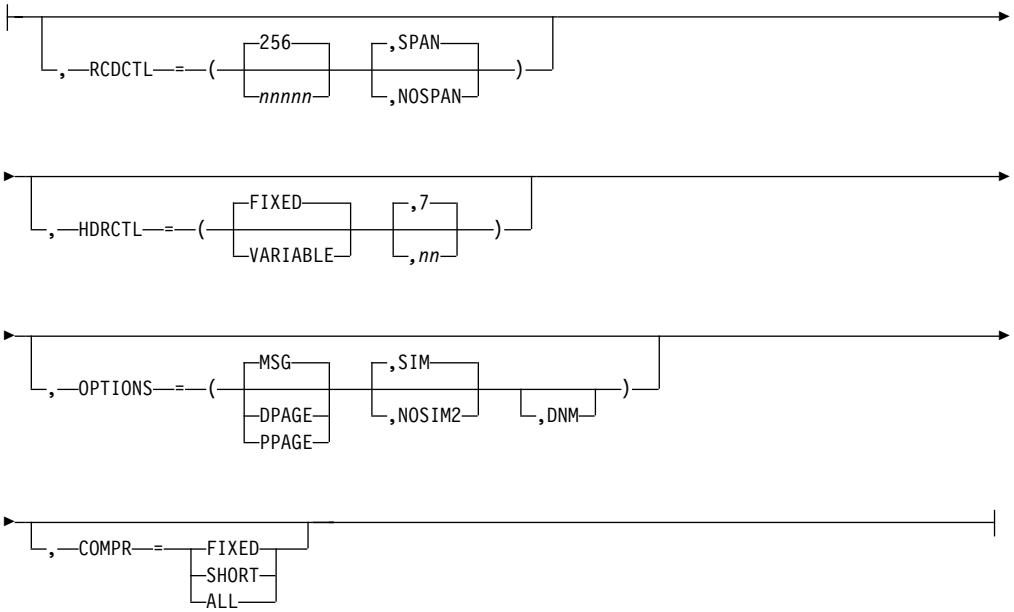
Format for DEV TYPE=DPM-An:



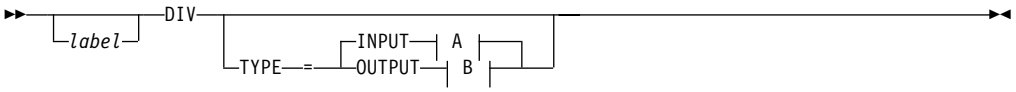
A:



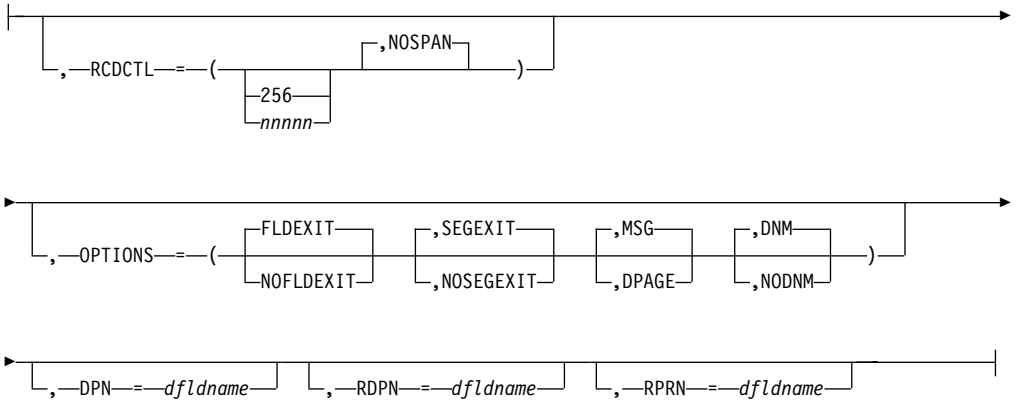
B:



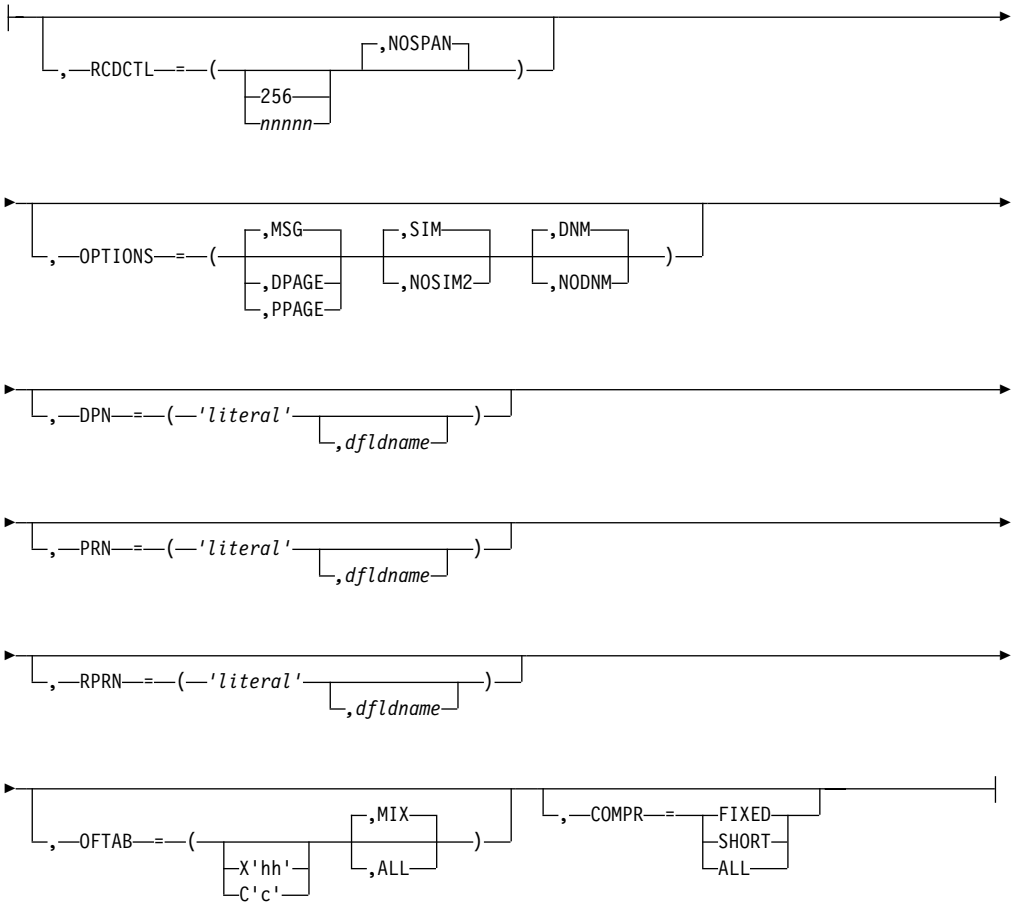
Format for DEV TYPE=DPM-Bn:


label
DIV
TYPE
INPUT
OUTPUT
A
B

A:


RCDCTL
256
nnnnn
NOSPAN
OPTIONS
FLDEXIT
NOFLDEXIT
SEGEXIT
NOSEGEXIT
MSG
DPAGE
DNM
NODNM
DPN
RDPN
RPRN
dfldname

B:


RCDCTL
256
nnnnn
NOSPAN
OPTIONS
MSG
DPAGE
PPAGE
SIM
NOSIM2
DNM
NODNM
DPN
'literal'
dfldname
PRN
RPRN
OFTAB
X'hh'
C'c'
MIX
ALL
COMPR
FIXED
SHORT
ALL

Parameters:

label
A one- to eight-character alphanumeric name can be specified to uniquely identify this statement.

TYPE=
Describes an input only format (INPUT), an output only format (OUTPUT), or both (INOUT).

If DIV TYPE=OUTPUT or TYPE=INPUT is specified, certain DEV statement keywords are applicable.

For example, specifying WIDTH=80 for DEV TYPE=SCS1 indicates that fields can be printed in columns 1 through 80 on output and received from columns 1 through 80 on input. Specifying WIDTH=80 for DEV TYPE=SCS2 indicates that both the card reader and card punch have the same number of punch positions. Specifying WIDTH=80 and HTAB=(SET,5) for DEV TYPE=SCS1 indicates that fields can be printed in columns 5 through 80 and received from columns 5 through 80 on input. In this case DFLD POS=(1,5) or POS=5 on input is the same as if you specified column 1 and a left margin position at 1. You enter data the same way, regardless of where the left margin is currently set.

RCDCTL=
This parameter is valid only if MODE=RECORD is specified on the DEV statement. For DEV TYPE=DPM-An or DPM-Bn only, RCDCTL specifies the maximum length of an input or output transmission record. For DPM-An, RCDCTL specifies whether (SPAN) or not (NOSPAN) fields can span records. The RCDCTL number cannot be larger than 32000 and should not be less than the length of the message output header (For DPM-An, see HDRCTL discussion.) The default value is 256. RCDCTL creates record definitions even if RCD statements are used in the same format definition.

- For DEV TYPE=DPM-An or TYPE=Bn and DIV TYPE=INPUT

  For an input format definition, fields must *not* span record boundaries, and therefore must be within the length specified by the RCDCTL value. NOSPAN is the default.

- For DEV TYPE=DPM-An or Bn and DIV TYPE=OUTPUT

  The RCDCTL size specified should be less than or equal to the output buffer size specified in the OUTBUF= macro at IMS system definition. If the RCDCTL size is greater than the OUTBUF value specified, one record may require multiple output transmissions and may produce undesirable results in the remote program. If fields do not exactly fit in the defined records, and NOSPAN has been specified, records may not be completely filled.

  The RCDCTL also specifies whether (SPAN) (for DPM-An only) or not (NOSPAN) a field may span record boundaries. If SPAN is specified (for DPM-An only), some fields may span a record boundary (but never a PPAGE boundary), and the remote program must include logic to associate the partial fields or deal with them separately.

  If NOSPAN is specified, every field is entirely contained within a record and no field will have a length greater than the RCDCTL value specified.

  The first data field is the first field of the message for OPTIONS=MSG. The first data field is the first field of the DPAGE or PPAGE for OPTIONS=DPAGE and PPAGE respectively. If the first data field does not fit in the same record as the output message header, and if OPTIONS=DPAGE

or PPAGE has been specified, the first data record will be sent in the next transmission. The output message header will be transmitted by itself (as is always the case for OPTIONS=MSG).

NULL=

For DEV TYPE=DPM-An and DIV TYPE=INPUT only, NULL= specifies whether MFS is to ignore (KEEP) or search for and replace (DELETE) trailing nulls in fields. If NULL=DELETE is specified, MFS searches input message fields for trailing nulls or for fields that are all nulls, and replaces the nulls with the fill character specified in the message definition. See “Optional Deletion of Null Characters for DPM-An” on page 196 for a discussion of the effects of NULL=DELETE.

OPTIONS=

For DIV TYPE=INPUT, the OPTIONS keyword specifies the exit routines to be called, the type of paging or delivery requested, and, for DPM-Bn only, the selection of the DPAGE data name to be used to map data.

For DIV TYPE=OUTPUT, the OPTIONS= keyword specifies the type of paging or delivery requested, the type of attribute processing requested, and, for DPM-Bn only, the selection of the DPAGE data name to be used to map data.

For DPM output messages, the option selection determines how records are constructed for transmission to the remote program or ISC subsystem and effects the distribution of processing and logic between the IMS application program and the remote program or ISC subsystem.

- For DEV TYPE=DPM-An or TYPE=DPM-Bn and DIV TYPE=INPUT

  FLDEXIT|NOFLDEXIT

  SEGEXIT|NOSEGEXIT
  Input data from this device type can be partially edited by the remote program before it is sent to IMS. For input format definitions, this parameter specifies whether (FLDEXIT and SEGEXIT) or not (NOFLDEXIT and NOSEGEXIT) exit routines specified in the MSG definition MFLD and SEG statements, respectively, are to be called for this DPM format. If NOFLDEXIT or NOSEGEXIT is specified, the corresponding exit routine is bypassed. FLDEXIT and SEGEXIT are the defaults.

  MSG
  Specifies that an input message can be created from a single DPAGE.

  DPAGE
  Specifies that an input message can be created from multiple DPAGEs. If multiple DPAGE input is not requested in MFS definitions, messages may *not* be created from more than one DPAGE. In this case:

  If a single DPAGE is transmitted and contains more data than defined for the DPAGE selected, the input message is rejected and an error message is issued.

  If multiple DPAGEs are transmitted, the input message is rejected and an error message is issued.

  NODNM (DPM-An only)

DNM/NODNM (DPM-Bn only)
When a data name (DNM) is specified or defaulted to (DPM-Bn only), a specific DPAGE is selected to map the current or only data transmission when:

The DPAGE data name is supplied as the DSN parameter in the message header, and

The DPAGE data name matches a defined DPAGE data name.

If these conditions are not met, the last defined DPAGE name is used to map the data, unless the DPAGE is defined as conditional.

When no data name (NODNM) is specified (for either DPM-An or -Bn) MFS selects a specific DPAGE by performing a conditional test on the data received and the COND= parameter.

- For DEV TYPE=274x, SCS1, SCS2, FIN, and DIV TYPE=INPUT

  MSG
  Specifies that an input message can be created from a single DPAGE.

  DPAGE
  Specifies that an input message can be created from multiple DPAGEs. If multiple DPAGE input is not requested in MFS definitions, messages may *not* be created from more than one DPAGE. In this case:
  - If a single DPAGE is transmitted and contains more data than defined for the DPAGE selected, the input message is rejected and an error message is sent to the other subsystem.
  - If multiple DPAGEs are transmitted, the input message is rejected and an error message is sent to the other subsystem.

- For DEV TYPE=DPM-An or TYPE=DPM-Bn and DIV TYPE=OUTPUT

  MSG
  Is the default and specifies that IMS will transmit all the DFLDs within a message together as a single message group. The message is preceded by an output message header. All DFLDs are transmitted. For DPM-Bn, the data structure name is optional in the header.

  DPAGE
  Specifies that IMS will transmit all DFLDs that are grouped in one logical page together. The logical page will be transmitted in one or more records. If PPAGE statements are defined with the DPAGE, each PPAGE statement begins a new record. An additional logical page will be sent when a paging request is received from the remote program. Each logical page is preceded by an output message header, and the label on the DPAGE is placed in the header. For DPM-Bn, the data structure name is optional in the DD header and depends on the specification of DNM/NODNM.

  PPAGE
  Specifies that IMS will transmit the DFLDs that are grouped in one presentation page (PPAGE) together in one chain. The presentation page will be transmitted in a group of one or more records. An additional presentation page will be sent when a paging request is sent to IMS from the remote program. Each presentation page is preceded by an output message header, and the label on the PPAGE statement is placed in the

header. For DPM-Bn, the data structure name is optional in the DD header and depends on the specification of DNM/NODNM.

SIM/NOSIM2
Specifies whether (SIM) or not (NOSIM2) MFS is to simulate attributes. SIM, the default, indicates that MFS is to simulate the attributes specified by the IMS application program and place the simulated attributes in corresponding DFLDs that are defined with ATTR=YES or YES,nn. The first byte of the field is used for the simulated attributes. If the MFLD does not supply 3270 attribute information (by means of the ATTR=YES or YES,nn operand) for the corresponding DFLD specifying ATTR=YES or YES,nn, a blank is sent in the first byte of the field. The application designer of the remote program or ISC subsystem is responsible for interpreting the simulated attribute within the remote program or ISC subsystem.

If NOSIM2 is specified, MFS sends a 2-byte bit string to the remote program or subsystem. This bit string is sent exactly as received from the IMS application program. 3270 extended bytes, if any (ATTR=YES,nn), are always sent as received from the application program and follow the 2-byte string of 3270 attributes. If the MFLD does not supply attribute information, binary zeros are sent in the two bytes preceding the data for the field.

See ATTR= on the DFLD statement for additional information.

DNM (DPM-An only)
May be used with the FORMS= keyword on the DEV statement to specify a literal in the message header. See the FORMS= keyword in this chapter and the discussion of output message headers with the forms literal in Chapter 4. This parameter is optional.

DNM/NODNM (DPM-Bn only)
If DNM is specified or defaulted to, MFS includes the following in the DD header:

The FMT name, if OPTIONS=MSG

The DPAGE name, if OPTIONS=DPAGE

The PPAGE name, if OPTIONS=PPAGE

If NODNM is specified, no data structure name (DSN) is supplied in the DD header.

HDRCTL=
Specifies, for DEV TYPE=DPM-An and DIV TYPE=OUTPUT only, the characteristics of the output message header.

FIXED
Specifies that a fully padded output message header is to be sent to the remote program. The structure of the fixed output message header is the same for all DPM output messages built using this FMT definition. The content of the output message header is shown in an example under “Output Format Control for SLU P DPM-An” on page 224. The base DPM output message header has a length of 7, and includes the version ID.

VARIABLE
Specifies that MIDNAME and DATANAME will have trailing blanks omitted

and their length fields adjusted accordingly. If MIDNAME is not used, neither the MIDNAME field nor its length is present.

nn Specifies the minimum length of the header, that is, the base header without MFS fields, as shown in the example under “Output Format Control for SLU P DPM-An” on page 224. The default is 7, which is the length of the base message header for DPM. Specifying other than 7 may cause erroneous results in the remote program.

The parameters referenced below as RDPN=, DPN=, PRN=, and RPRN= refer to both the ISC ATTACH function management header and the equivalent ISC SCHEDULER function management header.

RDPN=
For DIV TYPE=INPUT, the dfldname specification permits the suggested return destination process name (RDPN) to be supplied in the input message MFLD referencing this dfldname. If dfldname is not specified, no RDPN is supplied in the input message.

DPN=
For DIV TYPE=OUTPUT, the 'literal' specification requests MFS to use this literal as the DPN in the output ATTACH message header. The literal cannot exceed 8 characters. If the dfldname is also specified, the data supplied in the MFLD referencing this dfldname is used as the DPN in the output ATTACH message header. If no output message MFLD reference to the dfldname exists, the 'literal' is used. If the data in the MFLD referencing the dfldname is greater than 8 characters, the first 8 characters are used.

PRN=
For DIV TYPE=INPUT, the dfldname specification permits the suggested primary resource name (PRN) to be supplied in the input message MFLD referencing this dfldname. If the dfldname is not specified, no PRN is supplied in the input message to the application program.

For DIV TYPE=OUTPUT, the 'literal' specification requests MFS to use this literal as the PRN in the output ATTACH message header. The literal cannot exceed 8 characters. If the dfldname is also specified, the data supplied in the MFLD referencing this dfldname is used as the PRN in the output ATTACH message header. If no output message MFLD reference to the dfldname exists, the 'literal' is used. If the data in the MFLD referencing the dfldname is greater than 8 characters, the first 8 characters are used.

RPRN=
For DIV TYPE=INPUT, the dfldname specification permits the suggested return primary resource name (RPRN) to be supplied in the input message MFLD referencing this dfldname. If dfldname is not specified, no RPRN is supplied in the input message to the application program.

For DIV TYPE=OUTPUT, the 'literal' specification requests MFS to use this literal as the suggested return primary resource name (RPRN) in the output ATTACH message header. The literal cannot exceed 8 characters. If the dfldname is also specified, the data supplied in the MFLD referencing this dfldname is used as the RPRN in the output ATTACH message header. If no output message MFLD reference to the dfldname exists, the 'literal' is used. If the data in the MFLD referencing the dfldname is greater than 8 characters, the first 8 characters are used.

OFTAB=
Directs MFS to insert output field tab separator characters in the output data

stream for the message. If OPTIONS=DNM and OFTAB, then the OFTAB character is placed in the DD header and an indicator is set to MIX or ALL. If OPTIONS=NODNM, then no DD header is sent.

X'*hh*'
Character whose hexadecimal representation is "hh" is used as the output field tab separator character. Specification of X'3F' or X'40' is invalid.

C"c"
Character "c" is used as the output field tab separator character. Specification of C""b is invalid.

Restriction: The character specified cannot be present in the data stream from the IMS application program. If it is present, it is changed to a blank (X'40').

If an output field tab separator character is defined, either MIX or ALL may also be specified. Default value is MIX.

MIX
Specifies that the output field tab separator character is to be inserted into each individual field with no data or with less data than the defined DFLD length.

ALL
Specifies that the output field tab separator character is to be inserted into all fields, regardless of data length.

COMPR=
Requests MFS to remove trailing blanks from short fields, fixed-length fields, or all fields presented by the application program.

For DPM-AN devices, trailing blanks are removed at the end of a segment if all of the following conditions are true:

1. FILL=NULL or FILL=PT is specified.
2. GRAPHIC=YES is specified for the current segment being mapped.
3. OPT=1 or OPT=2 is specified in the MSG segment.

If conditions 1, 2, and 3 above are met, replacement of trailing blanks occurs as follows:

FIXED
Specifies that trailing blanks from fixed-length fields are to be replaced by nulls.

SHORT
Specifies that trailing blanks fields shortened by the application program are to be replaced by nulls.

ALL
Specifies that trailing blanks from all fields are to replaced by nulls.

The trailing nulls are then compressed at the end of the record. See the description of the FILL= operand for additional information.

For DPM-BN devices, trailing blanks are removed if all of the following conditions are true:

1. OFTAB is specified on the current DIV statement, or FILL=NULL or FILL=PT is specified.

2. GRAPHIC=YES is specified for the current segment being mapped.
3. OPT=1 or OPT=2 is specified in the MSG segment.

If conditions 1, 2, and 3 above are met, the removal of trailing blanks occurs as follows:

FIXED
Specifies that trailing blanks are to be removed from fixed-length fields.

SHORT
Specifies that trailing blanks are to be removed from fields shortened by the application program.

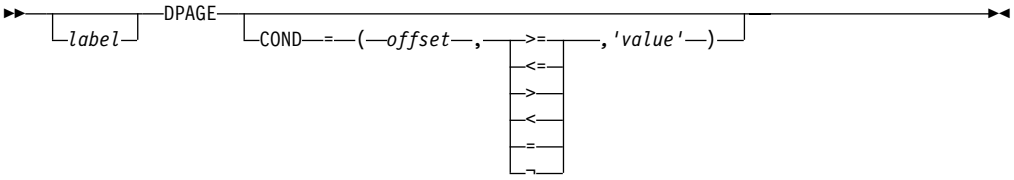
ALL
Specifies that trailing blanks are to be removed from all fields.

Related Reading: For additional information about blank compression for DPN-BN devices, see “Trailing Blank Compression” on page 232.

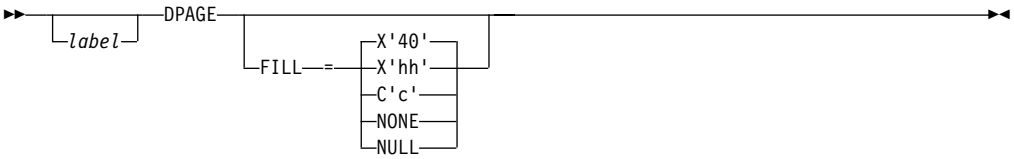
DPAGE Statement

The DPAGE statement defines a logical page of a device format. This statement can be omitted if none of the message descriptors referring to this device format (FMT) contains LPAGE statements and if no specific device option is required.

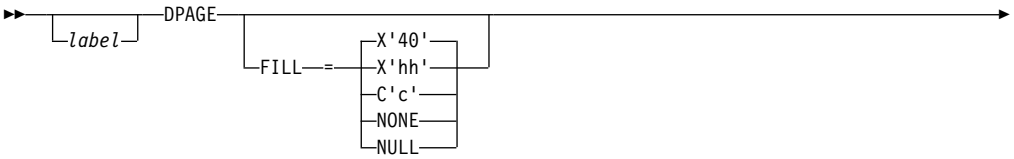
Format for DEV TYPE=274X, DPM-An, or DPM-Bn AND DIV TYPE=INPUT:

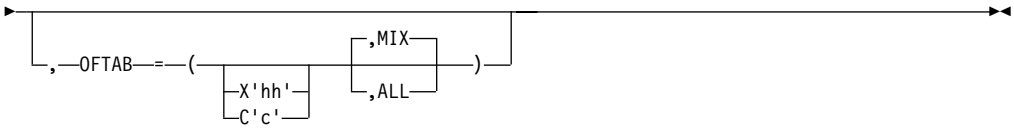


Format for DEV TYPE=274X or DPM-An AND DIV TYPE=OUTPUT:

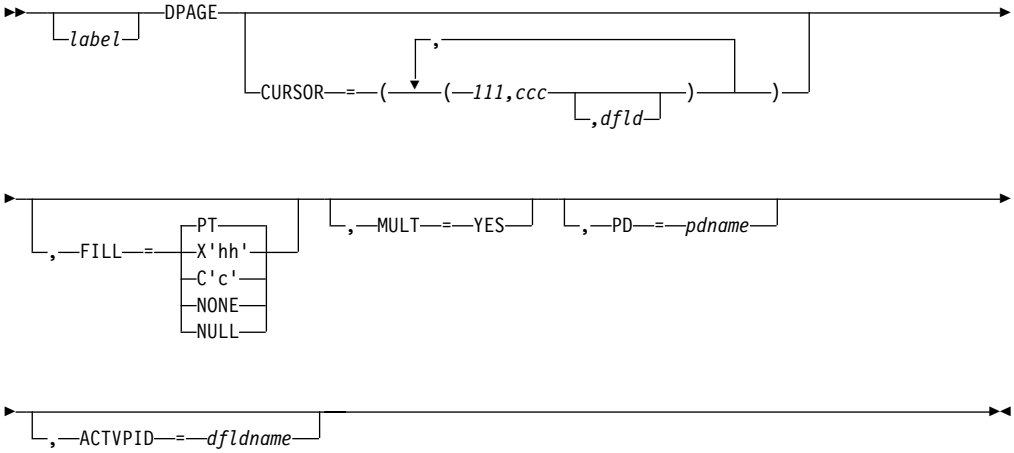


Format for DEV TYPE=DPM-Bn AND DIV TYPE=OUTPUT:

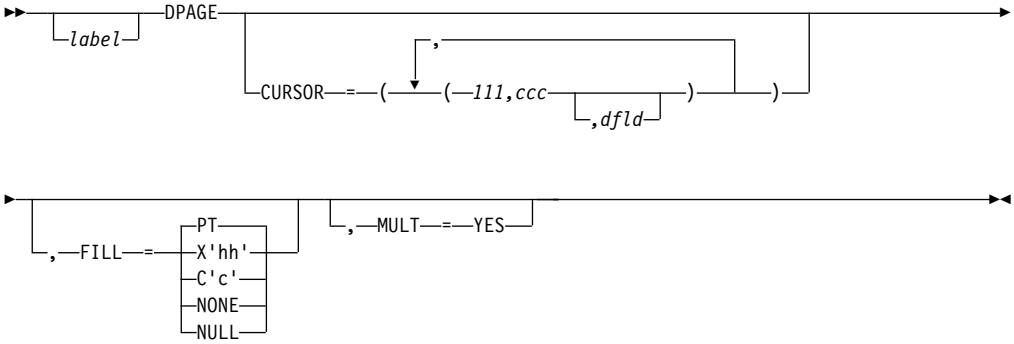




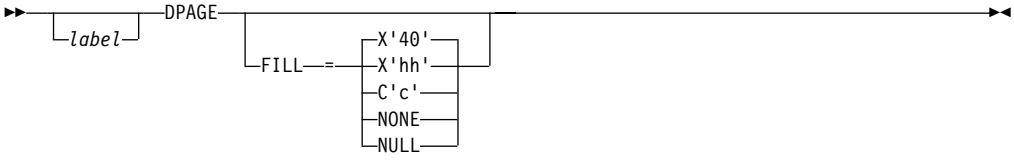
Format for DEV TYPE=3270-An:



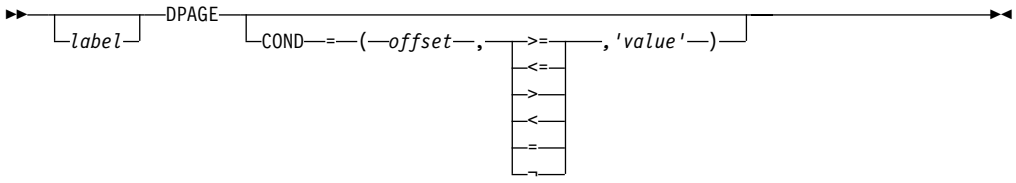
Format for DEV TYPE=3270:



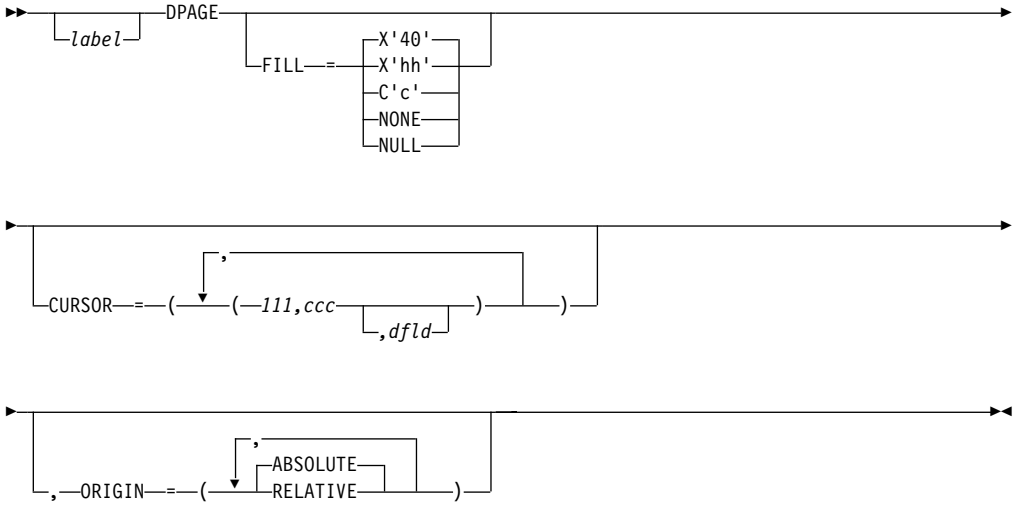
Format for DEV TYPE=3270P:



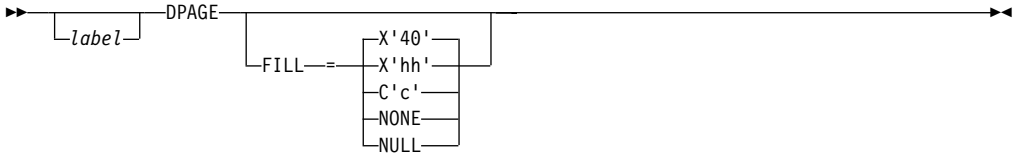
Format for DEV TYPE=FIN:



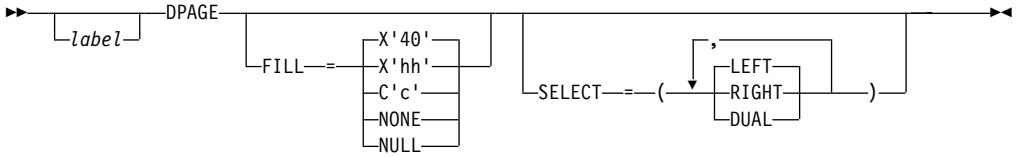
Format for DEV TYPE=FIDS, FIDS3, FIDS4, or FIDS7:



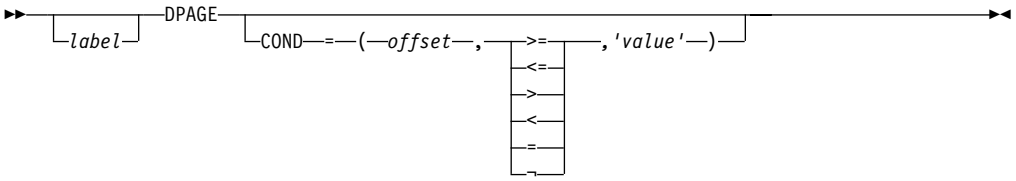
Format for DEV TYPE=FIJP or FIPB:



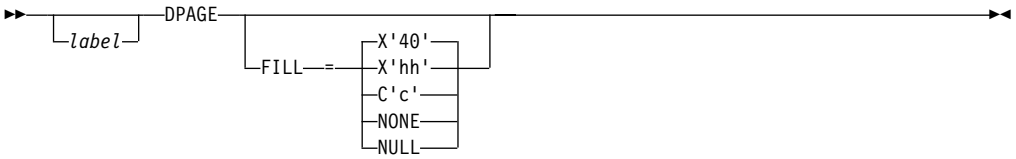
Format for DEV TYPE=FIFP:



Format for DEV TYPE=SCS1 or SCS2 AND DIV TYPE=INPUT:



Format for DEV TYPE=SCS1 or SCS2 AND DIV TYPE=OUTPUT:



Parameters:

label

A 1- to 8-byte alphanumeric name may be specified for this device format that contains LPAGE SOR= references, or if only one DPAGE statement is defined for the device. If multiple DEV statements are defined in the same FMT definition, each must contain DPAGE statements with the same label.

For device type DPM-An and DIV statement OPTIONS=DPAGE, this name is sent to the remote program as the data name in the output message header. If the label is omitted, MFS generates a diagnostic name and sends it to the remote program in the header. If the DPAGE statement is omitted, the label on the FMT statement is sent in the output message header. If OPTIONS=DNM, the label on the FMT statement is sent as the DSN in the DD header.

COND=

Specifies a conditional test to be performed on the first input record. The offset specified is relative to zero. The specification of the offset must allow for the LLZZ field of the input record (for example, the first data byte is at offset 4). If the condition is satisfied, the DFLDs defined following this DPAGE will be used to format the input. When no conditions are satisfied, the last defined DPAGE will be used only if the last defined DPAGE does not specify COND=. If the COND= parameter is specified for the last DPAGE defined and the last defined DPAGE condition is not satisfied, the input message will be rejected. Multiple LPAGE definitions are allowed in message input definitions.

If this keyword is specified, and OPTIONS=NODNM is specified on the DIV statement, this specification is used for DPAGE selection. If this keyword is specified and OPTIONS=DNM is specified on the DIV statement, the COND= specification is ignored and the data structure name from the DD header is used for DPAGE selection.

Lowercase data entered from 274X, Finance, SCS1, or SCS2 keyboards is not translated to uppercase when the COND= comparison is made. Therefore, the literal operand must also be in lowercase.

FILL=

Specifies a fill character for output device fields. Default value for all device types except the 3270 display is X'40'; default for the 3270 display is PT. For

3270 output when EGCS fields are present, only FILL=PT or FILL=NULL should be specified. A FILL=PT erases an output field (either a 1- or 2-byte field) only when data is sent to the field, and thus does not erase the DFLD if the application program message omits the MFLD. For DPM-Bn, if OFTAB is specified, FILL= is ignored and FILL=NULL is assumed.

NONE
Must be specified if the fill character from the message output descriptor is to be used to fill the device fields.

X'*hh***'**
Character whose hexadecimal representation is 'hh' will be used to fill the device fields.

C'*c***'**
Character 'c' will be used to fill the device fields.

NULL
Specifies that fields are not to be filled. For devices other than the 3270 display, 'compacted lines' are produced when message data does not fill the device fields.

For DPM-An devices, trailing nulls (X'3F') are removed from all records transmitted to the remote program or subsystem. Trailing nulls are removed up to the first non-null character. Null characters between non-null characters are transmitted. If the entire record is null, but more data records follow, a record containing a single null is transmitted to the remote program. If the entire record is null and more records follow, if OPTIONS=MSG or DPAGE, or in a PPAGE, if OPTIONS=PPAGE, then all null records are deleted to the end of that DPAGE or PPAGE.

PT
Is identical to NULL except for the 3270 display. For the 3270 display, specifies that output fields that do not fill the device field (DFLD) are followed by a program tab character to erase data previously in the field; otherwise, this operation is identical to FILL=NULL.

For 3270 display devices, any specification with a value less than X'3F' is changed to X'00' for control characters or to X'40' for other nongraphic characters. For all other devices, any FILL=X'hh' or FILL=C'c' specification with a value less than X'3F' is ignored and defaulted to X'3F' (which is equivalent to a specification of FILL=NULL).

MULT=YES
Specifies that multiple physical page input messages will be allowed for this DPAGE.

CURSOR=
Specifies the position of the cursor on a physical page. Multiple cursor positions may be required if a logical page or message consists of multiple physical pages. The value lll specifies line number, ccc specifies column; both lll and ccc must be greater than or equal to 1. The cursor position must either be on a defined field or defaulted. The default lll,ccc value for 3270 displays is 1,2. For Finance display components, if no cursor position is specified, MFS will not position the cursor—the cursor is normally placed at the end of the output data on the device. For Finance display components, all cursor positioning is absolute, regardless of the ORIGIN= parameter specified.

The **dfld** parameter provides a method for supplying the application program with cursor information on input and allowing the application program to specify cursor position on output.

Recommendation: Use the cursor attribute facility (specify ATTR=YES in the MFLD statement) for output cursor positioning.

The **dfld** parameter specifies the name of a field containing the cursor position. This name may be referenced by an MFLD statement and must *not* be used as the label of a DFLD statement in this DEV definition. The format of this field is two binary halfwords containing line and column number, respectively. When this field is referred to by a message input descriptor, it will contain the cursor position at message entry. If referred to by a message output descriptor, the application program places the desired cursor position into this field as two binary halfwords containing line and column, respectively. Binary zeros in the named field cause the specified lll,ccc to be used for cursor positioning during output. During input, binary zeros in this field indicate that the cursor position is not defined. The input MFLD referring to this dfld should be defined within a segment with GRAPHIC=NO specified or should use EXIT=(0,2) to convert the binary numbers to decimal.

ORIGIN=

Specifies page positioning on the Finance display for each physical page defined. Default value is ABSOLUTE.

ABSOLUTE

Erases the previous screen and positions the page at line 1 column 1. The line and column specified in the DFLD statement will become the actual line and column of the data on the screen.

RELATIVE

Positions the page starting on column 1 of the line following the line where the cursor is positioned at time of output. Results may be undesirable unless all output to the device is planned in a consistent manner.

OFTAB=

Directs MFS to insert the output field tab separator character specified on this DPAGE statement for the output data stream of the DPAGE being described.

X'*hh***'**

Character whose hexadecimal representation is 'hh' is used as the output field tab separator character. Specification of X'3F' or X'40' is invalid.

C'*c***'**

Character 'c' is used as the output field tab separator character. Specification of C' ' is invalid.

Restriction: The character specified cannot be present in data streams from the IMS application program. If it is present, it is changed to a blank (X'40').

If the output field tab separator character is defined, either MIX or ALL may also be specified. Default value is MIX.

MIX

Specifies that an output field tab separator character is to be inserted into each individual field with no data or with data less than the defined DFLD length.

ALL
Specifies that an output field tab separator character is to be inserted into all fields, regardless of data length.

SELECT=
Specifies carriage selection for a FIFP device with FEAT=DUAL specified in the previous DEV statement. It is your responsibility to ensure that proper forms are mounted and that left margins are set properly. Default value is LEFT.

LEFT
Causes the corresponding physical page defined in this DPAGE to be directed to the left platen.

RIGHT
Causes the corresponding physical page defined in this DPAGE to be directed to the right platen.

DUAL
Causes the corresponding physical page defined in this DPAGE to be directed to both the left and right platens.

PD=
(for the 3180 and 3290 in partition formatted mode) Specifies the name of the partition descriptor of the partition associated with the DPAGE statement. This is the parameter that maps a logical page of a message to or from the appropriate partition. The name of the PD must be contained within the PDB statement specified in the DEV statement.

ACTVPID=
(for the 3290 in partition formatted mode) Specifies the name of an output field in the message containing the partition identification number (PID) of the partition to be activated. This dfldname must be referenced by an MFLD statement and must not be used as the label of a DFLD statement in the DEV definition. The application program places the PID of the partition to be activated in this field. The PID must be in the format of a two byte binary number ranging from X'0000' to X'000F'.

Do not specify this operand for the 3180. Because only one partition is allowed for this device, you need not specify an active partition.

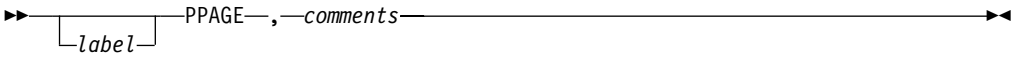
PPAGE Statement

The PPAGE statement, valid only for device types of DPM-An or DPM-Bn, defines the beginning of a presentation page. A presentation page is the unit of data delivered to the remote program in response to a paging request when OPTIONS=PPAGE has been specified in the DIV statement for this definition. For DPM-Bn MODE=RECORD only, if OPTIONS=MSG or DPAGE has been specified, paging is as described for those options under the DIV statement, and the PPAGE statement then defines the beginning of a new record (that is, it is equivalent to a RCD statement).

For an input DPAGE, only one PPAGE statement is allowed, and it must be placed between the DPAGE statement and the first DFLD statement. For an output DPAGE, if two consecutive PPAGE statements appear in the DPAGE for a message defined with OPTIONS=PPAGE, only an output message header with the PPAGE label as its data name is sent to the remote program, except OPTIONS=(PPAGE,DNM) for DPM-Bn. For DPM-Bn, a PPAGE statement without a DFLD statement is not allowed when OPTIONS=(PPAGE, NODNM) is specified for

DIV TYPE=OUTPUT. A warning message is issued, and the PPAGE statement is ignored. For OPTIONS=MSG or DPAGE, consecutive PPAGE statements are ignored.

Format:



Parameters:

label
A one- to eight-character alphanumeric name should be specified. For OPTIONS=PPAGE, this label is sent as the data name for DPM-An or as the data structure name for DPM-Bn in the message output header or DD header to identify the data structure of this presentation page to the remote program. If no label is specified, MFS generates a diagnostic label that is sent to the remote program in the header.

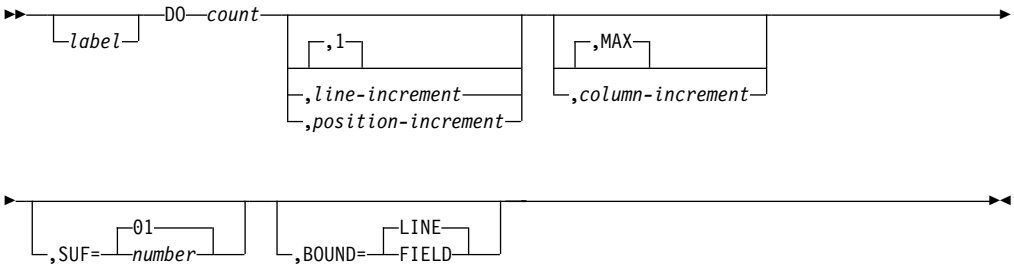
Recommendation: Specify a user-defined label because the MFS-generated name can change whenever the MFS definitions are recompiled.

The label specified should be unique, at least within a given FMT definition, and preferably within an IMS system if the remote program uses this label to identify the appropriate DSECT for formatting the data included in this presentation page.

DO Statement

The DO statement causes repetitive generation of DFLD and RCD statements between the DO and ENDDO statements. When DO is used, there are restrictions in the naming of DFLDs (refer to “DFLD Statement” on page 370).

Format:



Parameters:

label
A one- to eight-character alphanumeric name can be specified. It is not used.

count
Specifies how many times to generate the statements.

line-increment
Specifies how much to increase the line position after the first cycle. The first

cycle uses the *lll* value specified in the POS= keyword of the DFLD statement. The default is 1. This parameter is not specified for DEV type DPM-An or DPM-Bn.

position-increment
Specifies how much to increase the position parameter after the first cycle. The first cycle uses the *nnn* value specified in the POS= operand of the DFLD statement. The position increment is used for an input device format when MODE=STREAM is specified. This parameter is not specified for DEV type DPM-An or DPM-Bn.

MAX
Specifies that the line increment to be used at the end of each cycle and the column values in the DFLDs are to remain the same for each cycle. This parameter is not used if MODE=STREAM is specified for the device format or if DEV type is DPM-An or DPM-Bn; if present, it is ignored.

column-increment
Specifies how much to increase the column position after the first cycle. The first cycle uses the *ccc* value specified in the POS= keyword of the DFLD statement. The default is MAX. This parameter is not used for DEV type DPM-An or DPM-Bn, or when MODE=STREAM is specified for the device format, because it is ignored.

SUF=
Specifies the 2-digit suffix to be appended to the *dfldname* of the first group of generated DFLD statements. The default is 01. MFS increments the suffix by one on each subsequent generation of statements.

If the specified suffix exceeds 2 digits, MFS uses the rightmost 2 digits.

If the specified count is such that the generated suffix eventually exceeds 2 digits, MFS reduces the count to the largest legitimate maximum value. For example, if count equals 8 and SUF=95, invalid suffixes of 100, 101, and 102 would result. In this instance, MFS reduces the count to 5, processes the statement, and issues an error message.

BOUND=
Specifies when updates to line position and column position are to occur. The default is LINE. This parameter is not used if MODE=STREAM is specified for the device format or if DEV type is DPM-An or DPM-Bn; if present, it is ignored.

LINE
Specifies that all fields be inspected **before** the repetition is performed. If the column increment would cause any field in the group of DFLD statements to not fit on a line, the column position value for all fields is reset to the initial value, and the line position values are increased by the line-increment value.

FIELD
Specifies that each time the statement is repeated, the column position value is increased by the column-increment value. If MAX is specified, or the new column position value reaches device line length capacity, the line position value is increased by the line-increment value and the column position value is reset to its initial value.

Printing Generated DFLD Statements: The generated DFLD statements can be printed in a symbolic source format by specifying COMP in the parameter list of the EXEC statement. This provides a means of seeing the results of the DFLD statement generation without having to interpret the intermediate text blocks.

The following items are printed for each generated DFLD statement:

- The generated statement sequence number followed by a plus sign (+) to indicate that the DFLD statement was generated as a result of DO statement processing.
- The DFLD statement label, if present, including the appended suffix.
- The statement operator, DFLD.
- For EGCS literals, the G, SO, and SI are not present. Literals are truncated if there is insufficient room to print all specifications. Truncation is indicated by a portion of the literal with three periods (...), representing the truncated portion.
- ATTR=(YES,*nn*), if present.
- ATTR=YES, if present.
- ATTR=*nn*, if present.
- ATTR=(...), if attributes are present.
- EATTR=(...), if present.
- The RECORD or STREAM form of the POS= keyword, with the line and column or stream position updated by the respective increments. This is not printed if DEV type is DPM-An or DPM-Bn.
- SCA, if present.
- The field length, in the form of LTH=*nnnn*.

No other operands are printed, even if specified on the source DFLD statement.

For device type DPM-An or DPM-Bn, the RCD statement can appear between a DO and ENDDO statement. If it does, a new record boundary is created for each repetitive generation of the DFLD field following the RCD statement. For example, the following sequence causes the DFLDs A01, B01, and C01 to be in record 1, while A02, B02, and C02 are in record 2, and A03, B03, and C03 are in record 3.

```
    DO 3
    RCD
A   DFLD    LTH=10
B   DFLD    LTH=10
C   DFLD    LTH=10
    ENDDO
```

Alternatively, the RCD statement can immediately precede the DO statement. If it does, a new record boundary begins with the first DFLD after the DO statement and does not end until the ENDDO statement (or the maximum record length) is reached. For example, the following sequence causes the DFLD D01 to begin a new record, in which E01, D02, and E02 also occur.

```
    RCD
    DO 2
D   DFLD    LTH=10
E   DFLD    LTH=10
    ENDDO
```

RCD Statement

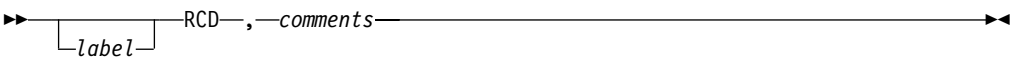
The RCD statement, valid for DEV TYPE=DPM-An or DPM-Bn only, can be used to influence the placement of DFLDs in records. The RCD statement precedes a DFLD statement and initiates a new transmission record for delivery to a remote program. DFLDs following the RCD statement are included into the transmission record until the next RCD statement or the maximum record length is reached (or, if NOSPAN is specified, until a field will not be fully contained in the current record).

Format Definition Statements: RCD

The RCD statement can be placed after the PPAGE, DO, DFLD, or ENDDO statements. (The effects of placing RCD before and after a DO statement are discussed in “DO Statement” on page 367.) If a RCD statement is immediately followed by another, only the first one is effective.

The RCD statement is invalid for STREAM mode.

Format:



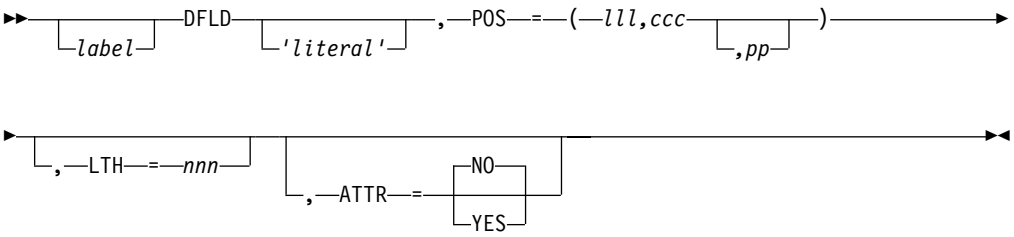
Parameters:

label
A one- to eight-character alphanumeric name can be specified. It is not used.

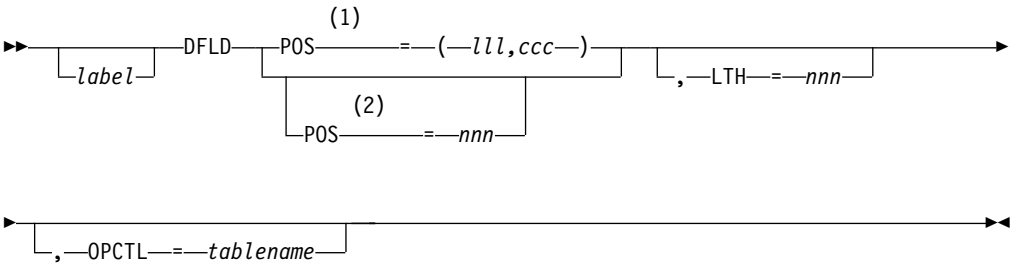
DFLD Statement

The DFLD statement defines a field within a device format which is read from or written to a terminal or remote program. Only those areas which are of interest to the IMS or remote application program should be defined. Null space in the format does not need to be defined.

Format for DEV TYPE=274X AND DIV TYPE=OUTPUT:



Format for DEV TYPE=274X AND DIV TYPE=INPUT:

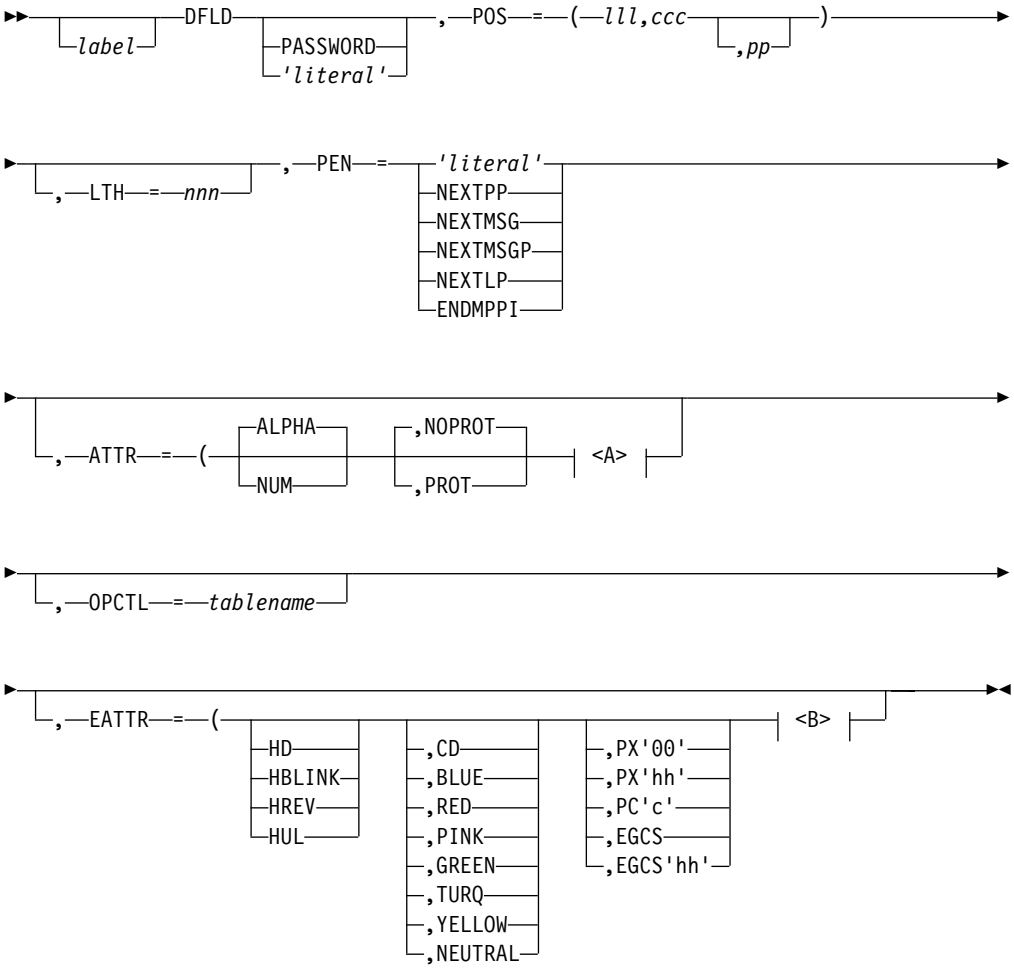


Notes:

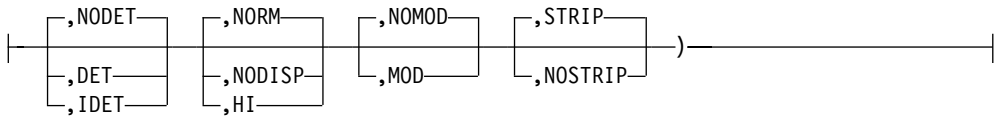
1 Used for MODE=RECORD only.

2 Used for MODE=STREAM only.

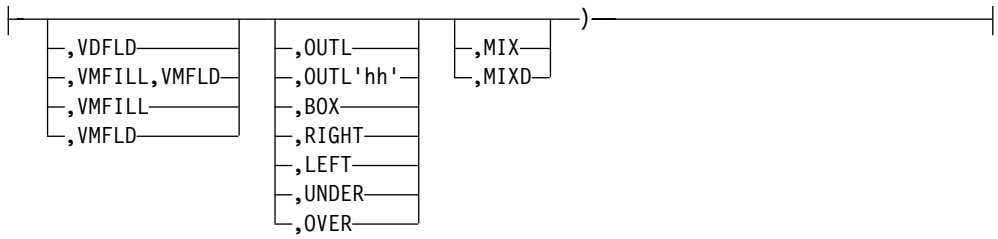
Format for DEV TYPE=3270 or 3270-An:



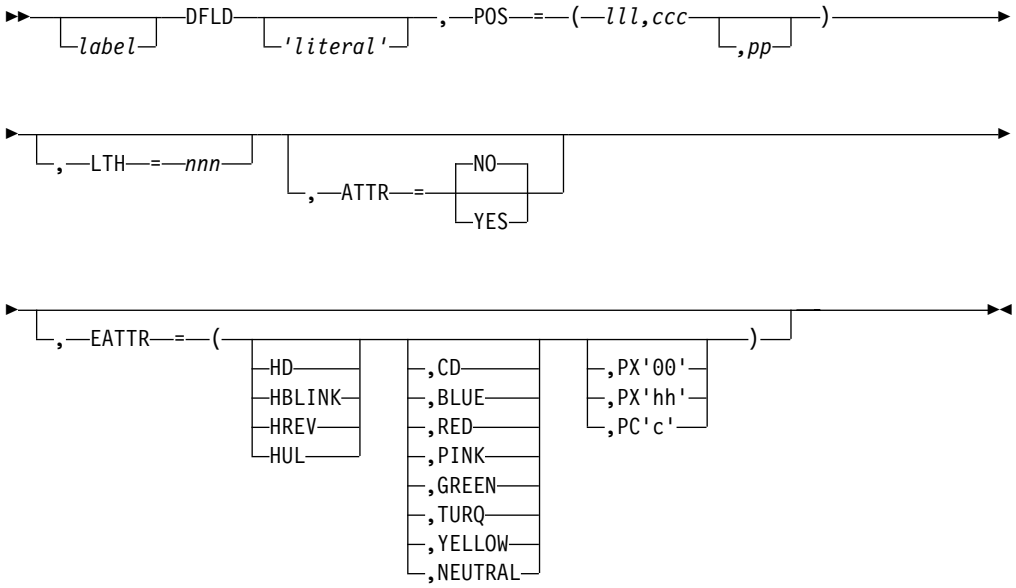
<A>:



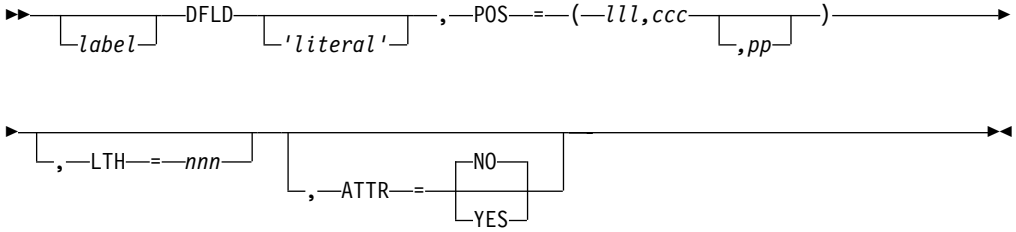
<B>:



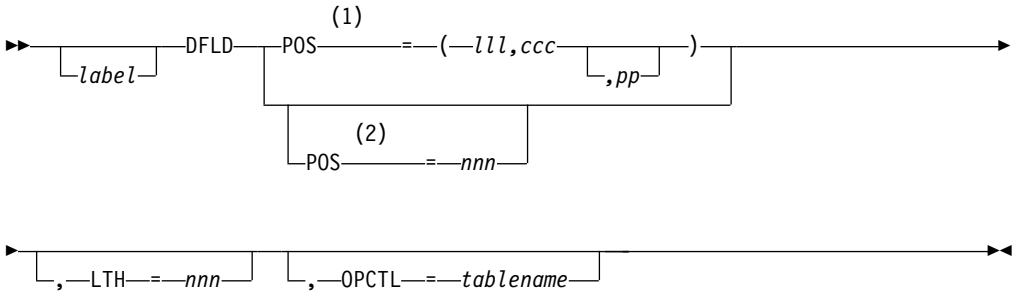
Format for DEV TYPE=3270P:



Format for DEV TYPE=FIDS, FIDS3, FIDS4, FIDS7, FIFP, FIJP, and FIPB:



Format for DEV TYPE=FIN:

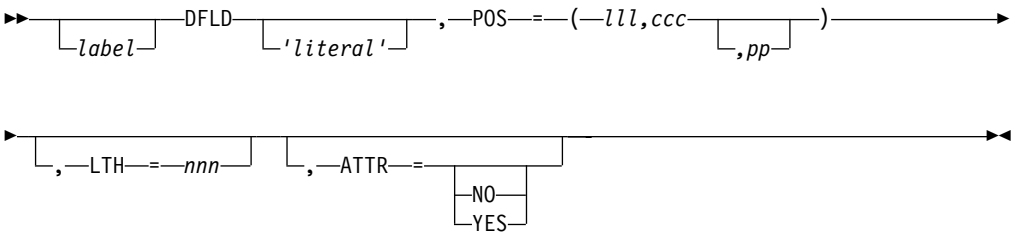


Notes:

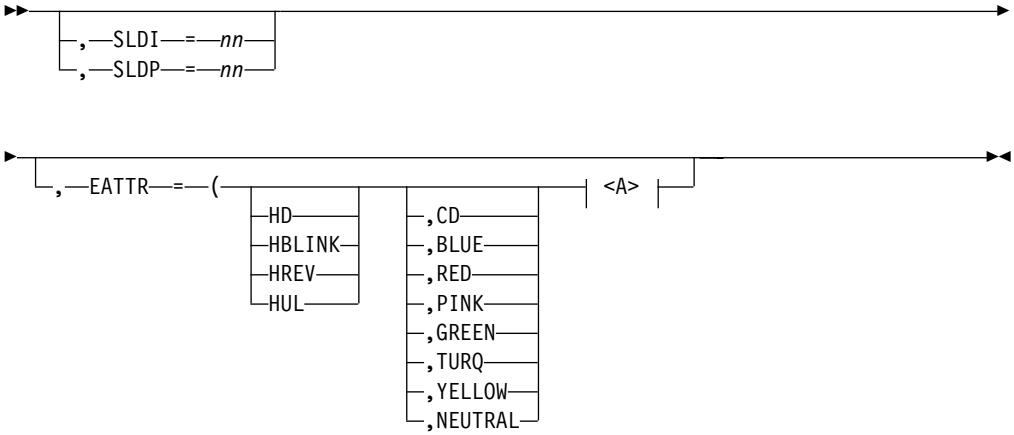
1 MODE=RECORD only

2 MODE=STREAM only

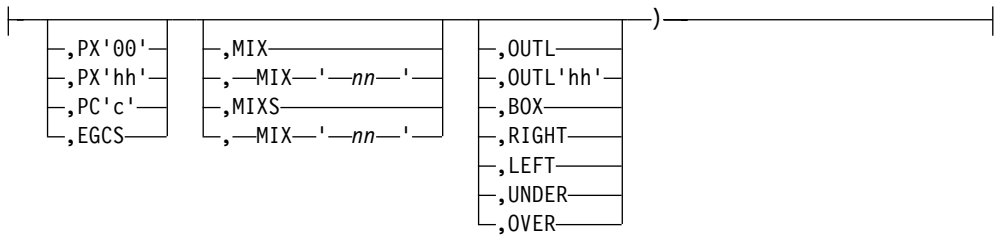
Format for DEV TYPE=SCS1 OR SCS2 AND DIV TYPE=OUTPUT:



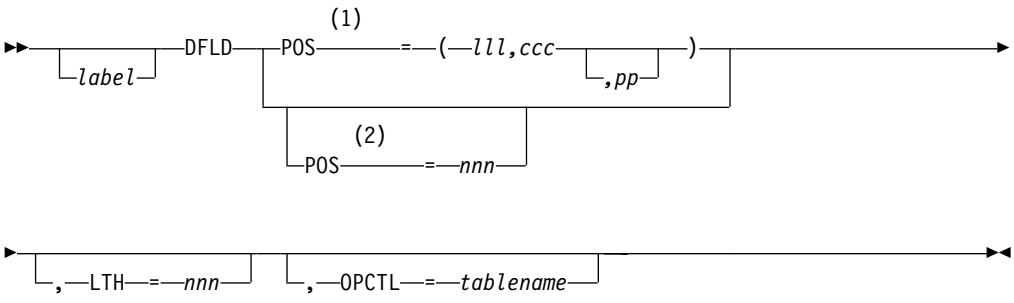
Format for SCS1 ONLY:



<A>:



Format for DEV TYPE=SCS1 or SCS2 AND DIV TYPE=INPUT:

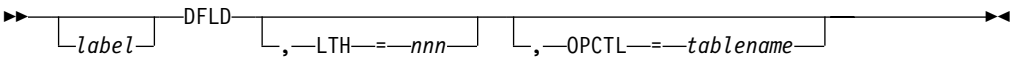


Notes:

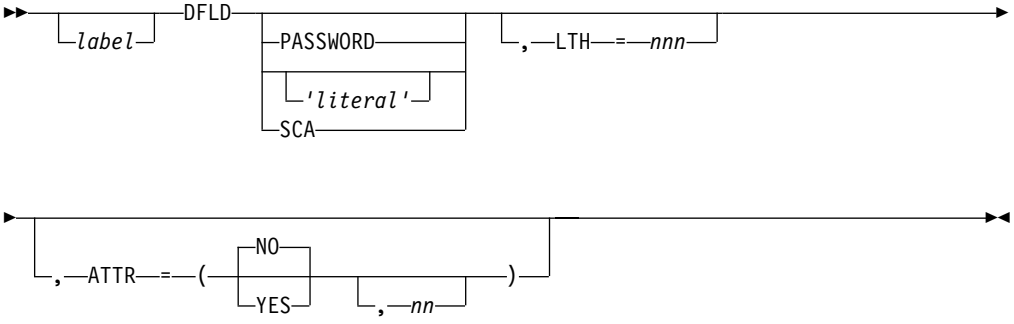
1 MODE=RECORD only

2 MODE=STREAM only

Format for DEV TYPE=DPM-An or DPM-Bn AND DIV TYPE=INPUT:



Format for DEV TYPE=DPM-An or DPM-Bn AND DIV TYPE=OUTPUT:



Parameters:

label

A one- to eight-character alphanumeric name can be specified. This label (dfldname) can be referred to by a message descriptor in transferring data to and from a terminal or remote program. If the repetitive generation function of MFS is used (DO and ENDDO statements), this dfldname should be restricted to 6 characters maximum length. When each repetition of the statement is generated, a 2-digit sequence number (01 to 99) is appended to the label. If the label specified here is greater than 6 characters and repetitive generation is used, the label is truncated at 6 characters, and a 2-digit sequence number is appended to form the 8-character name. No error message is provided if this occurs.

If PASSWORD, SCA, or '*literal*' is specified, label is not valid, and specification of a label will result in an error message. If a DPN, PRN, RDPN, or RPRN dfldname is specified on the DIV statement, the *dfldname* cannot be used as a DFLD label for the current DIV statement.

PASSWORD

Identifies this field as the location of the IMS password field for input messages.

Recommendation: Use the PASSWORD capability in the input message definition. If you specify PASSWORD you cannot refer to the field described by this DFLD statement with a message descriptor. Additionally, if you specify PASSWORD you must omit *label*.

'*literal***'**

Specifies a literal character string to be presented to the device. The length of *literal* cannot exceed 256 bytes for 3270 display devices, 40 bytes for FIDS and FIDS3, 64 bytes for FIDS4, 80 bytes for FID57, 132 bytes for 274X, 256 bytes

for 3270P, and line width for all printer and punch devices. For DPM, the length of *literal* cannot exceed the value specified in the RCDCTL operand.

For 3270 displays, literal fields have the PROT attribute whether specified or not; the NUM attribute is assumed if ALPHA is not specified.

Restriction: If you specify *literal* you cannot refer to the field described by this DFLD statement with a message descriptor. Additionally, if you specify *literal* you must omit *label*.

SCA
Specifies, for DPM definitions only, that SCA information, when sent by the IMS application program or specified in the DSCA, is to be sent in this DFLD.

If SCA is specified, *label* must not be specified.

POS=
Defines the first data position of this field in terms of line (lll), column (ccc), and physical page (pp) of the display format. If pp is omitted, 1 is assumed.

For DEV TYPE=274X, FIN,FIDS,FIDS3,FIDS4, FIDS7,FIJP,FIPB,FIFP,SCS1, or SCS2

lll,ccc
Specifies the record number and position within the record of this field. This form is required if MODE=RECORD. lll and ccc must be greater than or equal to 1.

nnn
Specifies the starting position of this field in STREAM mode input. If not specified, this field starts immediately following the preceding field, or at the left margin if this is the first field. If MODE=STREAM has been specified, and POS= is specified, this form is required. *nnn* must be greater than or equal to 1.

lll,*ccc*,*pp*
Specifies the line, column, and optionally, the physical page number for an output field. lll, ccc, and pp must be greater than or equal to 1.

For DEV TYPE=3270, 3270-An, or 3270P

lll,*ccc*,*pp*
Specifies the line, column, and optionally, the physical page number for an output field. *lll*, *ccc*, and *pp* must be greater than or equal to 1.

For 3270 displays, POS=(1,1) must not be specified. Fields must not be defined such that they wrap from the bottom to the top.

Restriction: On some models of 3270s, the display screen cannot be copied when a field starting on line 1, column 2, has both alphabetic and protect attributes.

For DEV TYPE=DPM-An or DPM-Bn

For DPM devices
The POS= keyword is ignored.

LTH=
Specifies the length of the field. This operand should be omitted if '*literal*' is specified in the positional parameter, in which case the length of *literal* is used

as the field length. Unpredictable output formatting can occur if this operand is used in conjunction with a '*literal*' and the two lengths are different. The specified LTH= cannot exceed the physical page size of the device.

The maximum allowable length for all devices except 3270, 3604 display, and DPM with RCDCT=NOSPAN is 8000 characters. For 3270 displays, the maximum length is one less than screen size. For example, for a 480-character display, the maximum length is 479 characters. For a FIDS display component, the maximum length is 240 characters; for a FIDS3, the maximum length is 480 characters; for a FIDS4, the maximum length is 1024 characters; for a FIDS7, the maximum length is 1920. A length of 0 must not be specified. For DPM, if RCDCT=NOSPAN is specified, the length must be less than or equal to the RCDCTL value, if RCDCTL is less than 8000. If SCA and LTH= are both specified, LTH must be 2.

POS= and LTH= do not include the attribute character position reserved for a 3270 display device or a DFLD with ATTR=YES specified. The inclusion of this byte in the design of display/printer formats is necessary because it occupies the screen/printed page position preceding each displayed/printed field even though it is not accessible by an application program.

When defining DFLDs for 3270 printers, a hardware ATTRIBUTE character is not used. Therefore, fields must be defined with a juxtaposition that does not allow for the attribute character unless ATTR=YES is specified. However, for printers defined as 3270P the last column of a print line (based on FEAT=, WIDTH=, or the device default width) cannot be used. The last column of the line is reserved for carriage control operations performed by IMS. Thus, if the print line specifies 120 (FEAT=120) and the DFLD specifies POS=(1,1),LTH=120 then 119 characters are printed on line 1 and one character on line 2.

For DPM definitions, if OPTIONS=NOSIM2 is specified on the DIV statement, and ATTR=YES or YES,*nn* is specified, 2 bytes plus the extended attributes are added to the length of the DFLD. The first two bytes are reserved for the binary 3270 attribute, (protect, numeric, and so forth.) If OPTIONS=SIM is specified, 1 byte or 1 byte plus the extended attributes is added to the length of the DFLD with ATTR=YES or YES,nn. The first byte of the field is thus reserved for the simulated attribute.

Detectable fields (DET or IDET) must include four positions in POS and LTH for a 1-byte detection designator character and 3 pad characters, unless the detectable field is the last field on a display line, in which case only one position for the detection designator character is required. The detection designator character must precede field data, and pad characters (if required) follow field data. Detection designator and required pad characters must be supplied by the application program or MFLD literal with the field data. Pad characters can also be required in the preceding field on the device.

ATTR=

Defines the display attributes of this field for each of the listed DEV TYPE, DIV TYPE combinations:

- For DEV TYPE=3270 or 3270-An

  Attribute keywords can be specified in any order and only those desired need be specified. The underlined keywords do not have to be specified, because they are defaults.

When two user-defined fields are seperated by two or more characters, MFS generates an undefined field to represent that space in the display buffer. The display attributes for an undefined field are NUM, PROT, and NODISP.

ALPHA|NUM
specifies whether the field should have the numeric attribute. The numeric attribute specifies that the Numeric Lock feature (automatic upshift of data entry keyboard) will be used by the 3275/3277 or 3276/3278. If NUM and PROT (discussed below) are specified for the field, the auto-skip feature is used. That is, upon entry of a character into the last character location of an unprotected field, the cursor automatically skips the field with the NUM and PROT attribute specifications and is positioned to the first character location of the next unprotected field. If an undefined field, as described in the ATTR= parameter above, follows the filled unprotected field, the auto-skip feature is used. This parameter, in conjunction with the PROT parameter below, is used to lock the COPY function. See ″PROT″ for details.

NOPROT|PROT
Specifies whether the field is protected from modification by you. For literal fields, PROT is used and specification of NOPROT is ignored.

The IMS copy function on remote 3270 terminals can be locked by setting the attribute value of protect and alpha for an attribute byte in line 1 and column 1 of a display. When the copy function is locked, it cannot be used to copy the contents of a display to a printer. For more information, see the appendix “IMS Support of Devices” in *IMS/ESA Operations Guide*. The “Local Copy Function” available on the 3274 and 3276 control units is not locked by the attribute setting. The “Local Copy Function” is invoked by the print key.

NODET|DET|IDET
Specifies the detectability of the field through light pen operations. DET specifies a deferred detectable field, while IDET indicates an immediately detectable field. You must provide appropriate designator and pad characters as discussed under the LTH= operand. Note that the 3270 display devices place restrictions on the number of detectable or mixed detectable and nondetectable fields that can precede that last detectable field on a given line.

NORM|NODISP|HI
Specifies the field’s display intensity as normal (NORM), high intensity (HI), or nondisplayable (NODISP). If NODISP is specified, DET or IDET cannot be specified.

When defining a high-intensity (HI) field, including a detection designator character as the first data byte causes the high-intensity (HI) field to be detectable.

NOMOD|MOD
defines whether or not the field-modified-attribute byte should be assumed for this field. MOD causes the terminal to assume the field has been modified by you even though it was not (that is, the modified data tag (MDT) is set in the field-modified-attribute byte). This should not be confused with the PROT attribute which prevents modification by you. MOD is ignored for literal fields.

When MOD is specified, each time MFS sends output for this physical page, the modified attribute is set (unless overridden by dynamic attribute modification).

Related Reading: For a description of when IMS resets modified data tags, see *IMS/ESA Operations Guide*.

STRIP|NOSTRIP
Specifies whether the pen detect designator byte preceding the input field should be stripped (STRIP) before presentation to the application program. If an EGCS attribute is defined for a light-pen-detectable field, you should specify ATTR=NOSTRIP on the DFLD statement and design the application program to bypass or remove the two designator characters from the input data. If ATTR=STRIP is specified or defaulted, MFS will only remove the first designator character and the last character in the field could be lost (truncated).

- For DIV TYPE=OUTPUT and DEV TYPE=274X, 3270P, FIDS, FIDS3, FIDS4, FIDS7, FIFP, FIJP, FIPB, FIS1, or SCS2

  Attribute keywords specify whether (YES) or not (NO) the first byte of this field will be used to display attribute information when the output message includes attribute information for the field. The default is NO. If ATTR=YES is specified, the LTH= and POS= keywords do not have to allow for the simulated attribute byte because the MFS preprocessor adjusts the keyword values internally. The action taken when ATTR=YES is specified is:

CURSOR	(FIDS, FIDS3, FIDS4, and FIDS7 ABSOLUTE output only). The cursor will be positioned to the first position of this field.
NODISP	No data sent regardless of other attributes
HI	An asterisk (*) is placed in the first byte
MODIFIED	An underscore character (_) is placed in the first byte
HI and MODIFIED	An exclamation point (!) is placed in the first byte

  If attribute information is not provided from the output message, the first byte is a blank.

- For DIV TYPE=OUTPUT, DEV TYPE=DPM-An, and DEV TYPE=DPM-Bn, 3270P, FIDS, FIDS3, FIDS4, FIDS7, FIFP, FIJP, FIPB, FIS1, or SCS2

  Attribute keywords specify whether (YES) or not (NO) the first one or two bytes of this field carries existing 3270 attributes and whether extended attributes (nn) are present. The keywords can be used in various combinations as follows:

  YES
  Specifies that the first one or two bytes of this field are used to convey the existing 3270 attributes (in simulated or binary form depending upon the specification of SIM or NOSIM2 respectively on the DIV statement) from the IMS application program to the remote program. (SIM causes MFS to simulate an attribute. NOSIM2 causes MFS to pass the bits exactly as entered.)

  Thus, if ATTR=YES is specified and OPTIONS=SIM or OPTIONS= is not specified, one byte is added to the length of the DFLD. If

OPTIONS=NOSIM2, two bytes are added to the length of the DFLD. These bytes are reserved as the attribute bytes to be transmitted to the remote program.

NO
Specifies that the first one or two bytes of this field will **not** be used to convey the existing 3270 attributes (in simulated or binary form respectively) from the IMS application program to the remote program. This is the default.

nn Is the number of extended attributes that can be dynamically modified, and is a number from 1 to 4. An invalid specification is defaulted to 1. Two additional bytes are added to the length of the DFLD for each attribute specified (2 x *nn*). The additional bytes, which just precede the data, either can (YES) or must not (NO) follow the bytes reserved for the existing 3270 attribute bytes. These bytes are used to convey the extended attributes (in binary form) from the IMS application program to the remote program. The attributes are always transmitted as presented from the IMS application program. They are never simulated or validated.

When used in combination, YES,*nn* specifies that both attributes and extended attributes are to be transmitted. In this case, and depending upon the specification of SIM and NOSIM2 as described above:

YES,*nn*
When specified with SIM, specifies that 3270 simulated attributes (1 byte) plus extended attributes (2 x *nn* bytes) of this field are to be transmitted from the IMS application program to the remote program. The total number of bytes used to convey all of these attributes to the remote program is 1 + (2 x *nn*).

When specified with NOSIM2, specifies that 3270 attributes in binary form (2 bytes) plus extended attributes (2 x *nn* bytes) of this field are to be transmitted from the IMS application program to the remote program. The total number of bytes used to convey all of these attributes, which are all in binary form, to the remote program is 2 + (2 x *nn*).

When used in combination, NO,*nn* specifies that only extended attributes are transmitted. Thus, the number of bytes transmitted, in binary form, is (2 x *nn*) only.

Valid specifications and the number of bytes which must be reserved are:

```
For  DIV ,OPTION=NOSIM2  then:
DFLD    ,ATTR=(YES,nn)     2 + (2 × nn)
DFLD    ,ATTR=(NO,nn)      2 × nn
DFLD    ,ATTR=(,nn)        2 × nn
DFLD    ,ATTR=YES          2
DFLD    ,ATTR=NO           0
For  DIV ,OPTION=SIM or not specified  then:
DFLD    ,ATTR=(YES,nn)     1 + (2 × nn)
DFLD    ,ATTR=(NO,nn)      2 × nn
DFLD    ,ATTR=YES          1
DFLD    ,ATTR=NO           0
```

EATTR=
Is valid for output DFLDs only and defines the extended attributes of this field for DEV TYPE=3270, 3270-An, 3270P, or SCS1.

Not all extended attributes apply to all device types. To ensure that your specifications for your device types are correct, refer to the component description manual for your device.

The operands specify:

- Additional field highlighting
- Field color
- Field outlining
- Input control
- Validation to be performed
- Local ID of the programmed symbol buffer

Characters are selected from the programmed symbol buffer and placed in the field. These operands can be specified in any order. When the device default value is selected for an operand, it is used to hold a place in the data stream to permit application program modification of the attribute so specified.

For details on modifying these attributes, see “Extended Field Attributes for Output Devices” on page 211.

To specify the additional highlighting for the field use the following:

HD	device default
HBLINK	blink
HREV	reverse video
HUL	underline

To specify the field’s color use the following:

- **BLUE**
- **RED**
- **PINK**
- **GREEN**
- **TURQ(uoise)**
- **YELLOW**
- **CD**
- **NEUTRAL**

The last two operands are used as follows:

CD	Used to specify the default.
NEUTRAL	Used to specify device-dependent. The particular color displayed for NEUTRAL is device-dependent. In general, NEUTRAL is white on displays and black on printers with single-plane programmed symbols and as multicolored on displays or printers with tri-plane programmed symbols.

The following five operands—PX'00', PX'*hh*', PC'*c*', EGCS, and EGCS'*hh*'—are mutually exclusive. That is, a field can be specified as having one of these characteristics, but not a combination thereof. For all 3270 devices, MFS does

not verify that any specified character set has been properly loaded. The programmed symbol buffers can be loaded by an IMS application program using the MFS bypass.

PX'00'|PX'*hh***'|PC'***c***'**
Specifies a value that must correspond to the local ID specified for a programmed symbol buffer already loaded or to the EGCS programmed symbol buffer.

PX'00'
Is the same as no specification, except that it allows an application program to specify a programmed symbol buffer for the field through dynamic modification of the programmed symbol attribute.

PX'*hh***'**
Is a hexadecimal character in the range X'40' through X'FE'.

PC'*c***'**
Is a hexadecimal character within the range X'40' through X'FE'.

EGCS|EGCS'*hh***'**
Is valid only on output DFLDs for the 3270 display. SCS1 device types can specify EGCS only and not EGCS '*hh*'.

When an extended graphic character set literal is specified on a DFLD statement, the extended graphic character set attribute is forced—that is, you do not have to code EATTR=EGCS'*hh*' for 3270 displays or EATTR=EGCS for SCS1 device types. For 3270 displays, a programmed symbol value of X'F8' is set.

Restriction: The IMS application program cannot modify the SCS1 DFLD extended graphic character set attribute.

When defining an EGCS field for a 3283 Model 52, the length must be an even number. If the EGCS field spans device lines, WIDTH= and POS= should be specified so that an even number of print positions are reserved on each of the device lines.

EGCS
Specifies the field attribute for the field as Extended Graphic Character Set. Also specifies the field attribute for the field as Double Byte Character Set.

EGCS'*hh***'**
'*hh*' is the programmed symbol value that is used. The value for '*hh*' can be any hexadecimal value from X'40' through X'FE' or X'00'. If '*hh*' is omitted from the extended graphic character set specification for a 3270 display, a programmed symbol value of X'F8' is assumed. '*hh*' is ignored if specified for an SCS1 device.

To define an EBCDIC field that can be dynamically modified by the IMS application program to accept extended graphic character set data, the programmed symbol attribute should be specified as EGCS'00'.

VDFLD|VMFILL|VMFLD|VMFILL,VMFLD
Defines the type of validation for the field as follows:

VDFLD	default
VMFILL	mandatory fill

VMFLD	mandatory field
VMFILL,VMFLD	a combination of mandatory fill and mandatory field

If a field is defined as protected (ATTR=PROT) or if it is a literal with validation attributes specified, then the validation attribute specifications are reset and a message is issued.

The following are used to specify field outlining:

OUTL'*hh***'**	Field outlining with field outlining value '*hh*'
OUTL	Device default
BOX	Box
RIGHT, LEFT, UNDER, OVER	Lines that can be specified individually or in combination

Field outlining value '*hh*' is a two-digit hexadecimal number between X'00' and X'0F'. If any other value is specified, the device default, X'00', is assumed. Table 46 shows the values for the field outlining patterns:

Table 46. Field Outlining Values

Value	UNDER	RIGHT	OVER	LEFT
00				
01	X			
02		X		
03	X	X		
04			X	
05	X		X	
06		X	X	
07	X	X	X	
08				X
09	X			X
0A		X		X
0B	X	X		X
0C			X	X
0D	X		X	X
0E		X	X	X
0F	X	X	X	X

Field outlining for 3270 displays and SCS1 printers can be dynamically modified by code in an application program. The position of left, right, over, and underlines differ according to the device.

The following is a brief description of field outlining for the IBM 5550 family (as 3270) of devices.

3270 display Left and right lines are printed in the position of the 3270 basic attribute byte. The overline of the current line and the underline of the preceding line are the same line.

The underline for the 24th line is the same line as the line separating the application program area and your message area.

SCS1 printer Left and right lines are printed in the byte reserved by MFS

before and after the current field. The overline of the current line and the underline of the preceding line are the same line. When an underline is specified in the last line of the page, an underline is drawn in the last line of the page, and an overline is drawn on the first line of the next page.

If one byte space exists between two adjacent fields, the right line of the first field is the same line as the left line of the second field.

MIX|MIXD|MIX'*nn***'|MIXS|MIXS'***nn***'**
Specify a DBCS/EBCDIC mixed field.

3270 display

MIX DBCS/EBCDIC mixed field

MIXD device default

Input control for the 3270 display can be dynamically modified by the application program. Refer to "Dynamic Modification of DBCS/EBCDIC Mixed Data" on page 290 for more information on dynamic modification.

SCS1 printer

MIX	DBCS/EBCDIC mixed field with SO/SI blank print option.
MIXS	DBCS/EBCDIC mixed field with SO/SI blank print suppress option.
MIX'*nn***'**	'*nn*' is the maximum number of SO/SI pairs. DBCS/EBCDIC mixed field with SO/SI blank print option.
MIXS'*nn***'**	'*nn*' is the maximum number of SO/SI pairs. DBCS/EBCDIC mixed field with SO/SI blank print suppress option.

The '*nn*' is buffer information used by MFS message editor and must be a two-digit decimal number between 01 and 31. If MIX or MIXS is specified, the MFS default is calculated as follows:

MIX DFLD length divided by 5 plus 1, or 31, whichever is smaller.

MIXS DFLD length divided by 3 plus 1, or 31, whichever is smaller.

When a field spans continuation lines, the number '*nn*' obtained from the field length with either of the above methods plus 1, is assigned to each line.

With the SCS1 printer, when DBCS/EBCDIC mixed data spanning across continuation lines is split at a DBCS character, MFS replaces the last character with a blank and places that character at the beginning of the next line. As a result, one print position is lost.

PEN=
Specifies a literal to be selected or an operator control function to be performed when this field is detected. If (1) '*literal*' is specified, (2) the field is defined as immediately detectable (ATTR= operand), and (3) contains the null or space designator character, the specified literal is placed in the field referred to by the

PEN operand of the preceding DEV statement when the field is detected (if no other device fields are modified). If another field on the device is modified, a question mark (?) is provided instead of the literal. Literal length must not exceed 256 bytes.

If (1) a control function is specified, (2) the field is defined as immediately detectable (ATTR= operand), and (3) contains the null or space designator character, the specified control function is performed when the field is detected and no other device fields are modified. If another field on the device is modified, a question mark (?) is provided and the function is not performed. Control functions that can be specified are:

NEXTPP—PAGE ADVANCE
Specifies a request for the next physical page in the current output message. If no output message is in progress, no explicit response is made.

NEXTMSG—MESSAGE ADVANCE
specifies a request to dequeue the output message in progress (if any) and to send the next output message in the queue (if any).

NEXTMSGP—MESSAGE ADVANCE PROTECT
Specifies a request to dequeue the output message in progress (if any), and send the next output message or return an information message indicating that no next message exists.

NEXTLP—NEXT LOGICAL PAGE
Specifies a request for the next logical page of the current message.

ENDMPPI—END MULTIPLE PAGE INPUT
Specifies the end of a multiple physical page input message.

ENDMPPI is valid only if data has been received and will not terminate multiple page input (MPPI) in the absence of data entry.

OPCTL=
Specifies the name of a table, defined by a TABLE statement, that is to be checked for operator control requests when this device field is received. OPCTL processing occurs when the input device data is processed. If a control function is selected, in most cases the control function is performed immediately; no IMS input message is created.

SLDI=
For SCS1 printers, specifies the line density for an output message in lines per inch. (See also SLDP=.) SLDI= can also be specified on the DEV statement. SLDI= is validated for a value from 1 through 72. The value specified must be consistent with the architecture of the device for which this value is specified (see the appropriate device or component manual).

If SLDI= is specified both on the DEV statement and the DFLD statement, two SLD data streams are created. One is sent at the beginning of a message to set the line density. The second is sent within the message, just prior to the field on which the SLDI= specification is encountered, but after any vertical tabs and new line characters. The SLDI= specification within the message changes the line density from that set at the beginning of the message, and this latter line density remains in effect until explicitly reset.

SLDP=
For SCS1 printers, specifies the line density for an output message in points per inch. (See also SLDI=.) SLDP= can also be specified on the DEV

statement. SLDP= is validated for a value from 1 through 72. The value specified must be consistent with the architecture of the device for which this value is specified (see the appropriate device or component manual).

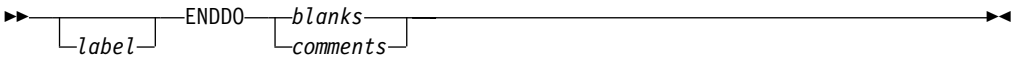
If SLDP= is specified both on the DEV statement and the DFLD statement, two SLD data streams are created. One is sent at the beginning of a message to set the line density. The second is sent within the message, just prior to the field on which the SLDP= specification is encountered, but after any vertical tabs and new line characters. The SLDP= specification within the message changes the line density from that set at the beginning of the message, and this latter line density remains in effect until explicitly reset.

Recommendation: Be careful, when defining set line density (SLDx) keywords, to ensure that forms alignment is maintained. If SLDx= is improperly defined, the forms might not align properly. Also, note that SLDI= and SLDP= are mutually exclusive. Neither SLDI= nor SLDP= can occur on a DFLD statement between a DO and an ENDDO statement.

ENDDO Statement

The ENDDO statement terminates the group of DFLD statements that are to be repetitively generated. The generated DFLD statements are printed immediately following the ENDDO statement. An ENDDO statement is required for each DO statement entered in this definition.

Format:



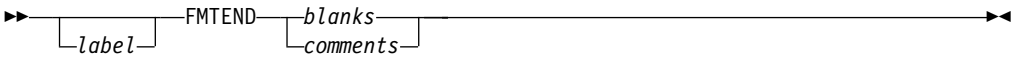
Parameters:

label
A one- to eight-character alphanumeric name can be specified. It is not used.

FMTEND Statement

The FMTEND statement terminates a device format definition and is required as the last statement in the device format definition. If this is the end of the input to SYSIN processing, the FMTEND statement must be followed by an END compilation statement.

Format:



Parameters:

label
A one- to eight-character alphanumeric name can be specified. It is not used.

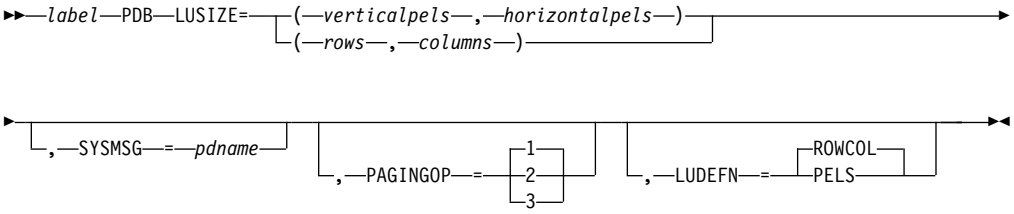
Partition Set Definition Statements

PDB Statement

This statement initiates and defines a partition set (a Partition Descriptor Block) for 3290 and 3180 devices in partitioned format mode. The PDB statement contains several parameters that describe certain characteristics of the entire partition set. Its name is referenced by the PDB keyword of a DEV statement if a partition set is required to format logical pages of a message.

At least one PD statement must be specified within each PDB. Note, however, that for a 3180 in partitioned format mode, **only one** PD statement should be specified within each PDB. This is because only one partition can be specified for the 3180. There are additional differences in specifications that can be made for the partitioned 3180 and 3290 which are described in the following section.

Format:



Parameters:

label
A one- to eight-character alphanumeric name (pdbname) for the PDB must be specified.

LUSIZE=
Describes the physical size of the Logical Unit display for which the PDB is defined. If LUDEFN=PELS, the size is specified in picture elements (pels). If LUDEFN=ROWCOL, the size is specified in rows and columns (this is the default value). For the 3180, LUSIZE must be specified in terms of rows and columns.

SYSMSG=
Specifies the partition name (*pdname*) for displaying system messages. The system message partition should have only one field defined. This DFLD should be defined as at least LTH=79 so the system message is not truncated.

If the current PDB defines a system message partition, then all system messages are directed to this partition. If a system message partition is not defined, but a SYSMSG field is defined in the current DOF, the system message is directed to the system message field of the active partition. Finally, if the current PDB does not define a partition for system messages and the DOF does not define a field for that purpose, a system message destroys the current partitioned format mode and the 3290 returns to standard format mode.

PAGINGOP=
Specifies the option number (1, 2, or 3) for the partition page presentation algorithm. These three algorithms specify different ways of presenting the initial

pages of the message to the partitions of the partition set. They also specify what paging actions result when you enter paging requests from the 3290 device.

The default of 1 must be accepted or specified on this operand for 3180 formats.

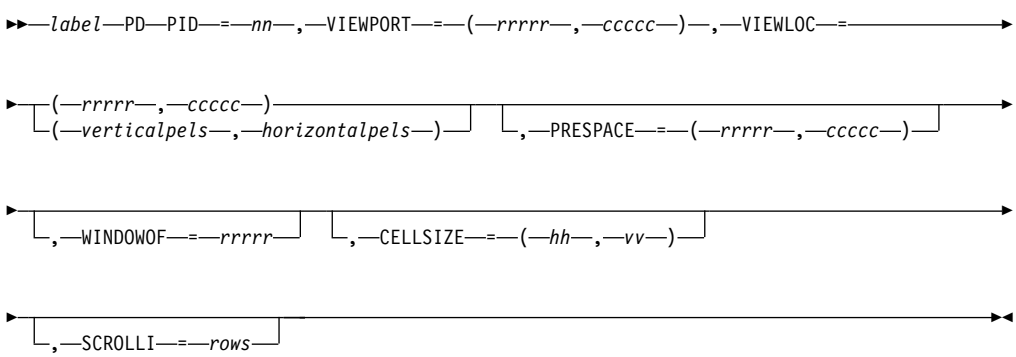
LUDEFN=
Indicates whether the LUSIZE parameter in the PDB statement and the VIEWLOC parameter in the PD statements are specified in rows and columns or in pels. LUDEFN is optional if all the PD statements use the same cell size and the default (ROWCOL) is acceptable. Note that ROWCOL must be specified or accepted as the default for 3180 formats.

If two or more PD statements within the same PDB specify different cell sizes, PELS **must** be chosen.

PD Statement

The Partition Definition statement defines one partition and its presentation space. Every partition set described by a PDB statement must contain at least one PD statement. Note, however, that for a 3180 in partitioned format mode, **only one** PD statement should be specified within each PDB.

Format:



Parameters:

label
A one- to eight-character alphanumeric name (pdname) must be specified. This name is referenced by the DPAGE statement to associate a logical page with its appropriate partition.

PID=
Specifies a partition identifier number for the partition. Values 00 through 15 are valid for 3290 formats. Each partition must have a unique PID. A value of 00 must be specified for 3180 formats, because only one partition need be identified.

VIEWPORT=
Specifies the size of the viewport for the partition. *rrrrr* indicates rows and *ccccc* indicates columns. For the 3180 device, the following restrictions apply:

- If the number of columns is greater than or equal to 80, then the number of rows must be less than or equal to 43.

- If the number of columns is greater than 80 and less than or equal to 132, then the number of rows must be less than or equal to 27.

VIEWLOC=
Specifies the location of the viewport on the display screen, in terms of the distance offset from the top left of the screen. When the LUDEFN parameter of the PDB statement is ROWCOL, the distance is expressed in rows and columns. *rrrrr* indicates rows and *ccccc* indicates columns. When the LUDEFN parameter is PELS, the distance is expressed in the number of pels from the top of the screen and the number of pels from the left of the screen. When defining formats for the 3180, VIEWLOC must be expressed in rows and columns.

PRESPACE=
Indicates the size of the presentation space buffer in rows and columns. *rrrrr* indicates rows and *ccccc* indicates columns. If this parameter is not specified, the default is the size of the viewport specified on the VIEWPORT parameter. When this parameter is specified, the columns parameter is optional and defaults to the columns specification on the VIEWPORT parameter. If columns are specified, they must be the same as the columns specified in the VIEWPORT parameter.

When specifying this operand for 3180 formats, the product of the number of rows times the number of columns might not be greater than 7680.

WINDOWOF=
Indicates the initial offset in rows of the top edge of the view window from the top of the presentation space. The window maps the portion of the presentation space to be displayed onto the viewport on the screen. During interactive processing, change the offset by scrolling. The default value of WINDOWOF is zero.

CELLSIZE=
Indicates the number of horizontal and vertical pels in a character cell. Note that this specification is in an unusual order for MFS. That is, the **width** of the character cell is specified first, **then the height**. This is the reverse of the usual MFS order.

For the 3290, the default is 6 X 12 PEL (for a small character). Valid values for the 3290 are 6 X 12 to 12 X 31, or the value 00 X 00. If the value is 00 X 00, the 3290 device will select a cell size for optimum readability. This prevents MFS from making validity checks on the viewport locations and possible overlaps. Therefore, be careful to choose viewport size and location specifications accurately.

For the 3180, this operand should be specified according to usable screen area size as follows:

- CELLSIZE=(12,12)
  - 24 x 80
  - 32 x 80
  - 43 x 80
- CELLSIZE=(10,16)
  - 27 x 132

SCROLLI=
Indicates the number of rows that are scrolled when the scrolling function is used. The default scrolling increment is one row. If the scrolling increment is

larger than the viewport size, part of the presentation space is not viewable on the screen. Specifying 0 as the scrolling increment disables the scrolling function.

PDBEND Statement

The PDBEND statement terminates a partition set definition (a partition descriptor block) and is required as the last statement of the definition. If this is the end of the input to SYSIN processing, the PDBEND statement must be followed by an END compilation statement.

Format:

Table Definition Statements

TABLE Statement

The TABLE statement initiates and names an operator control table that can be referred to by the OPCTL keyword of the DFLD statement (“DFLD Statement” on page 370). For a discussion of the use of the table, see “Operator Control Tables” on page 237. The TABLE statement, and the IF and TABLEEND statements that follow, must be outside of a MSG or FMT definition.

Format:

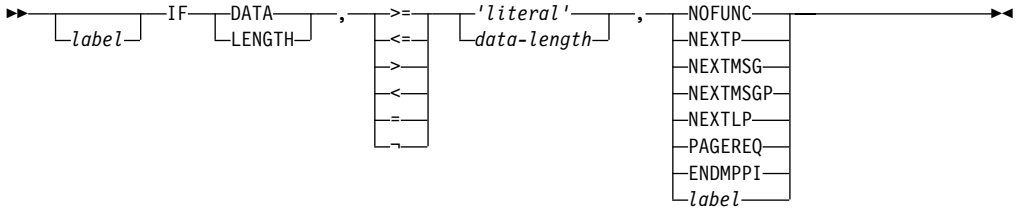
Parameters:

tablename
A 1- to 8-byte alphanumeric name for the table must be specified.

IF Statement

The IF statement defines an entry in the table named by the previous TABLE statement. Each IF statement defines a conditional operation and an associated control or branching function to be performed if the condition is true.

Format:



Parameters:

label
A one- to eight-character alphanumeric name can be specified. This label is required if a previous IF statement contained a branch function.

DATA
Specifies that the conditional operation is to be performed against the data received from the device for the field.

LENGTH
Specifies that the conditional operation is testing the number of characters entered for the field. The size limit for this field is the same as for DFLDs (see "DFLD Statement" on page 370).

=,<,>, ¬,≤,≥
Specify the conditional relationship that must be true to invoke the specified control function.

'*literal*'
Is a literal string to which input data is to be compared. The compare is done before the input is translated to upper case. If '*literal*' is specified, DATA must be specified in the first operand. If the input data length is not equal to the *literal* string length, the compare is performed with the smaller length, unless the conditional relationship is ¬ and the data length is zero, in which case the control function is performed. If the input is in lowercase, the ALPHA statement should be used and the literal coded in lowercase.

data-length
Specifies an integer value to which the number of characters of input data for the field is compared.

NOFUNC
Specifies that conditional function testing is to be terminated.

NEXTPP—PAGE ADVANCE
Specifies a request for the next physical page in the current output message. If no output message is in progress, no explicit response is made.

NEXTMSG—MESSAGE ADVANCE
Specifies a request to dequeue the output message in progress (if any) and to send the next output message in the queue (if any).

NEXTMSGP—MESSAGE ADVANCE PROTECT
Specifies a request to dequeue the output message in progress (if any), and either send the next output message or return an information message indicating that no next message exists.

NEXTLP—NEXT LOGICAL PAGE
Specifies a request for the next logical page of the current message.

PAGEREQ—LOGICAL PAGE REQUEST
Specifies that the second through last characters of input data are to be considered as a logical page request.

ENDMPPI—END MULTIPLE PAGE INPUT
Specifies the end of multiple physical page input (this input is the last for the message being created).

label
Specifies that testing is to continue with the IF statement bearing the label (branch). The label must be placed on an IF statement that follows the current statement in the TABLE definition (that is, it must be a forward branch function).

TABLEEND Statement

The TABLEEND statement establishes the end of a table definition. If this is the end of the input to SYSIN processing, the TABLEEND statement must be followed by an END compilation statement.

Format:

Parameters:

label
A one- to eight-character alphanumeric name can be specified. It is not used.

Compilation Statements

ALPHA Statement

The ALPHA statement specifies a set of characters to be considered alphabetic by the MFS language utility for the purpose of defining valid field names and literals.

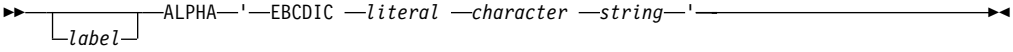
Restriction: The following characters cannot be made alphabetic using ALPHA.

b ¢ * < (+ | !! *) ; ¬ .
- / , % _ > ? :
' = ”
0 through 9

The characters A through Z, &; (X'50'), #, $, and @ are always considered alphabetic by the MFS language utility.

All the characters referred to above are known as standard characters. Therefore, all other characters are referred to as nonstandard characters.

Format:



Parameters:

label
A one- to eight-character alphanumeric name can be specified. It is not used.

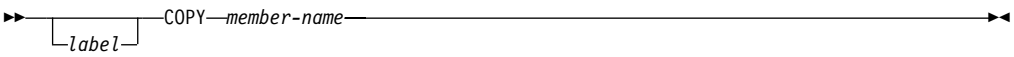
'*literal character string*'
Specifies the characters to be considered alphabetic by the MFS language utility. The use of an EGCS literal in an ALPHA statement causes an ERROR message.

COPY Statement

The COPY statement invokes a copy of a member of the partitioned data set represented by the SYSLIB DD statement. The copied member can request the nested copy of another member. The member to be copied cannot already exist at a higher level in a nested chain of copy requests. The nesting level available for

copy is limited only by the amount of storage available to the language utility preprocessor. The level of the COPY statement is indicated to the right of each printed COPY record.

Format:



Parameters:

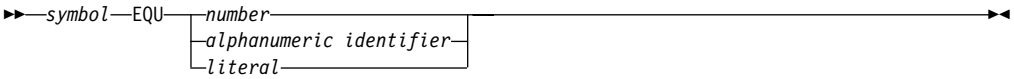
label
A one- to eight-character alphanumeric name can be specified. It is not used.

member-name
Specifies the name of the partitioned data set member to be copied into the input stream of the utility preprocessor.

EQU Statement

The EQU statement defines a symbol as a substitution variable. All subsequent occurrences of the symbol in the operand field of a statement is replaced by the value specified in the operand field of the EQU statement.

Format:



Parameters:

symbol
Specifies the symbol to be equated to the value specified in the operand field. The symbol must be a one- to eight-character alphanumeric identifier, the first character of which must be alphabetic.

number
Specifies the value to be represented by the symbol, and consists of 1 to 256 decimal digits.

alphanumeric identifier
Specifies the value to be represented by the symbol, and consists of 1 to 256 alphanumeric characters, the first of which must be alphabetic.

literal
Specifies the value to be represented by the symbol, and consists of 1 to 256 valid characters (not counting embedded second quotes), enclosed in quotes. The characters within the leading and trailing quotes replace the symbol when substitution occurs. An EGCS literal cannot be equated if any hexadecimal value within the literal is a X'7D' (a single quote character).

A symbol used in an equate (EQU) statement can be re-equated to another value.

There are no reserved words that cannot be used as symbols on the EQU statement. However, when defining symbols do not use a symbol as one of the words used by the MFS statement operands. Otherwise, the intended function of the MFS word cannot be used.

Example: Consider the following equate statement:

```
NOPROT EQU PROT
```

Then if one DFLD specifies ATTR=NOPROT and another DFLD specifies ATTR=PROT, both DFLDs would generate the protect attribute (PROT).

Restriction: Once an MFS word is equated, it cannot be restored to its original symbol. In other words, a symbol cannot be equated to itself.

Concatenated EQU Statements

A period (.) can be used to concatenate two equated values or one value and specific data, providing that at the point of concatenation a delimiter exists.

Example: Consider the following EQU statements:

```
A  EQU ATTR

AE EQU 'ATTR='

P  EQU '(PROT,NUM)'

EP EQU '=(PROT,NUM)'
```

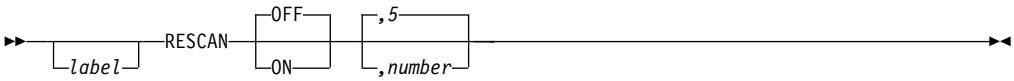
The following all produce the same results:

```
ATTR=(PROT,NUM)

ATTR=P

AE.P

A.EP

A=P
```

RESCAN Statement

The RESCAN statement controls the operation of EQU statements during replacement mode.

Format:



Parameters:

label
A one- to eight-character alphanumeric name can be specified. It is not used.

OFF|ON
Specifies whether (ON) or not (OFF) replacement text should be rescanned for further substitution. The default is OFF unless a number is specified.

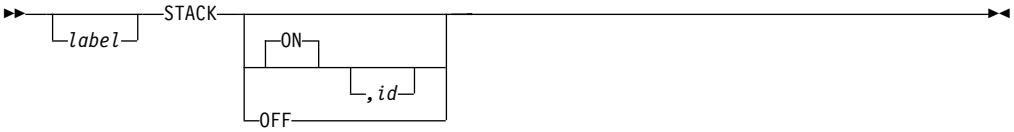
If ON is specified, replacement text can invoke further substitution within the substituted text up to a maximum number of occurrences.

5|number
Specifies how many times further substitution is allowed in a single rescan substitution. The default is 5. If recursive substitutions are attempted beyond the 'number', an error message is issued and substitution terminates. RESCAN ON,0 will be interpreted as RESCAN OFF.

STACK Statement

The STACK statement is used to delineate one or more SYSIN or SYSLIB records, and to request that those records, once processed, be kept (stacked) in processor storage for reuse at a later time. A stack of SYSIN/SYSLIB records must not contain STACK and UNSTACK statements. The letter S to the right of each printed record indicates that it is being stacked for future use.

Format:



Parameters:

label
A 1- to 8-character name can be specified. It is not used.

ON
Specifies the beginning of a stack of SYSIN/SYSLIB records. ON is the default, and it does not have to be specified to begin stacking.

OFF
Specifies the end of a stack of SYSIN/SYSLIB records.

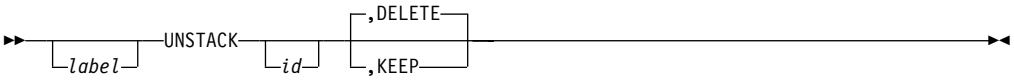
id Specifies the one-to eight-character alphanumeric name for the record stack. If the compilation only uses one stack, no ID is required; MFS assigns an ID of eight blanks to the stack.

When multiple stacking operations are requested, all stacks should be uniquely identified; one unnamed stack is permitted.

UNSTACK Statement

The UNSTACK statement requests retrieval of a previously processed stack of SYSIN/SYSLIB records and specifies whether the retrieved stack should be deleted after processing. The letter U to the right of each printed record indicates that it is being read from the processor storage stack for processing.

Format:



Parameters:

label
A one- to eight-character alphanumeric name can be specified. It is not used.

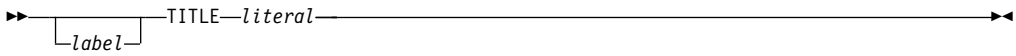
id Specifies the 1- to 8-character identifier of the stack to be retrieved and processed. If no ID is specified, MFS retrieves the stack identified by eight blanks.

DELETE|KEEP
Specifies whether (KEEP) or not (DELETE) the stack should be retained after retrieval and processing. The default is DELETE.

TITLE Statement

The TITLE statement is used to specify the heading to appear on the SYSPRINT listing.

Format:



Parameters:

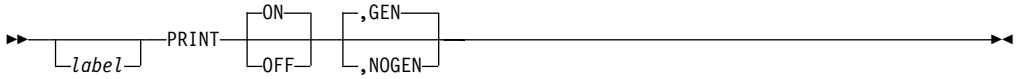
label
A one- to eight-character alphanumeric name can be specified. It is not used.

literal
Specifies the heading to be printed on the output listing. The heading can be specified as an EGCS literal. An EGCS literal of more than 108 bytes causes an error message.

PRINT Statement

The PRINT statement provides printing specifications for the SYSPRINT listing.

Format:



Parameters:

label
A one- to eight-character alphanumeric name can be specified. It is not used.

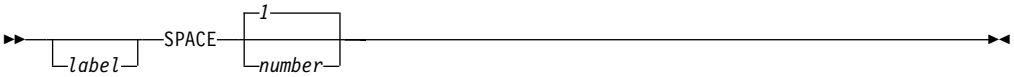
ON|OFF
Specifies whether (ON) or not (OFF) a listing should be printed. The default is ON.

GEN|NOGEN
Specifies whether (GEN) or not (NOGEN) the intermediate text blocks (ITBs) should be printed in hexadecimal following the statement at the left margin. If PRINT GEN is used following the ENDDO statement, all definitions generated for the iterative DO group are printed. The default is GEN.

SPACE Statement

The SPACE statement specifies the number of lines to skip when output is printed. The SPACE statement is printed.

Format:



Parameters:

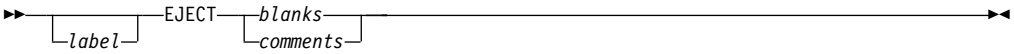
label
A one- to eight-character alphanumeric name can be specified. It is not used.

1|*number*
Specifies how many lines to skip after this statement is encountered. The default is 1.

EJECT Statement

The EJECT statement is used to eject a page in an output listing. The EJECT statement is printed.

Format:



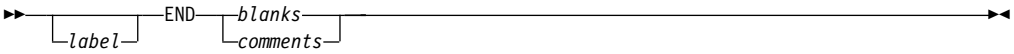
Parameters:

label
A one- to eight-character alphanumeric name can be specified. It is not used.

END Statement

The END statement is used to define the end of the input to SYSIN processing. If this statement is omitted, one is provided and an error message is issued.

Format:



Parameters:

label
A one- to eight-character alphanumeric name can be specified. It is not used.

Part 3. IMS Adapter for REXX

Chapter 11. IMS Adapter for REXX . . . 399
Addressing Other Environments . . . 400
REXX Transaction Programs . . . 400
IMS Adapter for REXX Overview Diagram . . . 402
IVPREXX Sample Application . . . 403
IVPREXX Example 1 . . . 403
IVPREXX Example 2 . . . 403
IVPREXX Example 3 . . . 404
IVPREXX Example 4 . . . 404
REXXTDLI Commands . . . 404
Addressable Environments . . . 405
REXXTDLI Calls . . . 405
Return Codes . . . 405
Parameter Handling . . . 406
Example DL/I Calls . . . 407
Environment Determination . . . 408
REXXIMS Extended Commands . . . 408
DLIINFO . . . 409
Format . . . 409
Usage. . . . 409
Example . . . 409
IMSRXTRC. . . . 410
Format . . . 410
Usage. . . . 410
Example . . . 410
MAPDEF . . . 410
Format . . . 410
Usage. . . . 411
Example . . . 412
MAPGET . . . 413
Format . . . 413
Usage. . . . 413
Examples . . . 413
MAPPUT . . . 413
Format . . . 413
Usage. . . . 414
Examples . . . 414
SET . . . 414
Format . . . 414
Usage. . . . 415
Examples . . . 415
SRRBACK and SRRCMIT . . . 415
Format . . . 415
Usage. . . . 416
STORAGE . . . 416
Format . . . 416
Usage. . . . 416
Example . . . 417
WTO, WTP, and WTL . . . 417
Format . . . 417
Usage. . . . 417
Example . . . 418
WTOR . . . 418

© Copyright IBM Corp. 1974, 2000

Format 418
Usage. 418
Example 418
IMSQUERY Extended Functions 418
Format 418
Usage. 419
Example 420

Chapter 12. IMS Adapter for REXX Exit Routine 421
Environment 421
Parameters 421

Chapter 13. Sample Execs Using REXXTDLI 425
SAY Exec: For Expression Evaluation 425
PCBINFO Exec: Display PCBs Available in Current PSB 426
PART Execs: Database Access Example 429
PARTNUM Exec: Show Set of Parts Near a Specified Number 430
PARTNAME Exec: Show a Set of Parts with a Similar Name 431
DFSSAM01 Exec: Load the Parts Database. 431
DOCMD: IMS Commands Front End 432
IVPREXX: MPP/IFP Front End for General Exec Execution 436

Chapter 11. IMS Adapter for REXX

The IMS adapter for REXX (REXXTDLI) provides an environment in which IMS users can interactively develop REXX EXECs under TSO/E (time-sharing option extensions) and execute them in IMS MPPs, BMPs, IFPs, or Batch regions.

This product does not compete with DFSDDLT0 but is used as an adjunct. The IMS adapter for REXX provides an application programming environment for prototyping or writing low-volume transaction programs.

The REXX environment executing under IMS has the same abilities and restrictions as those documented in the *TSO/E Version 2 Procedures Language MVS/REXX Reference*. These few restrictions pertain to the absence of the TSO, ISPEXEC, and ISREDIT environments, and to the absence of TSO-specific functions such as LISTDS. You can add your own external functions to the environment as documented in the *TSO/E Version 2 Procedures Language MVS/REXX Reference*.

IMS calls the REXX EXEC using IRXJCL. When this method is used, Return Code 20 (RC20) is a restricted return code. Return Code 20 is returned to the caller of IRXJCL when processing was not successful, and the EXEC was not processed.

A REXX EXEC runs as an IMS application and has characteristics similar to other IMS-supported programming languages, such as COBOL. Programming language usage (REXX and other supported languages) can be mixed in MPP regions. For example, a COBOL transaction can be executed after a REXX transaction is completed, or vice versa.

REXX flexibility is provided by the following:

- REXX is an easy-to-use interpretive language.
- REXX does not require a special PSB generation to add an EXEC and run it because EXECs can run under a standard PSB (IVPREXX or one that is established by the user).
- The REXX interface supports DL/I calls and provides the following functions:
  - Call tracing of DL/I calls, status, and parameters
  - Inquiry of last DL/I call
  - Extensive data mapping
  - PCB specification by name or offset
  - Obtaining and releasing storage
  - Messaging through `WTO`, `WTP`, `WTL`, and `WTOR`

The following system environment conditions are necessary to run REXX EXECs:

- DFSREXX0 and DFSREXX1 must be in a load library accessible to your IMS dependent or batch region; for example, STEPLIB.
- DFSREXX0 is stand-alone and must have the RENT option specified.
- DFSREXX1 must be link-edited with DFSLI000 and DFSCPIR0 (for `SRRCMIT` and `SRRBACK`) and optionally, DFSREXXU. The options must be REUS, not RENT.
- IVPREXX (copy of DFSREXX0 program) must be installed as an IMS transaction program. IVP (Installation Verification Program) installs the program. For more information, see “REXX Transaction Programs” on page 400.
- The PSB must be defined as assembler language or COBOL.

© Copyright IBM Corp. 1974, 2000

- SYSEXEC DD points to a list of data sets containing the REXX EXECs that will be run in IMS. You must put this DD in your IMS dependent or batch region JCL.
- SYSTSPRT DD is used for REXX output, for example tracing, errors, and SAY instructions. SYSTSPRT DD is usually allocated as SYSOUT=A or another class, depending on installation, and must be put in the IMS dependent or batch region JCL.
- SYSTSIN DD is used for REXX input because no console exists in an IMS dependent region, as under TSO. The REXX PULL statement is the most common use of SYSTSIN.

In this Chapter:

- "Addressing Other Environments"
- "REXX Transaction Programs"
- "REXXTDLI Commands" on page 404
- "REXXTDLI Calls" on page 405
- "REXXIMS Extended Commands" on page 408

Related Reading: See *TSO/E Version 2 Procedures Language MVS/REXX Reference* for more information on SYSTSPRT and SYSTSIN.

Addressing Other Environments

Use the REXX ADDRESS instruction to change the destination of commands. The IMS Adapter for REXX functions through two host command environments: REXXTDLI and REXXIMS. These environments are discussed in "Addressable Environments" on page 405. Other host command environments can be accessed with an IMS EXEC as well.

MVS is an environment provided by TSO in both TSO and non-TSO address spaces. MVS is used to run other programs such as EXECIO for file I/O. IMS does not manage the MVS EXECIO resources. An IMS COMMIT or BACKOUT, therefore, has no effect on these resources. Because EXECIO is not an IMS-controlled resource, no integrity is maintained. If integrity is an issue for flat file I/O, use IMS GSAM, which ensures IMS-provided integrity.

If APPC/MVS is available (MVS 4.2 or higher), other environments can be used. The environments are:

APPCMVS	Used for MVS-specific APPC interfacing
CPICOMM	Used for CPI Communications
LU62	Used for MVS-specific APPC interfacing

Related Reading: For more information on addressing environments, see *TSO/E Version 2 Procedures Language MVS/REXX Reference*.

REXX Transaction Programs

A REXX transaction program can use any PSB definition. The definition set up by the IVP for testing is named IVPREXX. A section of the IMS stage 1 definition is shown in the following example:

```
*********************************************************************
*   IVP APPLICATIONS DEFINITION FOR DB/DC, DCCTL                    *
*********************************************************************
         APPLCTN GPSB=IVPREXX,PGMTYPE=TP,LANG=ASSEM  REXXTDLI SAMPLE
           TRANSACT CODE=IVPREXX,MODE=SNGL,                          X
               MSGTYPE=(SNGLSEG,NONRESPONSE,1)
```

This example uses a GPSB, but you could use any PSB that you have defined. The GPSB provides a generic PSB that has an IOPCB and a modifiable alternate PCB. It does not have any database PCBs. The language type of ASSEM is specified because no specific language type exists for a REXX application.

Recommendation: For a REXX application, specify either Assembler language or COBOL.

IMS schedules transactions using a load module name that is the same as the PSB name being used for MPP regions or the PGM name for other region types. You must use this load module even though your application program consists of the REXX EXEC. The IMS adapter for REXX provides a load module for you to use. This module is called DFSREXX0. You can use it in one of the following ways:

- Copy to a steplib data set with the same name as the application PSB name. Use either a standard utility intended for copying load modules (such as IEBCOPY or SAS), or the Linkage Editor.
- Use the Linkage Editor to define an alias for DFSREXX0 that is the same as the application PGM name.

Example: Shown below is a section from the PGM setup job. It uses the linkage editor to perform the copy function to the name IVPREXX. The example uses the IVP.

```
//* REXXTDLI SAMPLE - GENERIC APPLICATION DRIVER
//*
//IVPREXX  EXEC PROC=LKED
//L.SYSIN  DD *
  INCLUDE  RESLIB(DFSREXX0)
    ENTRY  DFSREXX0
     NAME  IVPREXX(R)
/*
```

When IMS schedules an application transaction, the load module is loaded and given control. The load module establishes the REXX EXEC name as the PGM name with an argument of the Transaction Code (if applicable). The module calls a user exit routine (DFSREXXU) if it is available. The user exit routine selects the REXX EXEC (or a different EXEC to run) and can change the EXEC arguments, or do any other desired processing. See “Chapter 12. IMS Adapter for REXX Exit Routine” on page 421 for a description of this user exit.

Upon return from the user exit routine, the action requested by the routine is performed. This action normally involves calling the REXX EXEC. The EXEC load occurs using the SYSEXEC DD allocation. This allocation must point to one or more partitioned data sets containing the IMS REXX application programs that will be run as well as any functions written in REXX that are used by the programs.

Standard REXX output, such as SAY statements and tracing, is sent to SYSTSPRT. This DD is required and can be set to SYSOUT=A.

When the stack is empty, the REXX PULL statement reads from the SYSTSIN DD. In this way, you can conveniently provide batch input data to a BMP or batch

region. SYSTSIN is optional; however, you will receive an error message if you issue a PULL from an empty stack and SYSTSIN is not allocated. Figure 61 shows the JCL necessary for MPP region that runs the IVPREXX sample EXEC.

```
//IVP32M11 EXEC PROC=DFSMPR,TIME=(1440),
//         AGN=IVP,          AGN NAME
//         NBA=6,
//         OBA=5,
//         SOUT='*',         SYSOUT CLASS
//         CL1=001,          TRANSACTION CLASS 1
//         CL2=000,          TRANSACTION CLASS 2
//         CL3=000,          TRANSACTION CLASS 3
//         CL4=000,          TRANSACTION CLASS 4
//         TLIM=10,          MPR TERMINATION LIMIT
//         SOD=,             SPIN-OFF DUMP CLASS
//         IMSID=IVP1,       IMSID OF IMS CONTROL REGION
//         PREINIT=DC,       PROCLIB DFSINTXX MEMBER
//         PWFI=Y            PSEUDO=WFI
//*
//* ADDITIONAL DD STATEMENTS
//*
//DFSCTL   DD DISP=SHR,
//            DSN=IVPSYS32.PROCLIB(DFSSBPRM)
//DFSSTAT  DD SYSOUT=*
//* REXX EXEC SOURCE LOCATION
//SYSEXEC  DD DISP=SHR,
//            DSN=IVPIVP32.INSTALIB
//         DD DISP=SHR,
//            DSN=IVPSYS32.DFSEXEC
//* REXX INPUT LOCATION WHEN STACK IS EMPTY
//SYSTSIN  DD *
/*
//* REXX OUTPUT LOCATION
//SYSTSPRT DD SYSOUT=*
//* COBOL OUTPUT LOCATION
//SYSOUT   DD SYSOUT=*
```

Figure 61. JCL Code Used to Run the IVPREXX Sample Exec

IMS Adapter for REXX Overview Diagram

Figure 62 on page 403 shows the IMS adapter for REXX environment at a high level. This figure shows how the environment is structured under the IMS program controller, and some of the paths of interaction between the components of the environment.

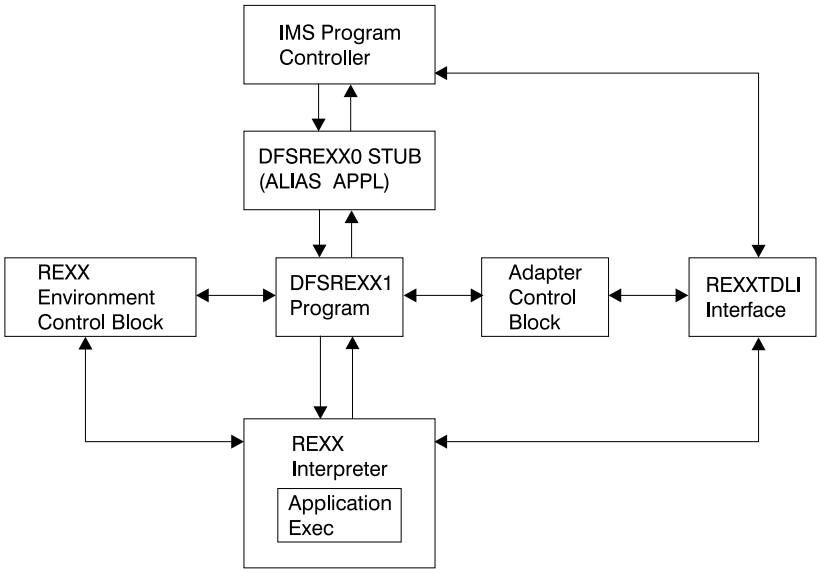


Figure 62. IMS Adapter for REXX Logical Overview Diagram

IVPREXX Sample Application

Figure 61 on page 402 shows the JCL needed to use IVPREXX from an MPP region. This EXEC can also be run from message-driven BMPs or IFP regions.

To use the IVPREXX driver sample program in a message-driven BMP or IFP environment, specify IVPREXX as the program name and PSB name in the IMS region program's parameter list. Specifying IVPREXX loads the IVPREXX load module, which is a copy of the DFSREXX0 front-end program. The IVPREXX program loads and runs an EXEC named IVPREXX that uses message segments sent to the transaction as arguments to derive the EXEC to call or the function to perform.

Interactions with IVPREXX from an IMS terminal are shown in the following examples:

IVPREXX Example 1

Entry:

```
IVPREXX execname
```

or

```
IVPREXX execname arguments
```

Response:

```
EXEC execname ended with RC= x
```

IVPREXX Example 2

Entry:

```
IVPREXX LEAVE
```

Response:

```
Transaction IVPREXX leaving dependent region.
```

IVPREXX Example 3

Entry:

```
IVPREXX HELLOHELLO
```

Response:

```
One-to-eight character EXEC name must be specified.
```

IVPREXX Example 4

Entry:

```
IVPREXX
```

or

```
IVPREXX ?
```

Response:

```
TRANCODE EXECNAME <Arguments> Run specified EXEC
TRANCODE LEAVE                Leave Dependent Region
TRANCODE TRACE level          0=None,1=Some,2=More,3=Full
TRANCODE ROLL                 Issue ROLL call
```

When an EXEC name is supplied, all of the segments it inserts to the I/O PCB are returned before the completion message is returned.

REXX return codes (RC) in the range of 20000 to 20999 are usually syntax or other REXX errors, and you should check the MVS system console or region output for more details.

Related Reading: See *TSO/E Version 2 Procedures Language MVS/REXX Reference* for more information on REXX errors and messages.

Stopping an Infinite Loop: To stop an EXEC that is in an infinite loop, you can enter either of the following IMS commands from the master terminal or system console:

```
/STO REGION p1 ABDUMP p2
/STO REGION p1 CANCEL
```

In these examples, *p1* is the region number and *p2* is the TRANCODE that the EXEC is running under. Use the /DISPLAY ACTIVE command to find the region number. This technique is not specific to REXX EXECs and can be used on any transaction that is caught in an infinite loop.

Related Reading: See *IMS/ESA Operator's Reference* for more information on these commands and others to help in this situation.

REXXTDLI Commands

The following section contains REXX commands and describes how they apply to DL/I calls. The terms *command* and *call* can be used interchangeably when explaining the REXXTDLI environment. However, the term *command* is used exclusively when explaining the REXXIMS environment. For consistency, *call* is used when explaining DL/I, and *command* is used when explaining REXX.

Addressable Environments

To issue commands in the IMS adapter for REXX environment, you must first address the correct environment. Two addressable environments are provided with the IMS adapter for REXX. The environments are as follows:

REXXTDLI Used for standard DL/I calls, for example `GU` and `ISRT`. The REXXTDLI interface environment is used for all standard DL/I calls and cannot be used with REXX-specific commands. All commands issued to this environment are considered to be standard DL/I calls and are processed appropriately. A `GU` call for this environment could look like this:

```
Address REXXTDLI "GU MYPCB DataSeg"
```

REXXIMS Used to access REXX-specific commands (for example, `WTO` and `MAPDEF`) in the IMS adapter for REXX environment. The REXXIMS interface environment is used for both DL/I calls and REXX-specific commands. When a command is issued to this environment, IMS checks to see if it is REXX-specific. If the command is not REXX-specific, IMS checks to see if it is a standard DL/I call. The command is processed appropriately.

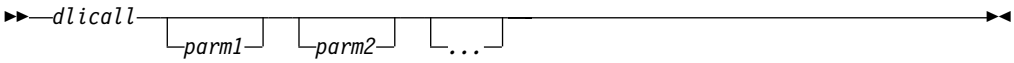
The REXX-specific commands, also called extended commands, are REXX extensions added by the IMS adapter for the REXX interface. A `WTO` call for this environment could look like this:

```
Address REXXIMS "WTO Message"
```

On entry to the scheduled EXEC, the default environment is MVS. Consequently, you must either use ADDRESS REXXTDLI or ADDRESS REXXIMSto issue the IMS adapter for REXX calls.

Related Reading: For general information on addressing environments, see *TSO/E Version 2 Procedures Language MVS/REXX Reference*.

REXXTDLI Calls



The format of a DL/I call varies depending on call type. The parameter formats for supported DL/I calls are shown in previous chapters of this book. The parameters for the calls are case-independent, separated by one or more blanks, and are generally REXX variables. See “Parameter Handling” on page 406 for detailed descriptions.

Return Codes

If you use the AIBTDLI interface, the REXX RC variable is set to the return code from the AIB on the DL/I call.

If you do not use the AIBTDLI interface, a simulated return code is returned. This simulated return code is set to zero if the PCB status code was GA, GK, or `bb`. If the status code had any other value, the simulated return code is X'900' or decimal 2304.

Related Reading: For more information on the AIBTDLI interface, see “Using the AIBTDLI Interface” on page 51“Using the AIBTDLI Interface” on page 93.

Parameter Handling

The IMS adapter for REXX performs some parameter setup for application programs in a REXX environment. This setup occurs when the application program uses variables or maps as the parameters. When the application uses storage tokens, REXX does not perform this setup. The application program must provide the token and parse the results just as a non-REXX application would. For a list of parameter types and definitions, see Table 47.

The REXXTDLI interface performs the following setup:

- The I/O area retrieval for the I/O PCB is parsed. The LL field is removed, and the ZZ field is removed and made available by means of the `REXXIMS('ZZ')` function call. The rest of the data is placed in the specified variable or map. Use the REXX LENGTH() function to find the length of the returned data.
- The I/O area building for the I/O PCB or alternate PCB is done as follows:
  - The appropriate LL field.
  - The ZZ field from a preceding `SET ZZ` command or X'0000' if the command was not used.
  - The data specified in the passed variable or map.
- The I/O area processing for the SPA is similar to the first two items, except that the ZZ field is 4 bytes long.
- The feedback area on the `CHNG` and `SETO` calls is parsed. The LLZZLL fields are removed, and the remaining data is returned with the appropriate length.
- The parameters that have the LLZZ as part of their format receive special treatment. These parameters occur on the `AUTH`, `CHNG`, `INIT`, `ROLS`, `SETO`, and `SETS` calls. The LLZZ fields are removed when IMS returns data to you and added (ZZ is always X'0000') when IMS retrieves data from you. In effect, your application ignores the LLZZ field and works only with the data following it.
- The numeric parameters on `XRST` and symbolic `CHKP` are converted between decimal and a 32-bit number (fullword) as required.

Table 47. IMS Adapter for REXX Parameter Types and Definitions

Type[1]	Parameter Definition
PCB	PCB Identifier specified as a variable containing one of the following: • PCB name as defined in the PSB generation on the PCBNAME= parameter. See *IMS/ESA Utilities Reference: System*for more information on defining PCB names. The name can be from 1 to 8 characters long and does not have to be padded with blanks. If this name is given, the AIBTDLI interface is used, and the return codes and reason codes are acquired from that interface. • An AIB block formatted to DFSAIB specifications. This variable is returned with an updated AIB. • A # followed by PCB offset number (#1=first PCB). Example settings are: – `IOPCB=:"#1"` – `ALTPCB=:"#2"` – `DBPCB=:"#3"` The IOAREA length returned by a database DL/I call defaults to 4096 if this notation is used. The correct length is available only when the AIBTDLI interface is used.

Table 47. IMS Adapter for REXX Parameter Types and Definitions (continued)

Type[1]	Parameter Definition
In	Input variable. It can be specified as a constant, variable, *mapname[2], or !token[3].
SSA	Input variable with an SSA (segment search argument). It can be specified as a constant, variable, *mapname[2], or !token[3].
Out	Output variable to store a result after a successful command. It can be specified as a variable, *mapname[2], or !token[3].
In/Out	Variable that contains input on entry and contains a result after a successful command. It can be specified as a variable, *mapname[2], or !token[3].
Const	Input constant. This command argument must be the actual value, not a variable containing the value.

Note:

1. The parameter types listed above correspond to the types shown (earlier in this book) under the specific DL/I calls, as well as to those shown in Table 48 on page 408.

   All parameters specified on DL/I calls are case independent except for the values associated with the STEM portion of the compound variable (REXX terminology for an array-like structure). A period (.) can be used in place of any parameter and is read as a NULL (zero length string) and written as a void (place holder). Using a period in place of a parameter is useful when you want to skip optional parameters.
2. For more information on *mapname, see “MAPGET” on page 413 and “MAPPUT” on page 413.
3. For more information on !token, see “STORAGE” on page 416.

Example DL/I Calls

The following example shows an `ISRT` call issued against the I/O PCB. It writes the message “Hello World”.

```
IO = "IOPCB"           /* IMS Name for I/O PCB */
OutMsg="Hello World"
Address REXXTDLI "ISRT IO OutMsg"
If RC¬=0 Then Exit 12
```

In this example, *IO* is a variable that contains the PCB name, which is the constant “IOPCB” for the I/O PCB. If a non-zero return code (RC) is received, the EXEC ends (Exit) with a return code of 12. You can do other processing here.

The next example gets a part from the IMS sample parts database. The part number is "250239". The actual part keys have a "02" prefix and the key length defined in the DBD is 17 bytes.

The following example puts the segment into the variable called *Part_Segment*.

```
PartNum = "250239"
DB      = "DBPCB01"
SSA     = 'PARTROOT(PARTKEY = '||Left('02'||PartNum,17)||')'
Address REXXTDLI "GU DB Part_Segment SSA"
```

Notes:

- In a real EXEC, you would probably find the value for PartNum from an argument and would have to check the return code after the call.
- The LEFT function used here is a built-in REXX function. These built-in functions are available to any IMS REXX EXEC. For more information on functions, see *TSO/E Version 2 Procedures Language MVS/REXX Reference*.

- The single quote (') and double quote (") are interchangeable in REXX, as long as they are matched.

The IMS.DFSISRC library includes the DFSSUT04 EXEC. You can use this EXEC to process any unexpected return codes or status codes. To acquire the status code from the last DL/I call issued, you must execute the IMSQUERY('STATUS') function. It returns the two character status code.

Environment Determination

If you use an EXEC that runs in both IMS and non-IMS environments, check to see if the IMS environment is available. You can check to see if the IMS environment is available in two ways:

- Use the MVS SUBCOM command and specify either the REXXTDLI or REXXIMS environments. The code looks like this:

```
Address MVS 'SUBCOM REXXTDLI'
If RC=0 Then Say "IMS Environment is Available."
        Else Say "Sorry, no IMS Environment here."
```

- Use the PARSE SOURCE instruction of REXX to examine the address space name (the 8th word). If it is running in an IMS environment, the token will have the value `IMS`. The code looks like this:

```
Parse Source . . . . . . . Token .
If Token='IMS' Then Say "IMS Environment is Available."
               Else Say "Sorry, no IMS Environment here."
```

REXXIMS Extended Commands

The IMS adapter for REXX gives access to the standard DL/I calls and it supplies a set of extended commands for the REXX environment. These commands are listed in Table 48 and are available when you ADDRESS REXXIMS. DL/I calls are also available when you address the REXXIMS environment.

Table 48 shows the extended commands. The following pages contain detailed descriptions of each command.

Table 48. REXXIMS Extended Commands

Command	Parameter Types [1]
DLIINFO	Out [PCB]
IMSRXTRC	In
MAPDEF	Const In [Const]
MAPGET	Const In
MAPPUT	Const Out
SET	Const In
SRRBACK	Out
SRRCMIT	Out
STORAGE	Const Const [In [Const]]
WTO	In
WTP	In
WTL	In
WTOR	In Out

Table 48. REXXIMS Extended Commands (continued)

Command	Parameter Types [1]

Note:

1. The parameter types listed correspond to the types shown in Table 47 on page 406.

   All parameters specified on DL/I calls are case-independent except for the values associated with the STEM portion of the compound variable (REXX terminology for an array-like structure). A period (.) can be used in place of any parameter and has the effect of a NULL (zero length string) if read and a void (place holder) if written. Use a period in place of a parameter to skip optional parameters.

DLIINFO

The DLIINFO call requests information from the last DL/I call or on a specific PCB.

Format

Call Name	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
DLIINFO	X	X	X	X	X

Usage

The *infoout* variable name is a REXX variable that is assigned the DL/I information. The *pcbid* variable name, when specified as described in "Parameter Handling" on page 406, returns the addresses associated with the specified PCB and its last status code.

The format of the returned information is as follows:

Word	Description
1	Last DL/I call ('.' if N/A)
2	Last DL/I PCB name (name or #number, '.' if N/A)
3	Last DL/I AIB address in hexadecimal (00000000 if N/A)
4	Last DL/I PCB address in hexadecimal (00000000 if N/A)
5	Last DL/I return code (0 if N/A)
6	Last DL/I reason code (0 if N/A)
7	Last DL/I call status ('.' if blank or N/A)

Example

```
Address REXXIMS 'DLIINFO MyInfo'         /* Get Info                  */
Parse Var MyInfo DLI_Cmd DLI_PCB DLI_AIB_Addr DLI_PCB_Addr,
          DLI_RC DLI_Reason DLI_Status .
```

Always code a period after the status code (seventh word returned) when parsing to allow for transparent additions in the future if needed. Words 3, 4, and 7 can be used when a *pcbid* is specified on the DLIINFO call.

IMSRXTRC

The IMSRXTRC command is used primarily for debugging. It controls the tracing action taken (that is, how much trace output via SYSTSPRT is sent to the user) while running a REXX program.

Format

Call Name	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
IMSRXTRC	X	X	X	X	X

Usage

The *level* variable name can be a REXX variable or a digit, and valid values are from 0 to 9. The initial value at EXEC start-up is 1 unless it is overridden by the user Exit. See “Chapter 12. IMS Adapter for REXX Exit Routine” on page 421. Traced output is sent to the DDNAME SYSTSPRT.

The IMSRXTRC command can be used in conjunction with or as a replacement for normal REXX tracing (TRACE).

Level	Description
0	Trace errors only.
1	The previous level and trace DL/I calls, their return codes, and environment status (useful for flow analysis).
2	All the previous levels and variable sets.
3	All the previous levels and variable fetches (useful when diagnosing problems).
4-7	All previous levels.
8	All previous levels and parameter list to/from standard IMS language interface. See message DFS3179 in *IMS/ESA Messages and Codes*.
9	All previous levels.

Example

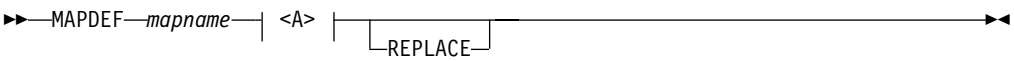
```
Address REXXIMS 'IMSRXTRC 3'
```

IMSRXTRC is independent of the REXX TRACE instruction.

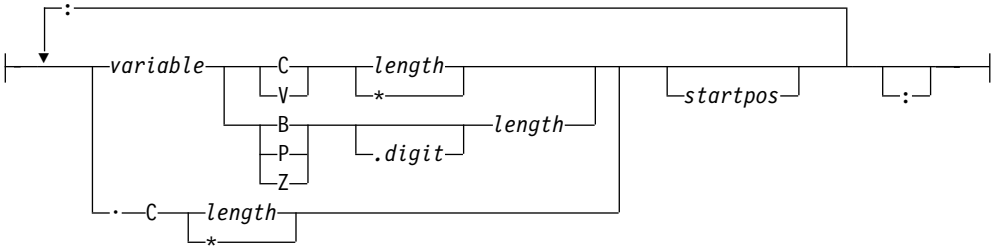
MAPDEF

The MAPDEF command makes a request to define a data mapping.

Format



<A>:



Call Name	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
MAPDEF	X	X	X	X	X

Usage

Data mapping is an enhancement added to the REXXIMS interface. Because REXX does not offer variable structures, parsing the fields from your database segments or MFS output maps can be time consuming, especially when data conversion is necessary. The `MAPDEF`, `MAPGET`, and `MAPPUT` commands allow simple extraction of most formatted data.

- *mapname* is a 1- to 16-character case-independent name.
- `definition` (**<A>**) is a variable containing the map definition.
- `REPLACE`, if specified, indicates that a replacement of an existing map name is allowed. If not specified and the map name is already defined, an error occurs and message DFS3171E is sent to the SYSTPRT.

The map *definition* has a format similar to data declarations in other languages, with simplifications for REXX. In this definition, you must declare all variables that you want to be parsed with their appropriate data types. The format is shown in <A> in the syntax diagram.

Variable name: The variable name *variable* is a REXX variable used to contain the parsed information. Variable names are case-independent. If you use a STEM (REXX terminology for an array-like structure) variable, it is resolved at the time of use (at the explicit or implicit `MAPGET` or `MAPPUT` call time), and this can be very powerful. If you use an index type variable as the STEM portion of a compound variable, you can load many records into an array simply by changing the index variable. Map names or tokens cannot be substituted for variable names inside a map definition.

Repositioning the internal cursor: A period (.) can be used as a variable place holder for repositioning the internal cursor position. In this case, the data type must be C, and the length can be negative, positive, or zero. Use positive values to skip over fields of no interest. Use negative lengths to redefine fields in the middle of a map without using absolute positioning.

The data type values are:

C Character

V Variable

B Binary (numeric)

Z Zoned Decimal (numeric)

P Packed Decimal (numeric)

All numeric data types can have a period and a number next to them. The number indicates the number of digits to the right of a decimal point when converting the number.

Length value: The *length* value can be a number or an asterisk (*), which indicates that the rest of the buffer will be used. You can only specify the *length* value for data types C and V. Data type V maps a 2-byte length field preceding the data string, such that a when the declared length is 2, it takes 4 bytes.

Valid lengths for data types are:

C 1 to 32767 bytes or *

V 1 to 32765 bytes or *

B 1 to 4 bytes

Z 1 to 12 bytes

P 1 to 6 bytes

If a value other than asterisk (*) is given, the cursor position is moved by that value.

The *startpos* value resets the parsing position to a fixed location. If *startpos* is omitted, the column to the right of the previous map variable definition (cursor position) is used. If it is the first variable definition, column 1 is used.

Note: A length of asterisk (*) does not move the cursor position, so a variable declared after one with a length of asterisk (*) without specifying a start column overlays the same definition.

Example

This example defines a map named DBMAP, which is used implicitly on a GU call by placing an asterisk (*) in front of the map name.

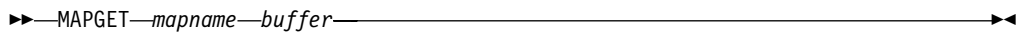
```
DBMapDef = 'RECORD         C    * :',  /* Pick up entire record     */
           'NAME           C   10 :',  /* Cols 1-10 hold the name   */
           'PRICE          Z.2  6 :',  /* Cols 11-16 hold the price */
           'CODE           C    2 :',  /* Cols 11-16 hold the code  */
           '.              C   25 :',  /* Skip 25 columns           */
           'CATEGORY       B    1'     /* Col 42 holds category     */
Address REXXIMS 'MAPDEF DBMAP DBMapDef'

:
:
Address REXXTDLI 'GU DBPCB *DBMAP' /* Read and decode a segment      */
If RC¬=0 Then Signal BadCall       /* Check for failure              */
Say CODE                           /* Can now access any Map Variable*/
```

The entire segment retrieved on the GU call is placed in RECORD. The first 10 characters are placed in NAME, and the next 6 are converted from zoned decimal to EBCDIC with two digits to the right of the decimal place and placed in PRICE. The next 2 characters are placed in CODE, the next 25 are skipped, and the next character is converted from binary to EBCDIC and placed in CATEGORY. The 25 characters that are skipped are present in the RECORD variable.

MAPGET

The MAPGET command is a request to parse or convert a buffer into a specified data mapping previously defined with the MAPDEF command.

Format



Call Name	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
MAPGET	X	X	X	X	X

Usage

The *mapname* variable name specifies the data mapping to use. It is a 1- to 16-character case-independent name. The *buffer* variable name is the REXX variable containing the data to parse.

Map names can also be specified in the REXXTDLI calls in place of variable names to be set or written. This step is called an implicit MAPGET. Thus, the explicit (or variable dependent) MAPGET call can be avoided. To indicate that a Map name is being passed in place of a variable in the DL/I call, precede the name with an asterisk (*), for example, 'GU IOPCB *INMAP'.

Examples

This example uses explicit support.

```
Address REXXTDLI 'GU DBPCB SegVar'
If RC=0 Then Signal BadCall          /* Check for failure              */
Address REXXIMS 'MAPGET DBMAP SegVar'/* Decode Segment                 */
Say VAR_CODE                         /*Can now access any Map Variable */
```

This example uses implicit support.

```
Address REXXTDLI 'GU DBPCB *DBMAP'   /* Read and decode segment if read*/
If RC=0 Then Signal BadCall          /* Check for failure              */
Say VAR_CODE                         /* Can now access any Map Variable*/
```

If an error occurs during a MAPGET, message DFS3172I is issued. An error could occur when a Map is defined that is larger than the input segment to be decoded or during a data conversion error from packed or zoned decimal format. The program continues, and an explicit MAPGET receives a return code 4. However, an implicit MAPGET (on a REXXTDLI call, for example) does not have its return code affected. Either way, the failing variable's value is dropped by REXX.

MAPPUT

This MAPPUT command makes a request to pack or concatenate variables from a specified Data Mapping, defined by the MAPDEF command, into a single variable.

Format

Call Name	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
MAPPUT	X	X	X	X	X

Usage

The *mapname* variable name specifies the data mapping to use, a 1- to 16-character case-independent name. The *buffer* variable name is the REXX variable that will contain the resulting value.

Map names can also be specified in the REXXTDLI call in place of variable names to be fetched or read. This step is called an implicit MAPPUT and lets you avoid the explicit MAPPUT call. To indicate that a Map name is being passed in the DL/I call, precede the name with an asterisk (*), for example, 'ISRT IOPCB *OUTMAP'.

Note: If the data mapping is only partial and some fields in the record are not mapped to REXX variables, then the first field in the mapping should be a character type of length asterisk (*), as shown in the “Example” on page 412. This step is the only way to ensure that non-mapped (skipped) fields are not lost between the MAPGET and MAPPUT calls, whether they be explicit or implicit.

Examples

This example uses explicit support.

```
Address REXXTDLI
'GHU DBPCB SegVar SSA1'                 /* Read segment              */
If RC¬=0 Then Signal BadCall            /* Check for failure         */
Address REXXIMS 'MAPGET DBMAP SegVar'   /* Decode Segment            */
DBM_Total = DBM_Total + Deposit_Amount  /* Adjust Mapped Variable    */
Address REXXIMS 'MAPPUT DBMAP SegVar'   /* Encode Segment            */
'REPL DBPCB SegVar'                     /* Update Database           */
If RC¬=0 Then Signal BadCall            /* Check for failure         */
```

This example uses implicit support.

```
Address REXXTDLI
'GHU DBPCB *DBMAP SSA1'                 /* Read and decode segment if read */
If RC¬=0 Then Signal BadCall            /* Check for failure               */
DBM_Total = DBM_Total + Deposit_Amount  /* Adjust Mapped Variable          */
'REPL DBPCB *DBMAP'                     /* Update Database                 */
If RC¬=0 Then Signal BadCall            /* Check for failure               */
```

If an error occurs during a MAPPUT, such as a Map field defined larger than the variable’s contents, then the field is truncated. If the variable’s contents are shorter than the field, the variable is padded:

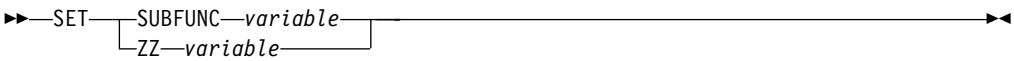
Character (C)	Padded on right with blanks
Character (V)	Padded on right with zeros
Numeric (B,Z,P)	Padded on the left with zeros

If a MAP variable does not exist when a MAPPUT is processed, the variable and its position are skipped. All undefined and skipped fields default to binary zeros. A null parameter is parsed normally. Conversion of non-numeric or null fields to numeric field results in a value of 0 being used and no error.

SET

The SET command resets AIB subfunction values and ZZ values before you issue a DL/I call.

Format



Call Name	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
SET	X	X	X	X	X

Usage

The SET SUBFUNC command sets the AIB subfunction used on the next DL/I call. This value is used only if the next REXXTDLI call passes a PCB name. If the call does pass a PCB name, the IMS adapter for REXX places the subfunction name (1 to 8 characters or blank) in the AIB before the call is issued. This value initially defaults to blanks and is reset to blanks on completion of any REXXTDLI DL/I call. For more information on subfunctions, see the appropriate chapters in this book.

The SET ZZ command is used to set the ZZ value used on a subsequent DL/I call. This command is most commonly used in IMS conversational transactions and terminal dependent applications to set the ZZ field to something other than the default of binary zeros. Use the SET command before an ISRT call that requires other than the default ZZ value. For more explanation on ZZ processing, see “Parameter Handling” on page 406.

Examples

This example shows the SET SUBFUNC command used with the INQY call to get environment information.

```
IO="IOPCB"
Func = "ENVIRON"                       /* Sub-Function Value */
Address REXXIMS  "SET SUBFUNC Func"    /* Set the value      */
Address REXXTDLI "INQY IO EnviData"    /* Make the DL/I Call */
IMS_Identifier = Substr(EnviData,1,8)  /* Get IMS System Name*/
```

This example shows the SET ZZ command used with a conversational transaction for SPA processing.

```
Address REXXTDLI 'GU IOPCB SPA'           /* Get first Segment         */
Hold_ZZ = IMSQUERY('ZZ')                  /* Get ZZ Field (4 bytes)    */

.
.
.
Address REXXIMS  'SET ZZ Hold_ZZ'        /* Set ZZ for SPA ISRT       */
Address REXXTDLI 'ISRT IOPCB SPA'        /* ISRT the SPA              */
```

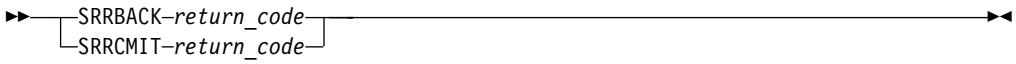
This example shows the SET ZZ command used for setting 3270 Device Characteristics Flags.

```
Bell_ZZ = '0040'X                        /* ZZ to Ring Bell on Term   */
Address REXXIMS  'SET ZZ Bell_ZZ'        /* Set ZZ for SPA ISRT       */
Address REXXTDLI 'ISRT IOPCB Msg'        /* ISRT the Message          */
```

SRRBACK and SRRCMIT

The Common Programming Interface Resource Recovery (CPI-RR) commands allow an interface to use the SAA resource recovery interface facilities for back-out and commit processing.

Format



Call Name	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
SRRBACK, SRRCMIT	X		X		

Usage

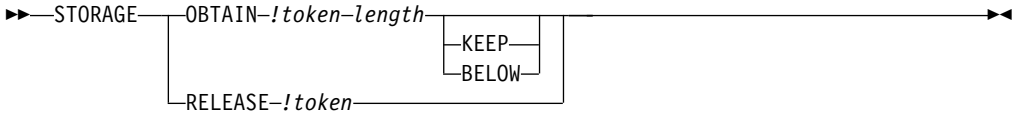
The return code from the SRR command is returned and placed in the *return_code* variable name as well as the REXX variable RC.

For more information on SRRBACK and SRRCMIT, see *IMS/ESA Administration Guide: Transaction Manager*and *System Application Architecture Common Programming Interface: Resource Recovery Reference*.

STORAGE

The STORAGE command allows the acquisition of system storage that can be used in place of variables for parameters to REXXTDLI and REXXIMS calls.

Format



Call Name	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
STORAGE	X	X	X	X	X

Usage

Although REXX allows variables to start with characters (*!*) and (#), these characters have special meanings on some commands. When using the REXXTDLI interface, you must not use these characters as the starting characters of variables.

The *!token* variable name identifies the storage, and it consists of an exclamation mark followed by a 1- to 16-character case-independent token name. The *length* variable name is a number or variable containing size in decimal to OBTAIN in the range 4 to 16777216 bytes (16 MB). The storage class has two possible override values, BELOW and KEEP, of which only one can be specified for any particular token. The BELOW function acquires the private storage below the 16 MB line. The KEEP function marks the token to be kept after this EXEC is terminated. The default action gets the storage in any location and frees the token when the EXEC is terminated.

Use the STORAGE command to get storage to use on DL/I calls when the I/O area must remain in a fixed location (for example, Spool API) or when it is not desirable to have the LLZZ processing. For more information on LLZZ processing, see “Parameter Handling” on page 406. Once a token is allocated, you can use it in REXXTDLI DL/I calls or on the STORAGE RELEASE command.

Note the following when using STORAGE:

- When used on DL/I calls, none of the setup for LLZZ fields takes place. You must fill the token in and parse the results from it just as required by a non-REXX application.
- You cannot specify both KEEP and BELOW on a single STORAGE command.
- The RELEASE function is only necessary for tokens marked KEEP. All tokens not marked KEEP and not explicitly released by the time the EXEC ends are released automatically by the IMS adapter for REXX.
- When you use OBTAIN, the entire storage block is initialized to 0.
- The starting address of the storage received is always on the boundary of a double word.
- You cannot re-obtain a token until RELEASE is used or the EXEC that obtained it, non-KEEP, terminates. If you try, a return code of -9 is given and the error message DFS3169 is issued.
- When KEEP is specified for the storage token, it can be accessed again when this EXEC or another EXEC knowing the token's name is started in the same IMS region.
- Tokens marked KEEP are not retained when an ABEND occurs or some other incident occurs that causes region storage to be cleared. It is simple to check if the block exists on entry with the IMSQUERY(!token) function. For more information, see "IMSQUERY Extended Functions" on page 418.

Example

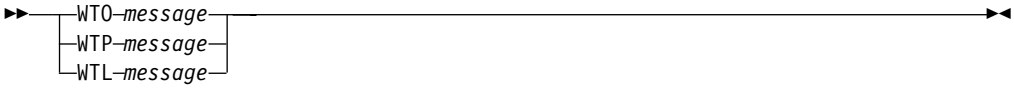
This example shows how to use the STORAGE command with Spool API.

```
/* Get 4K Buffer below the line for Spool API Usage */
Address REXXIMS 'STORAGE OBTAIN !MYTOKEN 4096 BELOW'
/* Get Address and length (if curious) */
Parse Value IMSQUERY('!MYTOKEN') With My_Token_Addr My_Token_Len.
Address REXXIMS 'SETO ALTPCB !MYTOKEN SETOPARMS SETOFB'

.
.
.
Address REXXIMS 'STORAGE RELEASE !MYTOKEN'
```

WTO, WTP, and WTL

The WTO command is used to write a message to the operator. The WTP command is used to write a message to the program (WTO ROUTCDE=11). The WTL command is used to write a message to the console log.

Format



Call Name	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
WTO, WTP, WTL	X	X	X	X	X

Usage

The *message* variable name is a REXX variable containing the text that is stored displayed in the appropriate place.

Example

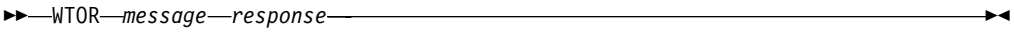
This example shows how to write a simple message stored the REXX variable MSG.

```
Msg = 'Sample output message.'          /* Build Message            */
Address REXXIMS 'WTO Msg'               /* Tell Operator            */
Address REXXIMS 'WTP Msg'               /* Tell Programmer          */
Address REXXIMS 'WTL Msg'               /* Log It                   */
```

WTOR

The WTOR command requests input or response from the MVS system operator.

Format



Call Name	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
WTOR	X	X	X	X	X

Usage

The *message* variable name is a REXX variable containing the text that will be displayed on the MVS console. The operator's response is placed in the REXX variable signified by the *response* variable name.

Attention: This command hangs the IMS region in which it is running until the operator responds.

Example

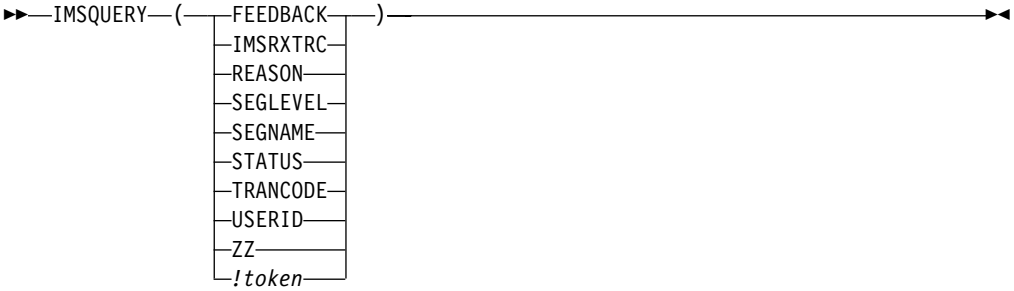
This example prompts the operator to enter ROLL or CONT on the MVS master or alternate console. Once the WTOR is answered, the response is placed in the REXX variable name *response*, and the EXEC will continue and process the IF statement appropriately.

```
Msg = 'Should I ROLL or Continue. Reply "ROLL" or "CONT"'
Address REXXIMS 'WTOR Msg Resp'         /* Ask Operator     */
If Resp = 'ROLL' Then                   /* Tell Programmer  */
   Address REXXTDLI 'ROLL'              /* Roll Out of this */
```

IMSQUERY Extended Functions

The IMSQUERY function is available to query certain IMS information either on the environment or on the prior DL/I call.

Format



Call Name	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
IMSQUERY	X	X	X	X	X

Usage

The format of the function call is: `IMSQUERY('Argument')` where Argument is one of the following values:

Argument	Description of Data Returned
FEEDBACK	FEEDBACK area from current PCB.
IMSRXTRC	Current `IMSRXTRC` trace level #.
REASON	Reason code from last call (from AIB if used on last REXXTDLI type call).
SEGLEVEL	Segment level from current PCB (Last REXXTDLI call must be against a DB PCB, or null is returned).
SEGNAME	Segment name from current PCB (Last REXXTDLI call must be against a DB PCB, or null is returned).
STATUS	IMS status code from last executed REXXTDLIcall (DL/I call). This argument is the two character status code from the PCB.
TRANCODE	Current transaction code being processed, if available.
USERID	Input terminal's user ID, if available. If running in a non-message-driven region, the value is dependent on the specification of the BMPUSID= keyword in the DFSDCxxx PROCLIB member: • If BMPUSID=USERID is specified, the value from the USER= keyword on the JOB statement is used. • If USER= is not specified on the JOB statement, the program's PSB name is used. • If BMPUSID=PSBNAME is specified, or if BMPUSID= is not specified at all, the program's PSB name is used.
ZZ	ZZ (of LLZZ) from last REXXTDLI command. This argument can be used to save the ZZ value after you issue a `GU` call to the I/O PCB when the transaction is conversational.
!token	Address (in hexadecimal) and length of specified token (in decimal), separated by a blank.

This value can be placed in a variable or resolved from an expression. In these cases, the quotation marks should be omitted as shown below:

```
Token_Name="!MY_TOKEN"
AddrInfo=IMSQUERY(Token_Name)
        /*  or  */
AddrInfo=IMSQUERY("!MY_TOKEN")
```

Although the function argument is case-independent, no blanks are allowed within the function argument. You can use the REXX STRIP function on the argument, if necessary. IMSQUERY is the preferred syntax, however REXXIMS is supported and can be used, as well.

Example

```
If REXXIMS('STATUS')='GB' Then Signal End_Of_DB
:
Hold_ZZ = IMSQUERY('ZZ')        /* Get current ZZ field*/
:
Parse Value IMSQUERY('!MYTOKEN') With My_Token_Addr My_Token_Len .
```

Chapter 12. IMS Adapter for REXX Exit Routine

You can use a user exit routine (DFSREXXU) with the IMS adapter for the REXX environment. It is optional and can be omitted from the link-edit step. The user exit routine is used more for an installation than for a specific exec. The user exit routine is provided by the IMS adapter for REXX and is called only when a new REXX transaction is scheduled and ends. The user exit is not associated with the standard REXX exits provided by TSO. A sample user exit routine (DFSREXXU) is shipped with IMS (in source code only). For the latest version of the DFSREXXU source code , see the IMS.SVSOURSE distribution library; member name is DFSREXXU.

The exit routine gets control before the environment is built, just before an exec is executed, and just after it ends. The routine has the ability to do the following:

- Override the exec name to be executed. This name defaults to the IMS program name.
- Choose not to execute any exec and have the IMS adapter for REXX return to IMS.

  It is the exit routine's responsibility to do any required processing such as issuing a GU (Get-Unique) call if the region type is MPP.
- Issue DL/I calls using the AIB interface as part of its logic in determining what exec to execute.
- Set REXX variables (through IRXEXCOM) before the exec is started. The variables are then available to the exec.
- Extract REXX variables (through IRXEXCOM) after the exec ends. These variables were set earlier by the exec or exit routine.
- Change the initial default IMSRXTRC tracing level.

The user exit routine must conform to all of the restrictions that apply to IMS application programs.

In this Chapter:

- "Environment"
- "Parameters"

Environment

The user exit routine must be named DFSREXXU and be link-edited with DFSREXX1 during installation of the IMS adapter for REXX. The routine must be written to be reentrant (RENT), AMODE 31, RMODE ANY.

Parameters

Entry parameters are:

R0 Pointer to REXX Environment Block as described in *TSO/E Version 2 Procedures Language MVS/REXX Reference*

R1 Pointer to parameter list

R13 Pointer to save area

R14 Return address

R15 Entry point address

© Copyright IBM Corp. 1974, 2000

Parameters

Upon exit, all registers except R15 must be restored. Only the parameters can be altered. The content of R15 is ignored on exit.

The parameter list contains a list of pointers to the parameters. All character data is left justified and padded with blanks, if necessary. Omitted fields are set to blanks. All fields are read-only unless otherwise specified. Table 49 on page 422 shows the user exit parameter list format.

Table 49. User Exit Parameter List

Name	Offset (Dec)	Data Type	Length (Dec)	Description
Function	0	Pointer	4	Pointer to one word function type. Func=0 on Setup Call, Func=1 on Entry Call, Func=2 on Exit Call.
EXECParm[1]	4	Pointer	4	Pointer to 128-byte area containing parameters that are passed to the REXX interpreter. The format of the area is a halfword length field contains the length of the text string that follows. The first blank separated word or the entire string if no blanks are present is the exec name to execute. On entry this field is set to the program name followed by one blank and the transaction code if available. The exit can rebuild this field when called on entry to alter the exec name and/or parameters that are passed. The length field can be set to zero indicating no exec is to be executed.
PgmName[1]	8	Pointer	4	Pointer to 8-byte area containing the Program name that was scheduled.
TranCode[1]	12	Pointer	4	Pointer to 8-byte area containing the Transaction Code that was scheduled, if available (MPP,BMP,IFP).
User_ID[1]	16	Pointer	4	Pointer to 8-byte area containing the current user ID for the scheduled program, if available (MPP,BMP,IFP).
IMSRXTRC	20	Pointer	4	Pointer to one word IMSRXTRC level. This value defaults to 1 at exec start-up but can be overridden by the user exit. See “IMSRXTRC” on page 410 for more information on values. Note that the level field here is a FULLWORD and not EBCDIC.

Table 49. User Exit Parameter List (continued)

Name	Offset (Dec)	Data Type	Length (Dec)	Description
UserArea	24	Pointer	4	Pointer to 8-byte (word aligned) user area that is passed on entry and is preserved verbatim on exit. This field is set to binary zeros whenever the REXX environment is built in the dependent region. The user area can be altered by the user exit and is provided as an anchor.
RetCode	28	Pointer	4	Pointer to one word return code. The return code must be set to zero.

Note:

1. When on a Setup call the next four parameters are not available; their addresses are 0.

For each user exit parameter described in Table 49 on page 422, Figure 63 shows the corresponding DFSREXXU parameter.

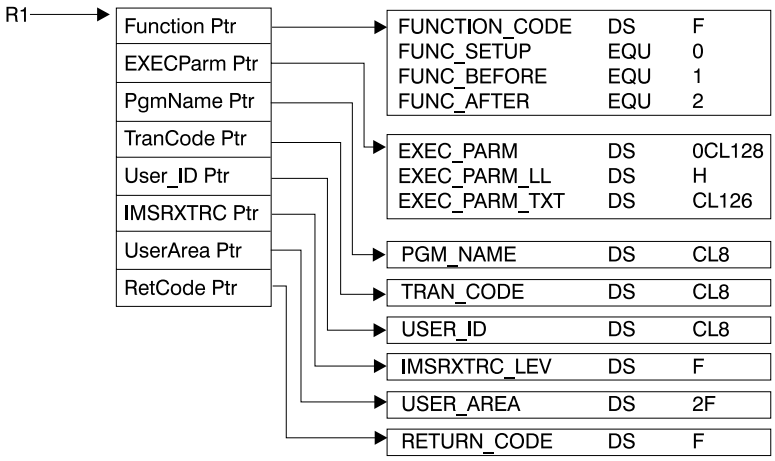


Figure 63. DFSREXXU Parameter List

Parameters

Chapter 13. Sample Execs Using REXXTDLI

This chapter shows samples of REXX execs that use REXXTDLI to access IMS services.

The example sets are designed to highlight various features of writing IMS applications in REXX. The samples in this chapter are simplified and might not reflect actual usage (for example, they do not use databases).

The PART exec database access example is a set of three execs that access the PART database, which is built by the IMS installation verification program (IVP). The first two execs in this example, PARTNUM and PARTNAME, are extensions of the PART transaction that runs the program DFSSAM02, which is supplied with IMS as part of IVP. The third exec is the DFSSAM01 exec supplied with IMS and is an example of the use of EXECIO within an exec.

In this Chapter:

- "SAY Exec: For Expression Evaluation"
- "PCBINFO Exec: Display PCBs Available in Current PSB" on page 426
- "PART Execs: Database Access Example" on page 429
- "DOCMD: IMS Commands Front End" on page 432
- "IVPREXX: MPP/IFP Front End for General Exec Execution" on page 436

SAY Exec: For Expression Evaluation

Figure 64 is a listing of the SAY exec. SAY evaluates an expression supplied as an argument and displays the results. The REXX command INTERPRET is used to evaluate the supplied expression and assign it to a variable. Then that variable is used in a formatted reply message.

```
/* EXEC TO DO CALCULATIONS */
Address REXXTDLI
Arg Args
If Args='' Then
   Msg='SUPPLY EXPRESSION AFTER EXEC NAME.'
Else Do
   Interpret 'X='Args           /* Evaluate Expression */
   Msg='EXPRESSION:' Args '=' X
End
'ISRT IOPCB MSG'
Exit RC
```

Figure 64. Exec To Do Calculations

This exec shows an example of developing applications with IMS Adapter for REXX. It also shows the advantages of REXX, such as dynamic interpretation, which is the ability to evaluate a mathematical expression at run-time.

A PDF EDIT session is shown in Figure 65 on page 426. This figure shows how you can enter a new exec to be executed under IMS.

© Copyright IBM Corp. 1974, 2000

```
EDIT ---- USER.PRIVATE.PROCLIB(SAY) - 01.03 ------------------ COLUMNS 001 072
COMMAND ===>                                                  SCROLL ===> PAGE
****** ***************************** TOP OF DATA ******************************
000001 /* EXEC TO DO CALCULATIONS */
000002 Address REXXTDLI
000003 Arg Args
000004 If Args='' Then
000005    Msg='SUPPLY EXPRESSION AFTER EXEC NAME.'
000006 Else Do
000007    Interpret 'X='Args           /* Evaluate Expression */
000008    Msg='EXPRESSION:' Args '=' X
000009 End
000010
000011 'ISRT IOPCB MSG'
000012 Exit RC
****** **************************** BOTTOM OF DATA ****************************
```

Figure 65. PDF EDIT Session on the SAY Exec

To execute the SAY exec, use IVPREXX and supply an expression such as:

```
IVPREXX SAY 5*5+7
```

This expression produces the output shown in Figure 66.

```
EXPRESSION: 5*5+7 = 32
EXEC SAY ended with RC= 0
```

Figure 66. Example Output from the SAY Exec

PCBINFO Exec: Display PCBs Available in Current PSB

The PCB exec maps the PCBs available to the exec, which are the PCBs for the executing PSB. The mapping consists of displaying the type of PCB (IO, TP, or DB), the LTERM or DBD name that is associated, and other useful information. Mapping displays this information by using the PCB function described in “DLIINFO” on page 409. Example output screens are shown in Figure 67 and Figure 68 on page 427. The listing is shown in Figure 69 on page 428. PCB mappings are created by placing DFSREXX0 in an early concatenation library and renaming it to an existing application with a PSB/DBD generation.

```
IMS PCB System Information Exec: PCBINFO
System Date: 09/26/92    Time: 15:52:15

PCB # 1: Type=IO, LTERM=T3270LC  Status=   UserID=         OutDesc=DFSMO2
         Date=91269 Time=1552155
PCB # 2: Type=TP, LTERM=* NONE * Status=AD
PCB # 3: Type=TP, LTERM=* NONE * Status=
PCB # 4: Type=TP, LTERM=CTRL     Status=
PCB # 5: Type=TP, LTERM=T3275    Status=
EXEC PCBINFO ended with RC= 0
```

Figure 67. Example Output of PCBINFO Exec on a PSB without Database PCBs.

```
IMS PCB System Information Exec: PCBINFO
System Date: 09/26/92    Time: 15:53:34

PCB # 1: Type=IO, LTERM=T3270LC  Status=   UserID=         OutDesc=DFSMO2
         Date=89320 Time=1553243
PCB # 2: Type=DB, DBD  =DI21PART Status=   Level=00 Opt=G
EXEC PCBINFO ended with RC= 0
```

Figure 68. Example Output of PCBINFO Exec on a PSB with a Database PCB.

PCBINFO Exec

```
/* REXX EXEC TO SHOW SYSTEM LEVEL INFO */
Address REXXTDLI
Arg Dest .
WTO=(Dest='WTO')
Call SayIt 'IMS PCB System Information Exec: PCBINFO'
Call SayIt 'System Date:' Date('U') '   Time:' Time()
Call Sayit ' '
/* A DFS3162 message is given when this exec is run because it does */
/* not know how many PCBs are in the list and it runs until it gets */
/* an error return code.  Note this does not show PCBs that are     */
/* available to the PSB by name only, i.e. not in the PCB list.     */
Msg='PCBINFO: Error message normal on DLIINFO.'
'WTP MSG'
Do i=1 by 1 until Result='LAST'
   Call SayPCB i
End
Exit 0


SayPCB: Procedure Expose WTO
   Arg PCB
   'DLIINFO DLIINFO #'PCB    /* Get PCB Address */
   If rc<0 Then Return 'LAST'   /* Invalid PCB Number */
   Parse Var DLIInfo  . . AIBAddr PCBAddr .
   PCBINFO=Storage(PCBAddr,255) /* Read PCB */
   DCPCB=(Substr(PCBInfo,13,1)='00'x) /* Date Field, must be DC PCB */
   If DCPCB then Do
      Parse Value PCBInfo with,
         LTERM 9 . 11 StatCode 13 CurrDate 17 CurrTime 21,
         InputSeq 25 OutDesc 33 UserID 41
      If LTERM='' then LTERM='* NONE *'
      CurrDate=Substr(c2x(CurrDate),3,5)
      CurrTime=Substr(c2x(CurrTime),1,7)
      If CurrDate¬='000000' then Do
         Call SayIt 'PCB #'Right(PCB,2)': Type=IO, LTERM='LTERM,
            'Status='StatCode 'UserID='UserID 'OutDesc='OutDesc
         Call SayIt '         Date='CurrDate 'Time='CurrTime
      End
      Else
         Call SayIt 'PCB #'Right(PCB,2)': Type=TP, LTERM='LTERM,
            'Status='StatCode
   End
   Else Do
      Parse Value PCBInfo with,
         DBDName 9 SEGLev 11 StatCode 13 ProcOpt 17 . 21 Segname . 29,
         KeyLen 33 NumSens 37
      KeyLen = c2d(KeyLen)
      NumSens= c2d(NumSens)

      Call SayIt 'PCB #'Right(PCB,2)': Type=DB, DBD  ='DBDName,
               'Status='StatCode 'Level='SegLev 'Opt='ProcOpt
   End
Return '

SayIt: Procedure Expose WTO
   Parse Arg Msg
   If WTO Then
      'WTO MSG'
   Else
      'ISRT IOPCB MSG'
Return
```

Figure 69. PCBINFO Exec Listing

PART Execs: Database Access Example

This set of execs accesses the PART database shipped with IMS. These execs demonstrate fixed-record database reading, SSAs, and many REXX functions. The PART database execs (PARTNUM, PARTNAME, and DFSSAM01) are described in this section.

The PARTNUM exec is used to show part numbers that begin with a number equal to or greater than the number you specify. An example output screen is shown in Figure 70.

To list part numbers beginning with the number “300” or greater, enter the command:

```
PARTNUM 300
```

All part numbers that begin with a 300 or larger numbers are listed. The listing is shown in Figure 72 on page 430.

```
IMS Parts DATABASE Transaction
System Date: 02/16/92    Time: 23:28:41

Request: Display 5 Parts with Part_Number >= 300
 1 Part=3003802          Desc=CHASSIS
 2 Part=3003806          Desc=SWITCH
 3 Part=3007228          Desc=HOUSING
 4 Part=3008027          Desc=CARD FRONT
 5 Part=3009228          Desc=CAPACITOR

EXEC PARTNUM ended with RC= 0
```

Figure 70. Example Output of PARTNUM Exec

PARTNAME is used to show part names that begin with a specific string of characters.

To list part names beginning with “TRAN”, enter the command:

```
PARTNAME TRAN
```

All part names that begin with “TRAN” are listed on the screen. The screen is shown in Figure 71. The listing is shown in Figure 73 on page 431.

```
IMS Parts DATABASE Transaction
System Date: 02/16/92    Time: 23:30:09

Request: Display 5 Parts with Part Name like TRAN
 1 Part=250239           Desc=TRANSISTOR
 2 Part=7736847P001      Desc=TRANSFORMER
 3 Part=975105-001       Desc=TRANSFORMER
 4 Part=989036-001       Desc=TRANSFORMER
End of DataBase reached before 5 records shown.

EXEC PARTNAME ended with RC= 0
```

Figure 71. Example Output of PARTNAME Exec

PART Execs

The DFSSAM01 exec is used to load the parts database. This exec is executed in batch, is part of the IVP, and provides an example of EXECIO usage in an exec. For details, see *IMS/ESA Installation Volume 1: Installation and Verification*.

PARTNUM Exec: Show Set of Parts Near a Specified Number

Requirement: The following REXX exec is designed to be run by the IVPREXX exec with PSB=DFSSAM02.

```
/* REXX EXEC TO SHOW A SET OF PARTS NEAR A SPECIFIED NUMBER    */
/* Designed to be run by the IVPREXX exec with PSB=DFSSAM02    */
/* Syntax:    IVPREXX PARTNUM string <start#>                  */

Address REXXTDLI
IOPCB='IOPCB'      /* PCB Name */
DataBase='#2'      /* PCB # */
RootSeg_Map = 'PNUM C 15 3 : DESCRIPTION C 20 27'
'MAPDEF ROOTSEG ROOTSEG_MAP'
Call SayIt 'IMS Parts DATABASE Transaction'
Call SayIt 'System Date:' Date('U') '   Time:' Time()
Call Sayit ' '

Arg  PartNum Segs .
If ¬DataType(Segs,'W') then Segs=5  /* default view amount */

PartNum=Left(PartNum,15)    /* Pad to 15 with Blanks */
If PartNum='' then
  Call Sayit 'Request: Display first' Segs 'Parts in the DataBase'
Else
  Call Sayit 'Request: Display' Segs 'Parts with Part_Number >=' PartNum
SSA1='PARTROOT(PARTKEY >=02'PartNum')'
'GU DATABASE *ROOTSEG SSA1'
Status=IMSQUERY('STATUS')
If Status='GE' then Do /* Segment Not Found */
   Call Sayit 'No parts found with larger Part_Number'
   Exit 0
End
Do i=1 to Segs While Status='  '
   Call Sayit Right(i,2) 'Part='PNum ' Desc='Description
   'GN DATABASE *ROOTSEG SSA1'
   Status=IMSQUERY('STATUS')
End
If Status='GB' then
   Call SayIt 'End of DataBase reached before' Segs 'records shown.'
Else If Status¬='  ' then Signal BadCall
Call Sayit ' '
   Exit 0

SayIt: Procedure  Expose IOPCB
   Parse Arg Msg
   'ISRT IOPCB MSG'
   If RC¬=0 then Signal BadCall
Return

BadCall:
   'DLIINFO INFO'
   Parse Var Info Call PCB . . . . Status .
   Msg = 'Unresolved Status Code' Status,
      'on' Call 'on PCB' PCB
   'ISRT IOPCB MSG'
Exit 99
```

Figure 72. PARTNUM Exec: Show Set of Parts Near a Specified Number

PARTNAME Exec: Show a Set of Parts with a Similar Name

Requirement: The following REXX exec is designed to be run by the IVPREXX exec with PSB=DFSSAM02.

```
/* REXX EXEC TO SHOW ALL PARTS WITH A NAME CONTAINING A STRING  */
/* Designed to be run by the IVPREXX exec with PSB=DFSSAM02     */
/* Syntax:   IVPREXX PARTNAME string <#parts>                   */

Arg  PartName Segs .
Address REXXIMS
Term    ='IOPCB'      /* PCB Name */
DataBase='DBPCB01'    /* PCB Name for Parts Database */

Call SayIt 'IMS Parts DATABASE Transaction'
Call SayIt 'System Date:' Date('U') '   Time:' Time()
Call Sayit ' '

If ¬DataType(Segs,'W') & Segs¬='*' then Segs=5
If PartName='' then Do
   Call Sayit 'Please supply the first few characters of the part name'
   Exit 0
End

Call Sayit 'Request: Display' Segs 'Parts with Part Name like' PartName
SSA1='PARTROOT '
'GU DATABASE ROOT_SEG SSA1'
Status=REXXIMS('STATUS')
i=0
Do While RC=0 & (i<Segs | Segs='*')
   Parse Var Root_Seg 3 PNum 18 27 Description 47
   'GN DATABASE ROOT_SEG SSA1'
   Status=REXXIMS('STATUS')
   If RC¬=0 & Status¬='GB' Then Leave
   If Index(Description,PartName)=0 then Iterate
   i=i+1
   Call Sayit Right(i,2)') Part='PNum ' Desc='Description
End
If RC¬=0 & Status¬='GB' Then Signal BadCall
If i<Segs & Segs¬='*' then
   Call SayIt 'End of DataBase reached before' Segs 'records shown.'
Call Sayit ' '
Exit 0

SayIt: Procedure Expose Term
   Parse Arg Msg
   'ISRT Term MSG'
   If RC¬=0 then Signal BadCall
Return

BadCall:
   Call "DFSSUT04" Term
Exit 99
```

Figure 73. PARTNAME Exec: Show Parts with Similar Names

DFSSAM01 Exec: Load the Parts Database

For the latest version of the DFSSAM01 source code, see the IMS.DFSEXECA distribution library; member name is DFSSAM01.

DOCMD: IMS Commands Front End

DOCMD is an automatic operator interface (AOI) transaction program that issues IMS commands and allows dynamic filtering of their output. The term “dynamic” means that you use the headers for the command as the selectors (variable names) in the filter expression (Boolean expression resulting in 1 if line is to be displayed and 0 if it is not). This listing is shown in Figure 80 on page 434.

Not all commands are allowed through transaction AOI, and some setup needs to be done to use this AOI.

Related Reading: See “Security Considerations for Automated Operator Commands” in *IMS/ESA Administration Guide: System* for more information.

Some examples of DOCMD are given in Figure 74, Figure 75, Figure 76, Figure 77 on page 433, Figure 78 on page 433, and Figure 79 on page 434.

```
Please supply an IMS Command to execute.
EXEC DOCMD ended with RC= 0
```

Figure 74. Output from = > DOCMD

```
Headers being shown for command: /DIS NODE ALL
Variable (header) #1 = RECTYPE
Variable (header) #2 = NODE_SUB
Variable (header) #3 = TYPE
Variable (header) #4 = CID
Variable (header) #5 = RECD
Variable (header) #6 = ENQCT
Variable (header) #7 = DEQCT
Variable (header) #8 = QCT
Variable (header) #9 = SENT
EXEC DOCMD ended with RC= 0
```

Figure 75. Output from = > DOCMD /DIS NODE ALL;?

```
Selection criteria =>CID>0<= Command: /DIS NODE ALL
NODE_SUB TYPE   CID       RECD ENQCT DEQCT   QCT  SENT
L3270A   3277   01000004     5    19    19     0    26 IDLE CON
L3270C   3277   01000005   116   115   115     0   122 CON
Selected 2 lines from 396 lines.
DOCMD Executed 402 DL/I calls in 2.096787 seconds.
EXEC DOCMD ended with RC= 0
```

Figure 76. Output from = > DOCMD /DIS NODE ALL;CID>0

```
Selection criteria =>TYPE=SLU2<= Command: /DIS NODE ALL
NODE_SUB TYPE   CID       RECD ENQCT DEQCT   QCT  SENT
WRIGHT   SLU2   00000000     0     0     0     0     0 IDLE
Q3290A   SLU2   00000000     0     0     0     0     0 IDLE
Q3290B   SLU2   00000000     0     0     0     0     0 IDLE
Q3290C   SLU2   00000000     0     0     0     0     0 IDLE
Q3290D   SLU2   00000000     0     0     0     0     0 IDLE
V3290A   SLU2   00000000     0     0     0     0     0 IDLE
V3290B   SLU2   00000000     0     0     0     0     0 IDLE
H3290A   SLU2   00000000     0     0     0     0     0 IDLE
H3290B   SLU2   00000000     0     0     0     0     0 IDLE
E32701   SLU2   00000000     0     0     0     0     0 IDLE
E32702   SLU2   00000000     0     0     0     0     0 IDLE
E32703   SLU2   00000000     0     0     0     0     0 IDLE
E32704   SLU2   00000000     0     0     0     0     0 IDLE
E32705   SLU2   00000000     0     0     0     0     0 IDLE
ADLU2A   SLU2   00000000     0     0     0     0     0 IDLE
ADLU2B   SLU2   00000000     0     0     0     0     0 IDLE
ADLU2C   SLU2   00000000     0     0     0     0     0 IDLE
ADLU2D   SLU2   00000000     0     0     0     0     0 IDLE
ADLU2E   SLU2   00000000     0     0     0     0     0 IDLE
ADLU2F   SLU2   00000000     0     0     0     0     0 IDLE
ADLU2X   SLU2   00000000     0     0     0     0     0 IDLE
ENDS01   SLU2   00000000     0     0     0     0     0 IDLE
ENDS02   SLU2   00000000     0     0     0     0     0 IDLE
ENDS03   SLU2   00000000     0     0     0     0     0 IDLE
ENDS04   SLU2   00000000     0     0     0     0     0 IDLE
ENDS05   SLU2   00000000     0     0     0     0     0 IDLE
ENDS06   SLU2   00000000     0     0     0     0     0 IDLE
NDSLU2A1 SLU2   00000000     0     0     0     0     0 ASR IDLE
NDSLU2A2 SLU2   00000000     0     0     0     0     0 ASR IDLE
NDSLU2A3 SLU2   00000000     0     0     0     0     0 ASR IDLE
NDSLU2A4 SLU2   00000000     0     0     0     0     0 ASR IDLE
NDSLU2A5 SLU2   00000000     0     0     0     0     0 IDLE
NDSLU2A6 SLU2   00000000     0     0     0     0     0 ASR IDLE
OMSSLU2A SLU2   00000000     0     0     0     0     0 IDLE
Selected 34 lines from 396 lines.
DOCMD Executed 435 DL/I calls in 1.602206 seconds.
EXEC DOCMD ended with RC= 0
```

Figure 77. Output from = > DOCMD /DIS NODE ALL;TYPE=SLU2

```
Selection criteria =>ENQCT>0 & RECTYPE='T02'<= Command: /DIS TRAN ALL
  TRAN   CLS ENQCT   QCT   LCT  PLCT CP NP LP SEGSZ SEGNO PARLM RC
TACP18     1   119     0 65535 65535  1  1  1     0     0  NONE  1
Selected 1 lines from 1104 lines.
DOCMD Executed 1152 DL/I calls in 5.780977 seconds.
EXEC DOCMD ended with RC= 0
```

Figure 78. Output from = > DOCMD /DIS TRAN ALL;ENQCT>0 & RECTYPE='T02'

```
Selection criteria =>ENQCT>0<= Command: /DIS LTERM ALL
LTERM     ENQCT   DEQCT    QCT
CTRL         19      19      0
T3270LC     119     119      0
Selected 2 lines from 678 lines.
DOCMD Executed 681 DL/I calls in 1.967670 seconds.
EXEC DOCMD ended with RC= 0
```

Figure 79. Output from = > DOCMD /DIS LTERM ALL;ENQCT>0

The source code for the DOCMD exec is shown in Figure 80.

```
/*********************************************************************/
/* A REXX exec that executes an IMS command and parses the           */
/* output by a user supplied criteria.                               */
/*                                                                   */
/*********************************************************************/
/* Format:  tranname DOCMD IMS-Command;Expression                    */
/* Where:                                                            */
/*    tranname is the tranname of a command capable transaction that */
/*             will run the IVPREXX program.                         */
/*    IMS-Command is any valid IMS command that generates a table of */
/*             output like /DIS NODE ALL or /DIS TRAN ALL            */
/*    Expression is any valid REXX expression, using the header names*/
/*             as the variables, like  CID>0 or SEND=0 or more       */
/*             complex like CID>0 & TYPE=SLU2                        */
/* Example: TACP18 DOCMD DIS A               Display active          */
/*          TACP18 DOCMD DIS NODE ALL;?      See headers of DIS NODE */
/*          TACP18 DOCMD DIS NODE ALL;CID>0  Show active Nodes       */
/*          TACP18 DOCMD DIS NODE ALL;CID>0 & TYPE='SLU2'            */
/*********************************************************************/
Address REXXTDLI
Parse Upper Arg Cmd ';' Expression
Cmd=Strip(Cmd);
Expression=Strip(Expression)
If Cmd='' Then Do
   Call SayIt 'Please supply an IMS Command to execute.'
   Exit 0
End
AllOpt= (Expression='ALL')
If AllOpt then Expression='
If Left(Cmd,1)¬='/' then Cmd='/'Cmd   /* Add a slash if necessary */
If Expression='' Then
   Call SayIt 'No Expression supplied, all output shown',
        'from:' Cmd
Else If Expression='?' Then
   Call SayIt 'Headers being shown for command:' Cmd
Else
   Call SayIt 'Selection criteria =>'Expression'<=',
        'Command:' Cmd
x=Time('R'); Calls=0
ExitRC= ParseHeader(Cmd,Expression)
If ExitRC¬=0 then Exit ExitRC
If Expression='?' Then Do
   Do i=1 to Vars.0
      Call SayIt 'Variable (header) #'i '=' Vars.i
      Calls=Calls+1
   End
End
```

Figure 80. DOCMD Exec: Process an IMS Command (Part 1 of 3)

```
Else Do
   Call ParseCmd Expression
   Do i=1 to Line.0
      If AllOpt then Line=Line.i
      Else Line=Substr(Line.i,5)
      Call SayIt Line
      Calls=Calls+1
   End
   If Expression¬='' then
      Call SayIt 'Selected' Line.0-1 'lines from',
           LinesAvail 'lines.'
   Else
      Call SayIt 'Total lines of output:' Line.0-1
   Call SayIt 'DOCMD Executed' Calls 'DL/I calls in',
        Time('E') 'seconds.'
    End
    Exit 0
ParseHeader:
   CurrCmd=Arg(1)
   CmdCnt=0
   'CMD IOPCB CURRCMD'
   CmdS= IMSQUERY('STATUS')
   Calls=Calls+1
   If CmdS='  ' then Do
      Call SayIt 'Command Executed, No output available.'
      Return 4
   End
   Else If CmdS¬='CC' then Do
      Call SayIt 'Error Executing Command, Status='CmdS
      Return 16
   End
   CurrCmd=Translate(CurrCmd,' ','15'x)  /* Drop special characters  */
   CurrCmd=Translate(CurrCmd,'__','-/')  /* Drop special characters  */
   CmdCnt=CmdCnt+1
   Interpret 'LINE.'||CmdCnt '= Strip(CurrCmd)'
   Parse Var CurrCmd  RecType Header
   If Expression='' then Nop
   Else If Right(RecType,2)='70' then Do
      Vars.0=Words(Header)+1
      Vars.1 = "RECTYPE"
      Do i= 2 to Vars.0
         Interpret 'VARS.'i '= "'Word(CurrCmd,i)'"'
      End
   End
   Else Do
      Call SayIt 'Command did not produce a header',
                  'record, first record's type='RecType
      Return 12
   End
Return 0
```

Figure 80. DOCMD Exec: Process an IMS Command (Part 2 of 3)

```
ParseCmd:
   LinesAvail=0
   CurrExp=Arg(1)
   Do Forever
      'GCMD IOPCB CURRCMD'
      CmdS= IMSQUERY('STATUS')
      Calls=Calls+1
      If CmdS¬='  ' then Leave
      /* Skip Time Stamps       */
      If Word(CurrCmd,1)='X99' & Expression¬='' then Iterate
      LinesAvail=LinesAvail+1
      CurrCmd=Translate(CurrCmd,' ','15'x)/* Drop special characters */
      If Expression='' then OK=1
      Else Do
         Do i= 1 to Vars.0
            Interpret Vars.i '= "'Word(CurrCmd,i)'"'
         End
         Interpret 'OK='Expression
      End
      If OK then Do
         CmdCnt=CmdCnt+1
         Interpret 'LINE.'||CmdCnt '= Strip(CurrCmd)'
      End
   End
   Line.0 = CmdCnt
   If CmdS¬='QD' Then
      Call SayIt 'Error Executing Command:',
                  Arg(1) 'Stat='CmdS
Return


SayIt: Procedure
   Parse Arg Line
   'ISRT IOPCB LINE'
Return RC
```

Figure 80. DOCMD Exec: Process an IMS Command (Part 3 of 3)

IVPREXX: MPP/IFP Front End for General Exec Execution

The IVPREXX exec is a front-end generic exec that is shipped with IMS as part of the IVP. It runs other execs by passing the exec name to execute after the TRANCODE (IVPREXX). For further details on IVPREXX, see “IVPREXX Sample Application” on page 403. For the latest version of the IVPREXX source code, see the IMS.DFSEXECA distribution library; member name is IVPREXX.

Part 4. For Your Reference

Chapter 14. Summary of TM Message and System Service Calls 439
Transaction Management Call Summary 439
System Service Call Summary. 440

Chapter 15. DL/I Status Codes 443
Status Code Tables. 443
Categories of DL/I Status Codes 443
Status Code Explanations 452

Chapter 16. DL/I Return and Reason Codes 475
Return and Reason Code Tables. 475
DL/I Return and Reason Code Explanations 491

© Copyright IBM Corp. 1974, 2000

Chapter 14. Summary of TM Message and System Service Calls

This chapter contains tables that summarize the transaction management message calls and the system service calls.

In this Chapter:

- "Transaction Management Call Summary"
- "System Service Call Summary" on page 440

Transaction Management Call Summary

Table 50 shows the parameters that are valid for each of the transaction management message calls. Optional parameters are enclosed in brackets, [ ].

Exception: Language-dependent parameters are not shown here. The variable *parmcount* is required for all PLITDLI calls. Either *parmcount* or **VL** is required for assembler language calls. Parmcount is optional in COBOL, C, and Pascal programs. See "Formatting DL/I Calls for Language Interfaces" on page 31 for language-specific information.

Related Reading: For detailed information on each call, its parameters, usage, and restrictions, see "Chapter 3. Writing DL/I Calls for Transaction Management" on page 59. For information on writing calls with programming language interfaces, see "Chapter 2. Defining Application Program Elements" on page 31.

Table 50. Summary of TM Message Calls

Function Code	Meaning	Use	Parameters	Valid for
AUTH	Authorization	Verifies user's security authorization.	function, i/o pcb or aib, i/o area	DB/DC, DCCTL
CHNG	Change	Sets destination on modifiable alternate PCB	function, alt pcb or aib, destination name[, options list, feedback area]	DB/DC, DCCTL
CMD	Command	Used by a program to issue IMS commands	function, i/o pcb or aib, i/o area	DB/DC, DCCTL
GCMD	Get Command	Retrieves second and any subsequent responses to a command	function, i/o pcb or aib, i/o area	DB/DC, DCCTL
GN	Get Next	Retrieves second and any subsequent message segments	function, i/o pcb or aib, i/o area	DB/DC, DCCTL
GU	Get Unique	Retrieves the first segment of a message	function, i/o pcb or aib, i/o area	DB/DC, DCCTL
ISRT	Insert	Builds an output message in a program's I/O area	function, i/o or alt pcb or aib, i/o area [,mod name.]	DB/DC, DCCTL

© Copyright IBM Corp. 1974, 2000

Transaction Management Call Summary

Table 50. Summary of TM Message Calls (continued)

Function Code	Meaning	Use	Parameters	Valid for
PURG	Purge	Enqueues messages from a PCB to destinations	function, i/o or alt pcb or aib[, i/o area, mod name.]	DB/DC, DCCTL
SETO	Set options. Sets processing options for advanced print functions and APPC/IMS message processing.	Feedback area returns information about errors in the options list.	function, i/o pcb or alternate pcb or aib, i/o area, options list[, feedback area]	BMP, MPP, IFP DB/DC, DCCTL

System Service Call Summary

Table 51 is a summary of which system service calls you can use in each type of IMS TM application program, and the parameters for each call. Optional parameters are shown in brackets ([ ]).

System service calls issued in a DCCTL environment must refer only to I/O PCBs or GSAM database PCBs. Calls that cannot be used in a DCCTL environment are noted.

Language-dependent parameters are not shown here. For language-specific information, see “Formatting DL/I Calls for Language Interfaces” on page 31.

For detailed information on each call, its parameters, usage, and restrictions see “Chapter 4. Writing DL/I Calls for System Services” on page 89. For information on writing calls with programming language interfaces see “Chapter 2. Defining Application Program Elements” on page 31.

Table 51. Summary of System Service Calls

Function Code	Meaning and Use	Options	Parameters	Valid for
APSB	Allocate PSB. Allocates a PSB for use in CPI-C driven application programs.	None	function, aib	MPP
CHKP (Basic)	Basic checkpoint. For recovery purposes.	None	function, i/o pcb or aib, i/o area	batch, BMP, MPP
CHKP (Symbolic)	Symbolic checkpoint. For recovery purposes.	Can specify seven program areas to be saved.	function, i/o pcb or aib, i/o area length, i/o area[, area length, area]	batch, BMP
DPSB	Deallocate PSB. Frees a PSB in use by a CPI-C driven application program.	None	function, aib	MPP
GMSG	Retrieve a message from the AO exit routine.	Can wait for an AOI message when none is available.	function, aib, i/o area	DB/DC and DCCTL(BMP, MPP, IFP), DB/DC and DBCTL(DRA thread), DBCTL(BMP non-message driven)

Table 51. Summary of System Service Calls (continued)

Function Code	Meaning and Use	Options	Parameters	Valid for
GSCD [1]	Get the address of the system contents directory.	None	function, i/o pcb or aib, i/o area	batch
ICMD	Issue an IMS command and retrieve the first command response segment.	None	function, aib, i/o area	DB/DC and DCCTL(BMP, MPP, IFP), DB/DC and DBCTL(DRA thread), DBCTL(BMP non-message driven)
INIT	Application receives data availability status codes.	Checks each PCB for data availability.	function, i/o pcb or aib, i/o area	batch, BMP, MPP, IFP
INQY	Inquiry. Retrieves information about output destinations, session status, execution environment, and the PCB address.	None	function, aib, i/o area	batch, BMP, MPP, IFP
LOGb	Log. Write a message to the system log.	None	function, i/o pcb or aib, i/o area	batch, BMP, MPP, IFP
RCMD	Retrieve the second and subsequent command response segments resulting from an ICMD call.	None	function, aib, i/o area	DB/DC and DCCTL(BMP, MPP, IFP), DB/DC and DBCTL(DRA thread), DBCTL(BMP non-message driven)
ROLB	Rollback. Backs out messages sent by the application program.	Call returns last message to i/o area.	function, i/o pcb or aib[, i/o area]	batch, BMP, MPP, IFP
ROLL	Roll. Backs out output messages and terminates the conversation.	None	function	batch, BMP, MPP
ROLS	Returns message queue positions to sync points set by the SETS or SETU call.	Issues call with i/o PCB or aib	function, i/o pcb or aib i/o area, token	batch, BMP, MPP, IFP
SETS	Sets intermediate sync (backout) points.	Cancels all existing backout points. Can establish up to 9 backout points.	function, i/o pcb or aib, i/o area, token	batch, BMP, MPP, IFP
SETU	Sets intermediate sync (backout) points.	Cancels all existing backout points. Can establish up to 9 backout points.	function, i/o pcb or aib, i/o area, token	batch, BMP, MPP, IFP
SYNC	Synchronization	Request commit point processing.	function, i/o pcb or aib	BMP
XRST	Restart. Works with symbolic CHKP to restart application program failure.	Can specify up to 7 areas to be saved.	function, i/o pcb or aib, i/o area length, i/o area[, area length, area]	batch, BMP

Table 51. Summary of System Service Calls (continued)

Function Code	Meaning and Use	Options	Parameters	Valid for

Note:

1. GSCD is a Product-sensitive programming interface.

Chapter 15. DL/I Status Codes

This section contains reference information on all IMS status codes. The information is divided into two parts:

- Status code tables
  - Database Calls
  - Message Calls
  - System Service Calls
- Status code explanations

Status Code Tables

The status code tables briefly explain each status code and list the calls for which you can receive each status code. The tables also include a column of numbers representing the category of each status code; the numbers and the corresponding explanations are below.

For information about each of the status codes, see Table 52, Table 53 on page 447, and Table 54 on page 450.

Exception: Although the calls APSB, DPSB, and ROLL are included in Table 54 on page 450, they do not receive status codes.

Categories of DL/I Status Codes

The numbers in the category column of the status codes tables refer to the following categories of status codes:

1. Those indicating exceptional but valid conditions. The call is completed.
2. Those indicating warning or information-only status codes on successful calls (for example, GA and GK). If the call requested data, IMS returns the data to the I/O area. The call is completed.
3. Those indicating warning status codes on successful calls when data is not returned to the I/O area. The call is completed.
4. Those indicating improper user specifications. Most status codes are in this category. The call is not completed.
5. Those indicating system, I/O, or security errors encountered during the execution of I/O requests. The call is not completed.
6. Those indicating unavailable data.

Table 52. Database Calls

PCB Status Code	CLSE (GSAM)	DEQ	GU, GHU	GN, GHN	GNP, GHNP	DLET, REPL	ISRT (LOAD)	ISRT (ADD)	FLD	POS	OPEN (GSAM)	QUERY	REFRESH	TERM	Category	Description
AB		X	X	X	X	X	X	X	X	X			X		4	Segment I/O area required; none specified in call. Only applies to full-function DEQ calls.
AC			X	X	X	X	X	X							4	Hierarchic error in SSAs.

© Copyright IBM Corp. 1974, 2000

Table 52. Database Calls (continued)

PCB Status Code	CLSE (GSAM)	DEQ	GU, GHU	GN, GHN	GNP, GHNP	DLET, REPL	ISRT (LOAD)	ISRT (ADD)	FLD	POS	OPEN (GSAM)	QUERY	REFRESH	TERM	Category	Description
AD		X	X	X	X	X	X	X	X	X			X		4	Function parameter incorrect. Only applies to full-function DEQ calls.
AF			X	X											4	GSAM detected invalid variable-length record.
AH							X	X							4	Required SSA missing. Options list not specified in SETO call.
AI			X	X	X	X	X	X							5	Data management OPEN error.
AJ		X	X	X	X	X	X	X	X						4	Incorrect parameter format in I/O area; incorrect SSA format; incorrect command used to insert a logical child segment. I/O area length in AIB is invalid; incorrect class parameter specified in Fast Path Q command code.
AK			X	X	X		X	X	X	X					4	Invalid SSA field name.
AM			X	X	X	X	X	X	X	X					4	Call function not compatible with processing option, segment sensitivity, transaction code, definition, or program type.
AO			X	X	X	X	X	X		X					5	I/O error: OSAM, BSAM, or VSAM.
AT						X	X	X							4	User I/O area too long.
AU			X	X	X	X	X	X							4	SSAs too long.
BA			X	X	X	X		X							6	Call could not be completed because data was unavailable.
BB			X	X	X	X		X							6	Call could not be completed because data was unavailable and updates are backed out only since the last commit point.
DA						X			X						4	Segment key field or nonreplaceable field has been changed.
DJ						X									4	No preceding successful GHU or GHN call or an SSA supplied at a level not retrieved.
DX						X									4	Violated delete rule.
EM										X						Normally for a utility.
FA			X						X						4	MSDB arithmetic overflow error occurred.

Table 52. Database Calls (continued)

PCB Status Code	CLSE (GSAM)	DEQ	GU, GHU	GN, GHN	GNP, GHNP	DLET, REPL	ISRT (LOAD)	ISRT (ADD)	FLD	POS	OPEN (GSAM)	QUERY	REFRESH	TERM	Category	Description
FC										X					4	POS call for direct dependent segments only.
FD			X	X	X	X	X	X	X						2	Deadlock occurred.
FE									X						4	FSA error, not field name.
FF								X							3	No space in MSDB.
FG									X						4	Combination of FE and FW codes.
FH			X	X	X	X	X	X		X					3	DEDB inaccessible.
FI			X	X	X	X	X	X		X					4	I/O area not in user's dependent region.
FM			X	X	X		X	X		X					4	Randomizing routine return code = 4.
FN									X						4	FSA error, field name.
FP						X		X	X						4	Invalid hexadecimal or decimal data.
FR			X	X	X	X	X	X	X	X					5	Total buffer allocation exceeded.
FS							X								3	DEDB areas are full.
FT			X	X	X	X	X			X					4	Too many SSAs on call.
FV									X						3	MSDB verify condition failed.
FW		X	X	X	X	X	X	X	X	X					2	More resources needed than normally allowed. For the DEQ call, Fast Path was not able to release any buffers.
FY			X	X	X	X	X	X	X	X					4	Attempt to read sequential data preceding the current position.
GA				X	X										2	Crossing hierarchical boundary.
GB				X											1	End of database.
GC			X	X			X			X					3	Crossing unit of work (UOW) boundary.
GD								X							1	Call did not have SSAs for all levels above insert and has lost segment position.
GE			X	X	X			X							1	Segment not found.
GG			X	X	X										5	Segment contains invalid pointer.
GK				X	X										2	Crossing segment boundaries on same level.
GL		X													4	Invalid user log code. Only applies to full-function DEQ calls.

Table 52. Database Calls (continued)

PCB Status Code	CLSE (GSAM)	DEQ	GU, GHU	GN, GHN	GNP, GHNP	DLET, REPL	ISRT (LOAD)	ISRT (ADD)	FLD	POS	OPEN (GSAM)	QUERY	REFRESH	TERM	Category	Description
GP					X			X		X					4	No parentage established.
HT										X						Normally for a utility.
II								X							3	Segment already exists.
IX								X							4	Violated insert rule.
LB							X								1	Segment being loaded already exists in database.
LC							X								4	Key field of segments out of sequence.
LD							X								4	No parent for this segment has been loaded.
LE							X								4	Sequence of sibling segments not the same as DBD sequence.
LS								X							1	Work may be backed out because sufficient CI space was not preallocated for the area.
NA												X	X		6	A database was unavailable.
NE						X									3	DL/I call issued by index maintenance cannot find segment.
NI						X	X	X							1	Index maintenance found duplicate segment in index.
NO						X	X	X							5	I/O error: OSAM, BSAM, or VSAM.
NU												X	X		6	A database was unavailable for update.
OS										X						Normally for a utility.
RX						X									4	Violated replace rule.
TH												X			4	No PSB was scheduled (command-level only).
TI								X							4	Invalid path to segment (command-level only).
			X	X	X	X		X						X	5	DL/I not active (command-level only).
			X	X	X	X	X	X							5	Invalid system DIB (command-level only).
TO						X									4	Path replace error (command-level only).
TP			X	X	X	X	X	X							4	Invalid number for PCB or invalid processing option (command-level only).

Table 52. Database Calls (continued)

PCB Status Code	CLSE (GSAM)	DEQ	GU, GHU	GN, GHN	GNP, GHNP	DLET, REPL	ISRT (LOAD)	ISRT (ADD)	FLD	POS	OPEN (GSAM)	QUERY	REFRESH	TERM	Category	Description
TR		X	X	X	X	X		X		X				X	4	CICS XDLIPRE user exit determined the preceding request should not be executed.
TY			X	X	X	X	X	X							5	Database not open (command-level only).
TZ			X	X	X	X		X			X				5	Length of segment greater than 64 KB.
UC							X								1	Checkpoint taken (Utility Control Facility (UCF) status code).
US							X								1	Stop (UCF status code).
UX							X								1	Checkpoint and stop (UCF status code).
V1						X	X	X							4	Segment length not within limits of DBDGEN.
V2			X	X	X	X	X	X			X			X	4	Segment length invalid (command-level only).
V3			X	X	X			X							4	Field length missing or invalid (command-level only).
V4			X	X	X	X	X	X							4	Length of variable-length segment invalid (command-level only).
V5			X	X	X	X		X							4	Offset if invalid (command-level only).
V6			X	X	X	X		X							4	Concatenated key length invalid (command-level only).
XX			X	X			X	X							5	Internal GSAM error.
ƀƀ[1]	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	No status code returned. Proceed.

Note:

1. ƀƀ indicates blank.

Table 53. Message Calls

PCB Status Code	AUTH	GU	GN	ISRT	CHNG	CMD	GCMD	PURG	SETO	Category	Description
AA				X	X					4	CHNG call for alternate response PCB can specify only logical terminal destination; transaction code destination specified.
AB	X	X	X	X	X	X	X			4	Segment I/O area required; none specified in call.
AD		X	X	X	X	X	X	X	X	4	Function parameter invalid.
AH									X	4	Required SSA missing. Options list not specified in SETO call.

Table 53. Message Calls (continued)

PCB Status Code	AUTH	GU	GN	ISRT	CHNG	CMD	GCMD	PURG	SETO	Category	Description
AJ									X	4	Invalid parameter format in I/O area; invalid SSA format; invalid command used to insert a logical child segment. I/O area length in AIB is invalid.
AL	X	X	X	X	X	X	X	X		4	Call using I/O PCB in batch program.
AP	X	X	X	X	X	X	X	X		4	Specifying more than four user call parameters for a TP PCB is not valid.
AR					X				X	4	Error in option list related to IMS/ESA option keyword.
AS					X				X	4	The PRTO= option contained invalid data set processing options.
AT	X			X				X		4	User I/O area too long.
AX	X			X	X				X	5	System error. Call not completed successfully.
AY				X						4	Alternate response PCB referenced by ISRT call has more than one physical terminal assigned for input purposes. Notify master terminal operator.
AZ								X		4	The conversational program has issued a PURG call to PCB that cannot be purged.
A1	X				X					4	AUTH call attempted with invalid generic class name or error occurred attempting to set destination name specified in the CHNG call.
A2					X					4	Call attempted with invalid PCB (PCB not modifiable or ISRT operation already done).
A3				X				X		4	Call attempted to a modifiable TP PCB with no destination set.
A4	X			X	X					4	Security violation detected during processing of either an AUTH call, a CHNG call, or an ISRT on a conversational response.
A5				X				X		4	Format name specified on second or subsequent message ISRT or PURG.
A6				X						4	Output segment size limit exceeded on call.
A7				X						4	Number of output segments inserted exceeded the limit by one. Any further queue manager calls are prohibited to prevent message queue overflow.
A8				X						4	ISRT to alternate response PCB followed ISRT to I/O PCB or ISRT to I/O PCB followed ISRT to alternate response PCB.
A9				X						4	Alternate response PCB referenced by call requires that the source physical terminal receive the output response.
CA						X				4	No such command. No command responses produced.
CB						X				4	Command, as entered, not allowed for AOI. No command responses produced.
CC						X				2	Command executed. One or more command responses produced.
CD						X				4	Entered command violates security. No command responses produced.
CE		X								2	Transaction rescheduled after CMD call. Commit point had not been reached.

Table 53. Message Calls (continued)

PCB Status Code	AUTH	GU	GN	ISRT	CHNG	CMD	GCMD	PURG	SETO	Category	Description
CF		X								2	Message on queue before IMS was last started.
CG		X								2	Transaction originated from AOI exit routine.
CH						X				5	AOI detected system error; CMD request not processed. Reissue CMD call.
CI		X								2	Transaction on queue before IMS last started. Transaction rescheduled. Commit point not reached.
CJ		X								2	Transaction from AOI exit routine. Message rescheduled. Commit point not reached.
CK		X								2	Transaction from AOI exit routine. Message on queue before IMS last started.
CL		X								2	Transaction from AOI exit routine. Message on queue before IMS last started. Message rescheduled. Commit point had not been reached.
CM						X				3	Command executed. No command response produced.
CN						X				4	IOASIZE= parameter on PSBGEN macro does not meet minimum requirement for CMD call.
FF		X								3	No space in MSDB.
FH		X								3	DEDB inaccessible.
FI			X	X	X					4	I/O area not in user's dependent region.
FS		X								3	DEDB areas are full.
FV		X								3	MSDB verify condition failed.
MR				X						–	Reserved
QC		X								3	No more input messages exist.
QD			X				X			3	No more segments exist for this message.
QE			X				X			4	GN request before GU. GCMD request before CMD.
QF		X		X				X	X	4	Segment less than five characters (segment length is message text length plus four control characters).
QH				X				X	X	4	Terminal symbolic error; output designation unknown to IMS (logical terminals or transaction code).
TG	X									4	No PSB was scheduled (command-level only).
	X									5	Invalid system DIB (command-level only).
TP	X									4	Invalid number for PCB or invalid processing option (command-level only).
TY	X									5	Database not open (command-level only).
TZ	X									5	Length of segment greater than 64 KB.
XA				X						4	Attempt to continue processing the conversation by passing SPA by a program-to-program switch after answering terminal.
XB				X						4	Program passed SPA to other program, but trying to respond.
XC				X						4	Program inserted message with Z1 field bits set. These bits are reserved for system use.
XE				X						4	Tried to ISRT SPA to express PCB.

Status Code Tables

Table 53. Message Calls (continued)

PCB Status Code	AUTH	GU	GN	ISRT	CHNG	CMD	GCMD	PURG	SETO	Category	Description
XF				X						4	Alternate PCB specified in ISRT call for SPA had destination set to a logical terminal, but was not defined as ALTRESP=YES. MSC direct routing does not support program-to-program switch between conversational transactions.
XG				X						4	Current conversation requires fixed-length SPAs. Attempt was made to insert SPA to transaction with a different or nonfixed-length SPA.
X2				X				X		4	First insert to transaction code PCB that is conversational is not a SPA.
X3				X						4	Invalid SPA.
X4				X						4	Insert to a transaction code PCB that is not conversational and the segment is a SPA.
X5				X						4	Insert of multiple SPAs to transaction code PCB.
X6				X						4	Invalid transaction code name inserted into SPA.
X7				X						4	Length of SPA is incorrect (user modified first 6 bytes).
X8				X						5	Error attempting to queue a SPA on a transaction code PCB.
␣␣[1]	X	X	X	X	X	X	X	X	X	1	No status code returned. Proceed.

Note:

1. ␣␣ indicates blank.

Table 54. System Service Calls

PCB Status Code	CHKP	INIT	INQY	LOG	PCB	ROLB	ROLL	ROLS	SETS	SETU	SNAP 1	STAT 2	SYNC	XRST	Category	Description
AB	X	X	X	X								X			4	Segment I/O area required; none specified in call.
AC												X			4	Hierarchic error in SSAs.
AD	X	X	X	X		X	X	X	X			X	X	X	4	Function parameter invalid.
AG			X												4	Partial data return. I/O area too small.
AJ		X	X					X							4	Invalid parameter format in I/O area; invalid SSA format; invalid command used to insert a logical child segment. I/O area length in AIB is invalid.
AL						X		X	X						4	Call using I/O PCB in batch program.
AP	X														4	More than 4 user call parameters for a TP PCB are invalid.
AQ			X												4	Invalid subfunction code.
AT				X											4	User I/O area too long.
BC		X													6	Call could not be completed because of a deadlock occurrence; updates are backed out only since the last commit point.

Table 54. System Service Calls (continued)

PCB Status Code	CHKP	INIT	INQY	LOG	PCB	ROLB	ROLL	ROLS	SETS	SETU	SNAP 1	STAT 2	SYNC	XRST	Category	Description
BJ			X												6	All of the databases included in the PSB are unavailable or no database PCBs are in the PSB.
BK			X												6	At least one of the databases included in the PSB is unavailable or has limited availability, or at least one PCB received an NA or NU status code.
FA	X												X		4	MSDB arithmetic overflow error occurred.
FD		X											X		3	Deadlock occurred.
FF	X														3	No space in MSDB.
FH	X												X		3	DEDB inaccessible.
FI	X					X									4	I/O area not in user's dependent region.
FS	X												X		3	DEDB areas are full.
FV	X												X		3	MSDB verify condition failed.
GA												X			2	Crossing hierarchic boundary.
GE	X											X	X		1	Segment not found.
GL				X											4	Invalid user log code.
NA		X													6	A database was unavailable.
NL	X														4	XEFRDER card not provided. Please supply one.
NU		X													6	A database was unavailable for update.
QC	X														3	No more input messages exist.
QE	X					X									4	GN request before GU. GCMD request before CMD.
QF	X														4	Segment less than five characters (segment length is message text length plus four control characters).
RA								X							4	Token does not match one for a SETS, or the PCB did not get BA or BB on last call.
RC								X							4	ROLS call issued with unsupported PCBs in the PSB, or the program is using an attached subsystem.
SA									X						5	Insufficient space.
SB									X						4	Would exceed maximum number of levels allowed.
SC									X	X					5	A SETS/SETU call was issued with unsupported PCBs in the PSB, or the program is using an attached subsystem.
TA					X										5	PSB not in PSB directory (command-level only).
TC					X										4	PSB already scheduled (command-level only).

Table 54. System Service Calls (continued)

PCB Status Code	CHKP	INIT	INQY	LOG	PCB	ROLB	ROLL	ROLS	SETS	SETU	SNAP [1]	STAT [2]	SYNC	XRST	Category	Description
TE					X										5	PSB initialization failed (command-level only).
TJ					X										5	DL/I not active (command-level only).
TL					X										4	Conflict in scheduling intent (command-level only).
TN	X			X	X	X	X					X		X	5	Invalid system DIB (command-level only).
TP					X							X			4	Invalid number for PCB or invalid processing option (command-level only).
TR				X	X				X			X			4	CICS XDLIPRE user exit determined the preceding request should not be executed.
TY					X							X			5	Database not open (command-level only).
TZ					X							X			5	Length of segment greater than 64 KB.
V2					X										4	Segment length invalid (command-level only).
V7												X			4	Statistics area length invalid (command-level only).
XD	X												X		1	IMS terminating. Further DL/I calls must not be issued. No message returned.
␣␣[3]	X	X	X	X	X	X		X	X	X	X	X	X	X	1	Good. No status code returned. Proceed.

Notes:

1. SNAP is a Product-sensitive programming interface.
2. STAT is a Product-sensitive programming interface.
3. ␣␣ indicates blank.

Status Code Explanations

This information appears in the three application programming guides. For EXEC DL/I commands, all status codes, except those identified as being returned to the application program, cause an abnormal termination of the application program.

All explanations apply to both DL/I call (call-level) programs and EXEC DLI (command-level) programs except where split. The term “request” means call, command, or both.

AA

Explanation: IMS ignored a CHNG or ISRT call because the alternate response PCB that is referenced in the call specified a transaction code as a destination. An alternate response PCB must have a logical terminal specified as its destination.

Programmer Response: Correct the CHNG or ISRT call.

AB

Explanation: An I/O area is required as one of the parameters on this call, and the call did not specify one. After this status code is returned, your position in the database is unchanged. AB only applies to full-function DEQ calls.

Programmer Response: Correct the call by including the address of an I/O area as one of the call parameters.

AC

Explanation:

For call-level programs:

An error is in an SSA for a DLET, Get, ISRT, or REPL call for one of these reasons:

- DL/I could not find a segment in the DB PCB specified in the call that has the segment name given in the SSA.
- The segment name is in the DB PCB, but the SSA specifying that segment name is not in its correct hierarchic sequence.
- The call specifies two SSAs for the same hierarchic level.

IMS also returns this status code when a STAT [5] call has an invalid statistics function. After this status code is returned, your position in the database is unchanged.

For command-level programs:

An error is in one of the WHERE or SEGMENT options on a Get or ISRT command for one of these reasons:

- DL/I could not find a segment in the DB PCB specified in the segment name given in the SEGMENT option.
- The segment name is in the DB PCB, but the qualification for the command does not specify it in its correct hierarchic sequence.
- The command specifies two SEGMENT options for the same hierarchic level.

Programmer Response: Correct the segment name in the SSA or SEGMENT option or in the statistics function of the STAT[5] call.

AD

Explanation:

For call-level programs:

Either the function parameter on the call is invalid or the function is not supported for the type of PCB specified in the call. Only applies to full-function DEQ calls. Some possible reasons are:

- The function parameter is invalid.
- A system service call used a PCB that is not the I/O PCB.
- A call issued in a DCCTL environment referred to an unsupported PCB or database.
- A message GU or GN call used an alternate PCB instead of the I/O PCB.
- A database call used a PCB that is not a DB PCB.
- A message GU used the I/O PCB without specifying IN=trancode in the BMP JCL.
- A SETS or ROLS call included the I/O area but omitted the token.
- A CPI Communications driven program issued the SETO call on the I/O PCB.
- A call was issued from an IFP region on an I/O PCB.
- Invalid subsystem level for spool API processing.

For command-level programs:

A command was issued that is not supported in the environment. An example is a system service command in an online program. If the command is correct, some other possible causes are:

- Referencing a DB PCB on a system service command. System service commands must reference the I/O PCB.
- Referencing an I/O PCB for a database command, or not defining an I/O PCB before issuing system service commands.
- A command issued in a DCCTL environment referred to an unsupported database or DB PCB.

Programmer Response: Be sure that the specified function is valid for the PCB specified by the request.

AF

Explanation: GSAM detected a variable-length record whose length is invalid on a GU, GHU, GN, or GHN call.

Programmer Response: Correct the program.

AG

Explanation: During INQY call processing, the I/O area was not large enough to contain all the output data. The I/O area was filled with partial data, as much as would fit in the area provided. AIBOALEN contains the actual length of the data returned to the application and AIBOAUSE contains the output area length that is required for the application program to receive all the data.

Programmer Response: Correct the application program by using a larger I/O area. The minimum size of the I/O area is the value contained in the AIBOAUSE field.

AH

Explanation: You get this status code if you:

1. Specify an options list parameter that was not specified in the call list.
2. The program issued an ISRT call that did not include any SSAs. The ISRT call requires an SSA.
3. If the program was issuing a GU call to a GSAM database, the GU did not specify an RSA. RSAs are

5. STAT is a Product-sensitive programming interface.

required on GU calls to GSAM databases. After this status code is returned, your position in the database is unchanged.

Programmer Response: For cause 2, correct the ISRT call by including a qualification; or for cause 3, correct the GU call by adding an RSA to the call.

AI

Explanation: A data management open error occurred. Some possible reasons are:

- An error is in the DD statements.
- Neither DD statements nor DFSMDA dynamic allocation members were provided for this database.
- The data set OPEN request did not specify load mode, but the data set was empty. An empty data set requires the load option in the PCB.
- The buffer is too small to hold a record that was read at open time.
- No DD statements or DFSMDA members were supplied for logically related databases.
- For an OSAM data set, the DSORG field of the OSAM DCB, DSCB, or JFCB does not specify PS or DA.
- For an old OSAM data set, the BUFL or BLKSIZE field in the DSCB is 0.
- The data set is being opened for load, and the processing option for one or more segments is other than L or LS.
- The allocation of the OSAM data set is invalid. The allocation is probably (1,,1), rather than (1,1) and this causes the DSORG to be P0.
- The processing option is L, the OSAM data set is old, and the DSCB LRECL, BLKSIZE, or both, does not match the DBD LRECL, BLKSIZE, or both.
- Incorrect or missing information prevented IMS from determining the block size or the logical record length.
- A catalog was not available for accessing a VSAM database that was requested.
- OS could not perform an OPEN, but the I/O request is valid. Either the data definition information is incorrect, or information is missing.
- RACF was used to protect the OSAM data set, and the control region has no update authorization.

If IMS returns message DFS0730I, you can determine the cause of the OPEN failure from this message in the job log. For more information, see the description of this message in *IMS/ESA Messages and Codes*.

Programmer Response: These kinds of problems often require the help of a system programmer or system administrator. But before you go to one of these specialists, some things you can do are:

- Check the DD statements. Make sure that the ddname is the same as the name specified on the DATASET statement of the DBD. The segment name area in the DB PCB (call level), or in the DIB (command level) has the ddname of the data set that could not be opened.
- Check the PSB and make sure that the appropriate processing options have been specified for each of the DB PCBs that your program uses.

AJ

Explanation: For call-level programs:

For calls that provide parameters in the I/O area, such as SETS, ROLS, and INIT, the format of the parameters in the I/O area is invalid.

For database calls that include SSAs, such as Get, DLET, REPL, and ISRT, the format of one of the SSAs is invalid. The number in the segment level number field of the DB PCB is the level number of the SSA that is invalid. Some possible reasons for the invalid SSA format are:

- The SSA contains a command code that is invalid for that call.
- The relational operator in the qualification statement is invalid.
- A qualification statement is missing a right parenthesis or a Boolean connector.
- A DLET call has multiple or qualified SSAs.
- A REPL call has qualified SSAs.
- An ISRT call has the last qualified SSA.
- An ISRT call that inserts a logical child segment into an existing database includes the D command code. ISRT calls for logical child segments cannot be path calls.
- The RSA parameter on a GSAM call is invalid.
- The SSA used an R, S, Z, W, or M command code for a segment for which no subset pointers are defined in the DBD.
- The subset command codes included in the SSA are in conflict; for example, if one SSA contained an S status code and a Z status code, Fast Path would return an AJ status code. S means to set the pointer to current position; Z means to set the pointer to 0. You could not use these status codes in one SSA.
- The pointer number following a subset pointer command code is invalid. Either you did not include a number, or you included an invalid character. The number following the command code must be between 1 and 8, inclusive.
- The SSA included more than one R command code. An SSA can include only one R command code.
- The specified size for the SSA is too small. After this status code is returned, your position in the database is unchanged.
- In response to a SETS or ROLS call, the length of the I/O area is 0, the LL field is less than 4, or the ZZ field is not 0.

- In response to an INIT call, the format of the I/O area is incorrect.
- For calls that provide the length of the I/O area in the AIB, such as INQY, the I/O area length is invalid.
- For SETO, I/O area length is less than 4096 or less than the minimum.
- For the Q command code, the specified lock class is not a letter (A-J).

For command-level programs:

An ISRT command attempted to insert a logical child segment using a path command. ISRT commands for logical child segments cannot be path commands.

Programmer Response:

If you receive this status code on a SETS, ROLS, or INIT request, correct the parameters provided in the I/O area.

If you receive this status code on a Get, DLET, REPL, or ISRT request, correct the invalid portion of the SSA. If you receive this status code on a GSAM call, correct the RSA.

AK

Explanation: For call-level programs:

An SSA contains an invalid field name, or the field name is not defined in the DBD. The number in the segment level number field of the DB PCB is the level number of the SSA that contains the invalid name.

You can also receive this status code if the program is accessing a logical child through the logical parent. DL/I returns AK if the field specified in the qualification has been defined for the logical child segment, and it includes (at least partially) the portion of the logical child that contains the concatenated key of the logical parent.

When you are using field-level sensitivity, a field you specified in the SSA has not been defined in the PSB. After this status code is returned, your position in the database is unchanged.

For command-level programs:

A WHERE option contains an invalid field name. (The field name is not defined in the DBD.) The number in the DIBSEGLV field of the DIB is the level number of the WHERE option that contains the invalid name.

Programmer Response: Correct the SSA or WHERE option.

AL

Explanation: You get this status code if you:

1. Issue a message call in a batch program.
2. Issue a ROLB, ROLS, or SETS call from a batch program under one of the following conditions:
   - The system log is not on DASD.
   - The system log is on DASD, but dynamic backout has not been specified using the BKO execution parameter.

Programmer Response: For cause 1, correct the program so that message calls in a batch program are not issued. For cause 2, either change the program or put the log on DASD with BKO specified on the execution parameter.

AM

Explanation: For call-level programs:

The call function is not compatible with the processing option in the PCB, the segment sensitivity, the transaction-code definition, or the program type. The level number in the PCB is the level number of the SSA that is invalid. Some of the reasons you might get this status code are:

- If you issue a retrieval call with the D command code in a program that does not have the P processing option specified in the DB PCB that was used for the call.
- If you issue a DLET or ISRT call to a terminal-related dynamic MSDB from a program with no input LTERM present. An example is a batch-oriented BMP.
- If the subset pointer referenced in the SSA was not defined in the program's PSB. For example, if your program's PSB specifies that your program can use subset pointers 1 and 4, and your SSA references subset pointer 5, Fast Path returns an AM status code to your program.
- If your program tried to use an S, Z, W, or M command code for a subset pointer to which it was not pointer update-sensitive, as defined in the program's PSB.
- If a BMP, a CICS online program, or an MPP issues an ISRT call with the D command code when the program does not have the P processing option specified in the DB PCB that was referenced in the call. Batch programs do not need the P processing option to issue an ISRT call with the D command code, unless the program uses field-level sensitivity.
- If the processing option is L and the program issued a call other than an ISRT call. Load programs can issue only ISRT calls.
- If a DLET, REPL, or ISRT call that references a DB PCB does not have the necessary processing option for that call. The minimum processing options for these calls are D for DLET, R for REPL, and I for ISRT.
- If you issue a DLET, REPL, or ISRT call for a segment to which the program is not sensitive.
- If you issue a CHKP call on a GSAM or VSAM data set opened for output. This code is returned in the GSAM PCB.
- If you issue a GSAM call with an invalid call function code.

- If you issue an ISRT or DLET call for the index target segment or a segment on which the index target is dependent in the physical database while using an alternate processing sequence.
- If you issue a path replace where the program does not have replace sensitivity, command code N is not specified, and the data for the segment is changed in the I/O area.
- If GSAM could not obtain buffer space because the region size is too small. This is shown by the value X'1C' in the field GBCRTNCD.
- If you issue a DLET, ISRT, or REPL call from a program where the TRANSACT macro that was used at IMS system definition specified INQUIRY=YES for the input message.
- If you issue a call from an ETO terminal to a terminal-related MSDB or a non-terminal-related MSDB with terminal-related keys. See *IMS/ESA Administration Guide: Transaction Manager* for more information on ETO.
- If you issue any type of call with update intent to a MSDB from a dynamically defined device such as a LU 6.2, ETO, or OTMA device.

After this status code is returned, your position in the database is unchanged.

For command-level programs:

The command is not compatible with the processing option in the PCB or segment sensitivity. The level number in the DIB is the level number of the qualification that is invalid. Some of the reasons you might get this status code are:

- If you issue a path retrieval command in a program that does not have the P processing option specified in the DB PCB that was used for the call.
- If the processing option is L and the program issued a command other than a LOAD command. Load programs can issue only LOAD commands.
- If you issue a DLET, REPL, or ISRT command that references a DB PCB that does not have the necessary processing option for that command. The minimum processing options for these calls are D for DLET, R for REPL, and I for ISRT.
- If you issue a DLET, REPL, or ISRT command for a segment to which the program is not sensitive.
- If you issue a CHKP command if a GSAM or VSAM data set is open for output.
- If you issue a GSAM call with an invalid call function code.
- If you issue an ISRT or DLET command for the index target segment, or a segment in the physical database on which the index target is dependent, while using an alternate processing sequence.
- If you issue a path replace where the program does not have replace sensitivity, command code N is not specified, and the data for the segment is changed in the I/O area.
- If you issue a call to a GSAM dummy data set. Any call to a GSAM dummy data set is invalid.

Programmer Response: Correct the request, or make the necessary changes in the PCB.

AO

Explanation: A BSAM, GSAM, VSAM, or OSAM physical I/O error occurred. When issued from GSAM, this status code means that the error occurred when:

- A data set was accessed.
- The CLOSE SYNAD routine was entered. The error occurred when the last block of records was written prior to the closing of the data set.

IMS does not return an AO status code for write errors with BSAM, VSAM, and OSAM.

If your program receives this status code after issuing a call, this call does not cause the database to be stopped.

Programmer Response: Determine whether the error occurred during input or output, and correct the problem. These problems usually require the help of a system programmer or system administrator.

AP

Explanation: A message or CHKP call is invalid because more than four parameters (or five if a parameter count is specified) are in a message call or a CHKP call issued in a transaction-oriented BMP. The following exceptions apply:

- A batch-oriented BMP can issue a CHKP call with more than 4 (or 5) parameters.
- One parameter after the I/O area parameter is allowed in order for the application program to specify a MOD name in an ISRT call. It is counted towards the maximum of four (or five) parameters.

Programmer Response: Correct the call.

AQ

Explanation: The AIB contains an invalid subfunction, or the INQY call specifies an invalid function.

Programmer Response: Specify a valid subfunction. Valid INQY call subfunctions are null, DBQUERY, ENVIRON, FIND, or PROGRAM.

AR

Explanation: The options list contains an error that is related to a keyword. The feedback area, if one is provided, will contain additional error information.

Programmer Response: Correct the request.

AS

Explanation: An IAFP specific processing error has occurred. The PRTO= option contained invalid data set processing options. The feedback area, if provided, will contain additional error information.

Programmer Response: Correct the request.

AT

Explanation: The length of the data in the program's I/O area is greater than the area reserved for it in the control region. The length of the area reserved is defined by the ACB utility program, DFSUACB0, and is printed as part of its output.

Programmer Response: If the program is in error, correct the program. If the program is correct, reserve a larger control region by specifying parameters on the PSBGEN statement of PSBGEN.

AU

Explanation: The total length of the SSAs in the database call is greater than the area reserved for them in the control region. The length of the area reserved is defined by the ACB utility program, DFSUACB0, and printed as part of its output. After this status code is returned, your position in the database is unchanged.

Programmer Response: If the program is in error, correct the program. If the program is correct, increase the PSB SSA space defined in the PSBGEN.

AX

Explanation: A failure to get CSA storage, a failure of the DFSLUMIF call, or a processing error with the IAFP Spool API occurred. When this code is returned, diagnostic information is written to the log in a '67D0' log record. Spool API processing errors return a DFS0013E message.

A RACROUTE REQUEST=VERIFY,EVIRON=CREATE (RACF RACINIT) made during an AUTH call for LU 6.2 was unsuccessful.

An OTMA user exit returned invalid routing information. See OTMA return codes in the *IMS/ESA Open Transaction Manager Access Guide*.

Programmer Response: These problems usually require the help of a system programmer or system administrator.

AY

Explanation: IMS ignored a message `ISRT` call because the logical terminal referenced by the alternate response PCB currently has more than one physical terminal assigned to it for input purposes.

Programmer Response: Ask the master terminal operator to determine (using /DISPLAY ASSIGNMENT LTERM x) which physical terminals (two or more) refer to this logical terminal. Use the /ASSIGN command to correct the problem.

AZ

Explanation: IMS ignored a `PURG` or `ISRT` call in a conversational program. Some possible reasons are:

- Issuing a `PURG` call referencing the I/O PCB or an alternate response PCB. Conversational programs can issue `PURG` calls only when the `PURG` call references an alternate PCB that is not an alternate response PCB.
- Issuing a `PURG` call to send the SPA.
- Issuing an `ISRT` or a `PURG` call referencing an alternate PCB that is set for an invalid destination or for a destination that IMS cannot determine.
- Issuing an `ISRT` call referencing an alternate PCB whose destination is a conversational transaction code when the first segment inserted is not the SPA; or when IMS cannot determine whether or not the SPA was the first segment inserted.

Programmer Response: Correct the `PURG` or `ISRT` call.

A1

Explanation: You might get the A1 status for the following reasons:

- AUTH call for LU 6.2 input did not find a PST LU 6.2 extension block or did not find a UTOKEN.
- CHNG call against alternate response PCB when the application program has not yet issued a GU.
- The MSC program routing exit routine (DFSCMPR0) was called while processing a CHNG call and one of the following occurred:
  - The exit routine rejected the call by returning with return code 8 (A1 status code).
  - The exit routine returned with a RC=4 to route the message back to the originating system; however, the originating system has not been determined because the application program has not issued a `GU`.
  - The SYSID returned in R0 by the exit routine is not a valid remote SYSID.
  - The MSNAME pointed to by the address in R1, set by the exit routine, is not a valid remote MSNAME.
- The destination name supplied in the I/O area of a `CHNG` call is invalid.

- The destination name supplied in the I/O area of a CHNG call is valid (the destination is a program and the PCB is not an alternate response PCB), but the transaction is Fast Path exclusive.
- AUTH call parameter list contained an invalid generic CLASS name. No access checking was done.

Programmer Response: Correct the CHNG or AUTH call, MSC program routing exit (DFSCMPR0), or ensure that the specified destination is valid.

A2

Explanation: The program issued a CHNG call against an invalid PCB. The PCB was invalid for one of these reasons:

- It was not an alternate PCB.
- It was an alternate PCB, but it was not modifiable.
- It was being used to process a message and had not completed processing it.

Programmer Response: Check the PCB that was used by the CHNG call and determine which PCB should have been used for the call.

A3

Explanation: The program issued an ISRT or PURG call that referenced a modifiable alternate PCB that did not have its destination set. IMS returns this status code to PURG calls only when the PURG call specified an I/O area as one of the parameters.

Programmer Response: Issue a CHNG call to set the destination of the modifiable alternate PCB, and then reissue the ISRT or PURG call.

A4

Explanation: A security violation was detected during processing of an AUTH, CHNG, or ISRT call of a SPA on a conversational response. Some of the reasons for this status code are:

- Transaction authorization is active and either RACF or a transaction authorization exit routine returned a nonzero return code.
- The user is not authorized for access to the RESOURCE name in the class requested in the AUTH call. No installation data is returned.
- No source CNT is available, which might be caused by the application program not having issued a GU.
- A program-to-program message switch is being done. In this case, the applicable authorization LTERM is based on the original message, and this authorization does not allow this function to be performed.

Programmer Response: Check the transaction code to make sure it was entered correctly. If it was, check with the person who handles security at your installation.

A5

Explanation: An ISRT or PURG call supplied an invalid parameter list. The call supplied the fourth parameter (the MOD name), but the ISRT or PURG being issued was not for the first segment of an output message.

Programmer Response: Correct the ISRT or PURG call.

A6

Explanation: For a message processing program (MPP or BMP), IMS ignored a message ISRT call because the length of the message segment being inserted exceeds the size specified in the SEGSIZE keyword of the TRANSACT macro. For a Fast Path program (IFP), the length of the output message to a Fast Path terminal exceeds the size specified in the FPBUF parm of the TERMINAL macro.

Programmer Response: Correct the output message segment.

A7

Explanation: IMS ignored a message ISRT call for one of the following reasons:

- The number of message segments inserted exceeds the number specified in the SEGNO keyword of the TRANSACT macro.
- The IMS queue manager or user Queue Manager Space Notification exit routine (DFSQSPC0) prohibited the insert in order to prevent the message queue data sets from overflowing.
- The IMS queue manager or user Queue Manager Space Notification exit routine (DFSQSPC0) prohibited the insert because the destination TRANSACTION or LTERM was stopped.

Programmer Response: Check the output messages and correct them. Use ROLB or another method to free buffer space.

A8

Explanation: IMS ignored an ISRT call because:

- An ISRT call to an alternate response PCB must not follow an ISRT call to the I/O PCB.
- An ISRT call to the I/O PCB must not follow an ISRT call to an alternate response PCB.

Programmer Response: Correct the ISRT call.

A9

Explanation: IMS ignored the ISRT call because:

- The ISRT call referenced an alternate response PCB defined as SAMETRM=YES, but the PCB represented a logical terminal that is not part of the originating physical terminal. An alternate response PCB defined as SAMETRM=YES must represent the

same physical terminal as the physical terminal associated with the originating logical terminal.
- The originating terminal is in response mode, and the alternate response PCB is not associated with that logical terminal.

IMS does not return this status code if the program makes either of these errors while communicating with a terminal in a remote IMS system through MSC.

Programmer Response: Determine whether the application program is in error, the output logical terminal has been incorrectly reassigned (using the /ASSIGN command), or if SAMETRM=YES should not have been specified for the alternate response PCB.

BA

Explanation: The request was not completed because it required access to unavailable data.

Only the updates done for the current request, prior to the time it encountered the unavailable data, are backed out. The state of the database is as it was before the failing request was issued. If the request was `REPL` or `DLET`, the PCB position was unchanged. If the request was a Get or `ISRT` request, the PCB position is unpredictable.

For a DEDB, this status code might be returned if no updates have been made by the last commit point. If updates have been made by the current call since the last commit point, a BB status code is returned instead. If changes have been made by a previous call, the application program must decide whether to commit these changes.

Rather than having an abnormal termination occur, this status code is returned to the application program that issued the EXEC DLI command.

Programmer Response: This is an information-only status code.

BB

Explanation: The BB status code is the same as BA except that all database updates that the program made since the last commit point are backed out, and all nonexpress messages sent since the last commit point are canceled. The PCB position for all PCBs is at the start of the database.

For a DEDB, this status code might be returned if updates have been made by the current call.

Rather than having an abnormal termination occur, this status code is returned to the application program that issued the EXEC DLI command.

Programmer Response: This is an information-only status code.

BC

Explanation: The response from an INIT STATUS GROUPB call was not completed because it required access to unavailable data.

All database resources that were allotted up to the last commit point are backed out, with the exception of GSAM and DB2. All output messages are backed out to the last commit point. Input messages are requeued.

Programmer Response: This is an information-only status code.

BJ

Explanation: All of the databases in the PSB are unavailable, or there are no database PCBs in the PSB.

Each PCB (excluding GSAM) received an NA status code as the result of the INQY DBQUERY call.

Programmer Response: This is an information-only status code.

BK

Explanation: At least one of the databases included in the PSB is unavailable or has limited availability.

At least one PCB received an NA or NU status code as the result of processing the INQY DBQUERY call.

Programmer Response: This is an information-only status code.

CA

Explanation: The program issued a `CMD` call with an invalid command verb, or the command verb does not apply to the IMS system that the program is running in. IMS does not return any command responses.

Programmer Response: Correct the command in the `CMD` call.

CB

Explanation: The command entered in the `CMD` call is not allowed from an AOI program. IMS does not return any command responses.

Programmer Response: Correct the command. For a list of the commands that an AOI program can issue, see *IMS/ESA Customization Guide*.

CC

Explanation: IMS has executed the command and returned one or more command responses.

Programmer Response: Your program should issue `GCMD` calls as necessary to retrieve the responses.

CD

Explanation: The command that was entered on the CMD call violates security, or the application program is not authorized to issue CMD calls. IMS does not execute the command or return any command responses.

Programmer Response: Correct the command. If necessary, check with the person in charge of security at your installation to find out why your program is restricted from using that command.

CE

Explanation: IMS rescheduled the message that this GU call retrieved since the last CMD call. The program had not reached a commit point when the message was rescheduled.

Programmer Response: This is an information-only status code.

CF

Explanation: The message being returned on the GU call was received by IMS before the start of this IMS execution. CF can be received on a CHKP call when an I/O area is specified for an MPP or message-oriented BMP. This occurs when a CHKP call issues an internal GU call.

Programmer Response: This is an information-only status code.

CG

Explanation: The message retrieved by this GU originated from an AOI exit routine.

Programmer Response: This is an information-only status code.

CH

Explanation: IMS ignored the CMD call just issued because the AOI command interface detected a system error and was unable to process the command. IMS processing continues.

Programmer Response: Reissue the command.

CI

Explanation: CI is a combination of CE and CF. The message retrieved by this GU was scheduled for transmission before IMS was last started. The message was rescheduled, but the program had not reached a commit point.

Programmer Response: This is an information-only status code.

CJ

Explanation: CJ is a combination of CE and CG. The message retrieved by this GU was scheduled for transmission before IMS was last started. The message originated from an AOI exit routine.

Programmer Response: This is an information-only status code.

CK

Explanation: CK is a combination of CF and CG. The message retrieved with this GU originated from an AOI user exit. The message was scheduled for transmission before IMS was last started.

Programmer Response: This is an information-only status code.

CL

Explanation: CL is a combination of CE, CF, and CG. Please see the explanations of those codes.

Programmer Response: This is an information-only status code.

CM

Explanation: The command that was entered on the CMD call has been executed and completed, but it resulted in an exception response that could not be built because of an insufficient amount of general work area (WKAP).

Programmer Response: Increase WKAP if you want retrieval of the response.

CN

Explanation: The IOASIZE= parameter that was specified on the PSBGEN macro is defined for less than the required minimum for CMD calls (132 bytes).

Programmer Response: Redefine IOASIZE= parameter on the PSBGEN for a minimum of 132 bytes.

DA

Explanation: The program issued a DLET or REPL that tried to modify the key field in the segment or, when using field-level sensitivity, a REPL call tried to modify a field that had REPL=NO specified on the SENFLD STMT in the PSB. You cannot change a segment's key field.

Programmer Response: Correct the request.

DJ

Explanation: The program issued a DLET or REPL call that was rejected because the segment was not currently in hold status. Some possible reasons for this status code are:

- The segment had not been previously retrieved with a Get Hold call.
- The segment was already deleted using this PCB. After one Get Hold call, multiple REPL calls or a DLET call following a REPL call are valid, but multiple DLET calls are not.
- The segment was obtained using a secondary index as the processing sequence. A subsequent DLET or REPL call using either this PCB or another PCB within the PSB caused the current secondary index entry for this PCB to be deleted.
- A checkpoint call was issued following the Get Hold call and preceding the REPL or DLET call.
- A rollback call was issued following the get hold call and preceding the REPL or DLET call.

Programmer Response: Correct the program so that the segment is in hold status when a DLET or REPL is issued.

DX

Explanation: The program issued a DLET that violates the delete rule for that segment.

Programmer Response: Check the program to see whether or not the program should delete that segment; if it should, check with your DBA (or the equivalent specialist at your installation) to find out what delete rule has been specified for that segment.

EM

Explanation: The EM status (empty area) indicates that there are no valid sequential dependent segments in the area.

Programmer Response: Check to see that the correct area is being processed by the utility and that sequential dependent segments have been inserted.

FA

Explanation: IMS returns this status code when the program reaches a commit point and an arithmetic overflow occurs in an MSDB, DEDB, or VSO DEDB during that commit interval since the last commit point (or, if the program had not reached a commit point, since the program began processing). You can receive this status code on a SYNC call, a CHKP call, or a GU call to the message queue, depending on your program. The overflow occurred after the program issued a FLD/CHANGE call, or a REPL call for the MSDB, DEDB, or VSO DEDB. When this happens, IMS issues an internal ROLB call to eliminate the changes that the program has made since the last commit point. All database positioning is lost.

Programmer Response: Reprocess the transaction.

FC

Explanation: The program issued a request that is not valid for the segment type.

Programmer Response: Correct the request.

FD

Explanation: A nonmessage driven BMP reached a deadlock when IMS tried to get DEDB or MSDB resources (either DEDB UOWs or overflow latches) for the program. Or, a mixed-mode BMP reached a deadlock on any resource, either Fast Path or full function. IMS eliminates all database updates that the program has made since the last SYNC call, CHKP request, or SYMCHKP command (or since the program started processing, if the program has not issued a SYNC call or CHKP request). All database positioning is lost, unless you specified the P processing option in the PCB. Messages to a non-express alternate TP PCB are discarded.

Rather than having an abnormal termination occur, this status code is returned to the application program that issued the EXEC DLI command.

Programmer Response: Start processing from the last commit point. If you reach a deadlock again, terminate the program.

FE

Explanation: IMS returns this status code any time a program issues a FLD call that receives a nonblank status code in the FSA.

Programmer Response: See “Fast Path Databases” in *IMS/ESA Application Programming: Database Manager* for an explanation of FSA status codes and correct the FLD call.

FF

Explanation: A program issued an ISRT call against an MSDB that has no free space. If IMS determines that there is no free space when the program issues the ISRT call, the program receives the FF status code for that call. IMS might not determine this until the program reaches the next commit point. In this case, IMS returns FF when the program issues a GU call to the message queue, a SYNC call, or a CHKP call, depending on which call caused the commit point.

Programmer Response: To avoid this situation, specify more space for the MSDB at the next system start (cold start or normal restart).

FG

Explanation: FG is a combination of FE and FW. A batch-oriented BMP issued a FLD call that received a nonblank status code in the FSA, and the program has depleted its normal buffer allocation.

Programmer Response: Check the FSA status code and correct the FLD call, and then issue SYNC or CHKP calls in the program more frequently. One way to handle this status code is to branch to an error routine that causes the program to issue SYNC or CHKP calls more frequently when it receives this status code.

FH

Explanation: A DEDB area was inaccessible to the requested service when the program issued a database request or when the program reached a commit point. The AREA was stopped or the DEDB randomizing routine was not loaded into storage. A /START DATABASE dedbname command will cause the DEDB randomizing routine to be reloaded.

If IMS returns this status code on a call that caused a commit point to occur (a SYNC call, a message GU, a CHKP request, or a SYMCHKP command), IMS issues an internal ROLB call to eliminate the program's database updates and message output created since the last commit point.

If your program is accessing a DEDB in a data-sharing environment, and if the authorization fails when your program issues its first DL/I call to the DEDB, Fast Path returned this status code. Fast Path also notified the master terminal operator of the authorization failure. Your position in the database is before the first root in the next area. A GN will get the next available record (unless that one is also inaccessible).

If a program has access to an area through a PCB with PROCOPT=H and another PCB without PROCOPT=H, it is possible that only calls to the PCB with PROCOPT=H will receive the FH status code. This is because the area is accessible to IMS, but the required HSSP (high-speed sequential processing) setup could not be established. Message DFS0533A explains the reason for this occurrence and is sent to the job log. This status code is also returned if the PROCOPT for one PCB is more restrictive than the PROCOPT of a different PCB in the same PSB. Position is set to the beginning of the next accessible area.

Rather than having an abnormal termination occur, this status code is returned to the application program that issued the EXEC DLI command.

Programmer Response: If the data in the area is important, contact the DBA. If the data in the area is unimportant, the program should roll back the changes. Your program can continue processing with the next available area.

If the status code is related to an HSSP setup problem, fix the error as described in the message DFS0533A in the job log.

FI

Explanation: The program's I/O area is not at a storage address that the program can access.

Programmer Response: Correct the program.

FM

Explanation: The application program issued a request for which the randomizing routine returned a return code of 4.

Rather than having an abnormal termination occur, this status code is returned to the application program that issued the EXEC DLI command.

Programmer Response: The database position has not changed. The application program must determine subsequent processing.

FN

Explanation: The program issued a FLD call that contains a field name in the FSA that is not defined in the DBD. IMS does not continue processing the FLD call or any of the FSAs in the FLD call. IMS returns an FN status code in this situation even if an earlier FSA in the same FLD call received an FE status code.

Programmer Response: Issue a ROLB call to remove the effects of the incorrect FLD call and then correct the FLD call.

FP

Explanation: The I/O area referenced by a REPL, ISRT, or FLD/CHANGE call to an MSDB contains an invalid packed-decimal or hexadecimal field.

Programmer Response: Correct the data in the I/O area.

FR

Explanation: One of the following situations exists:

- A batch-oriented BMP issued a database request that forced the system to go beyond the buffer limit specified for the region.
- A batch-oriented BMP received a GC status code in a PCB with PROCOPT=H. Another commit process was required before using the PCB again.

IMS eliminates all database changes made by the program since the last SYNC call, CHKP request, or SYMCHKP command the program issued (or since the program started processing if the program has not issued any SYNC calls, CHKP requests, or SYMCHKP commands). All database positions for PCBs not referring to a DEDB with PROCOPT=P or H active are

lost. If the PCB referred to a DEDB with PROCOPT=P or H active, the position is set to the valid position after the last commit process, or the start of the valid range if there was no commit process.

Rather than having an abnormal termination occur, this status code is returned to the application program that issued the EXEC DLI command.

Programmer Response: Either terminate the program and restart it with a larger buffer allocation, or provide a routine that causes frequent commit points. If PROCOPT=H is used, make sure that a commit point is requested after a GC code has been returned.

FS

Explanation: For a root segment or direct dependent segment, this status code is returned only to BMPs. For a sequential dependent segment, this status code can be returned to either a BMP or a message-driven program:

- A BMP issued an ISRT request for a root segment, a direct dependent segment, or a sequential dependent segment, but IMS could not get enough space in either the root-addressable or sequential dependent part of the DEDB area to insert the new segment:
  - If IMS returns this status code on an ISRT request for a root segment, a direct dependent segment, or a sequential dependent segment, the problem is with the root-addressable portion of the area, the independent overflow area, or the sequential dependent area.
  - If IMS returns this status code when the program issues a SYNC call, CHKP request, or SYMCHKP command, the problem is with the sequential dependent part of the area.

  In either case, IMS eliminates all of the database changes the program has made since the last commit point (or since the program started processing, if the program has not reached a commit point).
- A message-driven program issued an ISRT request for a sequential dependent segment, and the sequential dependent part is full.

Rather than having an abnormal termination occur, this status code is returned to the application program that issued the EXEC DLI command.

Programmer Response: Continue with other processing, and report the problem to the system programmer.

FT

Explanation: The Fast Path program issued a call to a Fast Path database that included too many SSAs. A call to a DEDB can include up to 15 SSAs. A call to an MSDB can include only one SSA.

Programmer Response: Correct the call.

FV

Explanation: At least one of the verify operations in a FLD call issued in a batch-oriented BMP failed when the program reached a commit point. IMS eliminates the database updates the program has made since it issued the last SYNC or CHKP call (or if the program has not issued a SYNC or CHKP call, since the program started processing). All database positioning is lost.

Programmer Response: Reprocess the transaction or terminate the program.

FW

Explanation: A BMP has used all buffers that are allocated for normal use, or all buffers have been modified. IMS returns this status code to warn you that you might be running out of buffer space. An FR status code might be imminent.

If you have been processing a DEDB, you get FW for requests that change data.

If you have been processing an MSDB, you get FW for all calls that change data and for GH calls.

With a DEQ call, you receive this code if no buffers can be released.

Rather than having an abnormal termination occur, this status code is returned to the application program that issued the EXEC DLI command.

Programmer Response: You can supply an error-handling routine, triggered by the FW status code, that will cause your program to issue SYNC calls, CHKP requests, or SYMCHKP commands as soon as an FW status code is returned to your program. This reduces the total buffer requirement. To avoid receiving the FW status code, issue SYNC or CHKP calls more frequently.

FY

Explanation: PROCOPT=H PCBs process segments sequentially in the forward direction. Position is established on a UOW and is moved forward only. Attempts to retrieve segments prior to the current UOW position are not allowed for HSSP application programs and will not be processed; they receive this status code.

Programmer Response: Change the application program to retrieve segments in a forward direction only; use a PCB with a PROCOPT value other than H to access the segments in the backward direction.

GA

Explanation: In trying to satisfy an unqualified GN or GNP call, IMS crossed a hierarchic boundary into a higher level.

Status Code Explanations

If IMS returns GA after a STAT[6] request, it means that the request that was just issued retrieved the total statistics for all the VSAM buffer subpools.

Rather than having an abnormal termination occur, this status code is returned to the application program that issued the EXEC DLI command. This call results in a return code of 0.

Programmer Response: This status code is an information-only status code.

GB

Explanation: In trying to satisfy a GN call, DL/I reached the end of the database or, if you used a SETR statement, the end of the current range. In this situation, the SSA for the call or qualification for the command specified data beyond the last occurrence, and the search was not limited to the presence of a known or expected segment occurrence.

For example, a GN call was specified for a key greater than a particular value, rather than a GU call specifying a key value beyond the highest value.

A GB status code can be returned for:

- An unqualified GN call
- A qualified GN call without a maximum key (if no data is returned to the I/O area)

In contrast, a GE status code, instead of a GB status code, can be returned for:

- A GU call
- A qualified GN call without a maximum key (if data is returned to the I/O area)
- A qualified GN call with a maximum key

IMS also returns this status code when it has closed a GSAM data set. The assumed position for a subsequent request for a GSAM or full-function database is the beginning of the database, or if a SETR statement was used for a DEDB database, the beginning of the current range.

Rather than having an abnormal termination occur, this status code is returned to the application program that issued the EXEC DLI command.

Programmer Response: User determined.

GC

Explanation: An attempt was made to cross a unit-of-work (UOW) boundary, or an area boundary in the case of PROCOPT=H. For a batch-oriented BMP PCB with PROCOPT=H or PROCOPT=P, at least one call on the referenced PCB changed position in the database since the last commit process or after the program began executing. IMS did not retrieve or insert a segment. Position is before the first segment of the following UOW.

Rather than having an abnormal termination occur, this status code is returned to the application program that issued the EXEC DLI command.

Programmer Response: User determined. However, if the GC status code results from a call that referred to a PCB with PROCOPT=H, the program must cause a commit process before any other call can be issued for that PCB. Failure to do so results in an FR status code.

GD

Explanation: The program issued an ISRT call that was not qualified for all levels above the level of the segment being inserted. For at least one of the levels for which no qualification was specified, a prior request using this PCB established valid position on a segment. That position is no longer valid for one of these reasons:

- The segment has been deleted by a DLET call using a different DB PCB.
- The segment was retrieved using an alternate processing sequence, and a REPL or DLET call for this DB PCB caused the index for the existing position to be deleted.

Rather than having an abnormal termination occur, this status code is returned to the application program that issued the EXEC DLI command.

Programmer Response: User determined.

GE

Explanation:

For call-level programs:

IMS returns this status code when:

- DL/I is unable to find a segment that satisfies the segment search argument described in a Get call.
- For an ISRT call, DL/I cannot find one of the parents of the segment being inserted.
- For an ISRT call, DL/I was requested to insert a root segment outside of the accessible range determined by a SETR statement.
- The program issued a STAT[7] call for OSAM buffer pool statistics, but the buffer pool does not exist.
- The program issued a STAT[7] call for VSAM buffer subpool statistics, but the subpools do not exist.
- A nonmessage driven BMP issued a FLD call to an MSDB segment. After the FLD call but before a commit point, the MSDB segment was deleted. GE can be returned for this reason after either a SYNC or a CHKP call.

---

6. STAT is a Product-sensitive programming interface.

7. STAT is a Product-sensitive programming interface.

For command-level programs:

- DL/I is unable to find a segment that satisfies the segment described in a Get command.
- For an ISRT command, DL/I cannot find one of the parents of the segment you're inserting.
- The program issued a STAT[7] command for ISAM or OSAM buffer pool statistics, but the buffer pool does not exist.
- The program issued a STAT call for [7] VSAM buffer subpool statistics, but the subpools do not exist.

Rather than having an abnormal termination occur, this status code is returned to the application program that issued the EXEC DLI command.

Programmer Response: The action you take depends on your program.

Note: When a GNP call for a DEDB sequential dependent segment results in a GE status code, the I/O area contains a length indication of 10 bytes and the original position of the deleted portion of the sequential dependent part. Position is at the end of the sequential dependent chain.

GG

Explanation: DL/I returns this status code if the segment being retrieved contains an invalid pointer and the application program has a processing option of GOT or GON. (Processing options are explained under PROCOPT in the discussion of program specification block generation in *IMS/ESA Utilities Reference: System*.) This can occur when update activity in the database is going on concurrently with your program's processing.

For call-level programs:

The PCB key feedback length and area will be based on the last segment that satisfied the call. Your position is at the beginning of the database.

For command-level programs:

If your request specified KEYFEEDBACK, the DIBKFBL will contain the length of the key of the last segment that satisfied the command. Your position is at the beginning of the database.

Rather than having an abnormal termination occur, this status code is returned to the application program that issued the EXEC DLI command.

Programmer Response: Continue processing with another segment or terminate the program. The request that resulted in the GG status code might be successful if you issue it again.

GK

Explanation: DL/I has returned a different segment type at the same hierarchic level for an unqualified GN or GNP call.

Rather than having an abnormal termination occur, this status code is returned to the application program that issued the EXEC DLI command. This call results in a return and reason code of 0.

Programmer Response: This is an information-only status code.

GL

Explanation: For either call-level or command-level programs:

The program issued a LOG request that contained an invalid log code for user log records. The log code in a LOG request must be equal to or greater than X'A0'.

For call-level programs:

DL/I returns GL on a DEQ request when the first byte of the I/O area referenced in the request did not contain a valid DEQ class (B-J).

For command-level programs:

EXECDLI returns a GL status for either a GN, GNP, GU, or DEQ command when the alphabetic character coded on the LOCKCLASS option is not within the range of B to J. An ABENDU1041 is then issued.

Programmer Response: Correct the log code, which is the first byte of the log message.

For call-level programs:

If the program received this status code for a DEQ request, check the DEQ class code in the I/O area.

For command-level programs:

Check the alphabetic character coded for class on the LOCKCLASS option to ensure that it is in the range from B to J.

GP

Explanation: The program issued a GNP when there is no parentage established, or the segment level specified in the GNP is not lower than the level of the established parent.

Programmer Response: Make sure you have established parentage before issuing GNP**,** and check the segment level specified by the GNP.

HT

Explanation: The HT status indicates that the High-Water-Mark time stamp (HWM TS) is less than the Logical Begin time stamp (LB TS).

Programmer Response: The time stamp in the High-Water-Mark segment was not updated on the area data set during utility setup and partner notification. Check whether the data-sharing partner is still running. The RLM may have a lock for the sequential dependent CI.

II

Explanation: The program issued an ISRT call that tried to insert a segment that already exists in the database. Current position after an II status code is just before the duplicate of the segment you tried to insert. Some of the reasons for receiving this status code are:

- A segment with an equal physical twin sequence field already exists for the parent.
- A segment with an equal logical twin sequence already exists for the parent.
- The logical parent has a logical child pointer, the logical child does not have a logical twin pointer, and the segment being inserted is the second logical child for that logical parent.
- The segment type does not have physical twin forward pointers and the segment being inserted is the second segment of this type for that parent, or it is the second HDAM root for one anchor point.
- The segment being inserted is in an inverted structure. (The immediate parent of this segment in the logical structure is actually its physical child in the physical structure.)
- A physically paired logical child segment already exists with a sequence field equal to that of the segment you're inserting. For example, the segment could have been inserted with no duplication, but when an attempt was made to position for the insert of its physical pair, the segment had a duplicate key to an existing twin segment.
- An application program inserted a key of X'X'FF...FF'' into a HISAM or HIDAM database.

Rather than having an abnormal termination occur, this status code is returned to the application program that issued the EXEC DLI command.

Programmer Response: User determined.

IX

Explanation: The program issued an ISRT call that violated the insert rule for that segment. Some of the reasons that IMS returns this status code are:

- The program tried to insert the logical child and logical parent, and the insert rule for the logical parent is physical and the logical parent does not exist.
- The program tried to insert the logical child and the logical parent, and the insert rule is logical or virtual and the logical parent does not exist. In the I/O area, the key of the logical parent does not match the corresponding key in the concatenated key in the logical child.
- The program tried to insert a logical child, and the insert rule of the logical parent is virtual and the logical parent exists. In the I/O area, the key in the logical parent segment does not match the corresponding key in the concatenated key in the logical child.
- The program tried to insert a physically paired segment, where both sides of the physical pair are the same segment type and the physical and logical parent are the same occurrence.
- The program issued an ISRT call to a GSAM database after receiving an AI or AO status code.

Programmer Response: Correct the ISRT or the program.

LB

Explanation: The segment that the program tried to load already exists in the database. Other possible causes are:

- A segment with an equal physical twin sequence field already exists for the parent.
- A segment type does not have a physical twin forward pointer, and the segment being inserted is either the second segment of this segment type for the parent or the second HDAM root for one anchor point.
- An application program inserted a key of X'FF...FF' into a HISAM or HIDAM database.

Rather than having an abnormal termination occur, this status code is returned to the application program that issued the EXEC DLI command.

Programmer Response: Correct the ISRT call or LOAD command, or find out if the load sequence is incorrect. Check with the DBA or the equivalent specialist at your installation.

LC

Explanation: The key field of the segment being loaded is out of sequence.

Programmer Response: Check the segment and determine where it should be loaded.

LD

Explanation: No parent has been loaded for the segment being loaded.

Programmer Response: Check the sequence of segments that have been loaded and determine where the parent should have been loaded.

LE

Explanation: The sequence of sibling segments being loaded is not the same as the sequence that is defined in the DBD.

Programmer Response: Check and correct the sequence of the segments that are being loaded.

LS

Explanation: The LS status means that an application program needed to allocate SDEP CI RBAs to contain the application programs' insert activity for a particular area in a Data Entry Database and the CIs could not be locked by the RLM. The application work may be committed, but some other application work may not have enough CI space, depending on how much SDEP insert work was done and the first committed application.

Programmer Response: Do a commit and be careful not to insert too many more SDEP segments.

MR

Explanation: Reserved.

NA

Explanation: The `INIT` call with DBQUERY in the I/O area or the QUERY command was issued, and at least one of the databases that could be accessed using this PCB was not available.

A request made using this PCB will probably result in a BA status code if the `INIT` STATUS GROUPA has been issued or in a DFS3303I message and 3303 pseudo abend if it has not. An exception is when the database is not available because dynamic allocation failed. In this case, a request results in an AI (unable to open) status code.

Programmer Response: This is an information-only status code.

NE

Explanation: Indexing maintenance issued a DL/I call, and the segment has not been found. This status code is included in message DFS0840I. The system console receives message DFS0840I `INDEX ERROR` (*dbdname*) `NE` (first *45* bytes of key). The application program receives a blank status code.

An application program could have processed a secondary index as a database and thus deleted some of the secondary index entries. Subsequently, when a source segment is deleted, the secondary index for the source statement might not be present. For this reason, when the application program deletes a source segment and the index entry is not present, the DFS0840I message is sent to the system console, but a blank status code is returned to the application program.

Programmer Response: Determine whether the secondary index has been processed as a database and, as a result, the key included in the DFS0840I message was deleted. If this is not the case, check the cause for the index inconsistency with the database and correct it.

NI

Explanation: There is a duplicate segment in a unique secondary index. While IMS was inserting or replacing a source segment for a secondary index defined with a unique sequence field, the insertion of the secondary index segment was attempted but was unsuccessful because an index segment with the same key was found. One possible cause for a duplicate segment in the index is that the index DBD incorrectly specified a unique key value—secondary index only.

In an online application program, the call is backed out and the program receives an NI status code.

For a batch program that does not log to the IMS DASD log, IMS abnormally terminates the program with a U0828 abend. You should run batch backout.

Programmer Response: The response is determined by the user. If duplicate secondary index entries occur, specify the index as nonunique, and provide an overflow entry-sequenced data set.

NL

Explanation: The application program issued an extended checkpoint call. Checkpoint information is written to the log data set, but there is no DD statement in the batch step for the log, so no checkpoint was written. The DD name for the log data set is IEFRDER. Although no checkpoint information was written, normal commit processing was performed.

Programmer Response: Provide an IEFRDER DD statement. No status is returned for a DD DUMMY statement.

NO

Explanation: A BSAM or VSAM physical I/O error occurred during a database request that is issued by the index maintenance function.

For an online program, all updates made for the call are backed out and the application program receives the NO status code. For a batch program that does not log to the IMS DASD log, IMS abnormally terminates the program with an 826 abend.

Programmer Response: See accompanying messages giving details of the error. In a batch environment, run batch backout.

NU

Explanation: An ISRT, DLET, or REPL request using this PCB might result in a BA status code if the INIT STATUS GROUPA call or QUERY command has been issued or in a DFS3303I message or 3303 pseudo abend if it has not. If the unavailable database is only required for delete processing, it is possible that the ISRT and REPL requests can be processed.

Programmer Response: This is an information-only status code.

OS

Explanation: The OS status indicates that the STOPRBA parameter value given for the DEDB Sequential Dependent Scan Utility is too large for the current sequential dependent CI set.

Programmer Response: Check the parameter value for validity and use a correct value or use the utilities default value for the scan end.

QC

Explanation: There are no more messages in the queue for the program. The reasons that IMS returns this status code are:

- An application program issued a successful CHKP call, but the message GU call issued internally by the CHKP call was unsuccessful (that is, it did not return a message).
- An application program processing APPC synchronous messages that does not set sync points for each message GU call (that is, mode=MULT on the TRANSACT macro) is returned a QC status code to force a sync point after each GU call.

  For more information regarding the TRANSACT macro, refer to *IMS/ESA Installation Volume 2: System Definition and Tailoring*.
- An MPP or transaction-oriented BMP issued a GU call to retrieve another message, but either there are no more messages or the processing limit (that is, PROCLIM=parm on the TRANSACT macro) has been reached.
- IMS is shutting down or:
  - A /PSTOP REGION command has been issued for the dependent region in which the application program is processing.
  - A database dump (/DBD) command has been issued.
  - A database recovery (/DBR) command is in operation.
  - A stop subsystem (/STOP SUBSYS) command has been issued.

    For more information regarding these commands, refer to *IMS/ESA Operator's Reference*.
- IMS wants to reschedule the region (quick reschedule). For more information regarding quick reschedule, refer to *IMS/ESA Administration Guide: System*.

Programmer Response: This is an information-only status code. The application program should terminate.

QD

Explanation: The program issued a message GN or GCMD call, but there are no more segments for this message.

Programmer Response: Process the message.

QE

Explanation: The program issued a message GN call before issuing a GU call to the message queue. In message-driven Fast Path programs, this code applies to message calls only. The program issued a message GN call before issuing a GU call to the message queue. In message-driven Fast Path programs, this code applies to message calls only. This code is also returned when a program issues a ROLB call, specifying the I/O area parameter, without having issued a successful message GU call during that commit interval. A message-driven Fast Path program issued a CHKP call to establish an internal GU call but the CHKP call failed with a status QC code. A successful GU call was never issued during the commit interval for the failing CHKP call. Information only status code for calls encountering SDEP full or FLD verify failures which are reprocessed via ROLB.

Programmer Response: Correct the program by:

- Issuing a GU call before the GN call
- Issuing a CMD call before the GCMD call
- Issuing a GU call before the ROLB call

QF

Explanation: The length of the segment is less than five characters. The minimum length allowed is the length of the message text plus four control characters.

Programmer Response: Correct the segment.

QH

Explanation: There has been a terminal symbolic error. The output logical terminal name or transaction code is unknown to IMS. Some reasons for receiving this status code are:

- The program tried to insert an alternate response PCB receiving a QC status code for a GU call.
- The program tried to insert to an I/O PCB that has a logical terminal name of blanks. This could occur after the program issued a GU call for a message that originated either from a batch-oriented BMP or a CPI Communications driven program.

- SMB or CNT could not be found.
- The program deallocated a conversation with a SETO call with the DEALLOCATE_ABEND option. Any subsequent ISRT calls are rejected with this status code.
- The program issued an ISRT call without first issuing a GU call.
- The logical terminal name or transaction code specified is Fast Path exclusive and is not available to this program.
- The program issued an ISRT call for a segment shorter than 5 bytes.
- The program issued an ISRT call for a SPA shorter than 6 bytes.
- The logical terminal name or transaction code has leading blanks, instead of being left-justified.

Programmer Response: Check the logical terminal name or transaction code, and correct it.

RA

Explanation: The token does not match a token for any outstanding SETS requests or the request was issued for a database PCB that did not get a BA status on the previous request.

Programmer Response: The outstanding SETS request might have been canceled by a commit process, or an error exists in the use of the token.

RC

Explanation: The ROLS call was issued with unsupported PCBs in the PSB, or the program is using an attached subsystem. If the ROLS call is in response to a SETS call, the call is rejected. If the ROLS call is in response to a SETU call, the call is processed, but updates to unsupported PCBs or an attached subsystem are not backed out. This status is only returned for a ROLS call in response to a SETU call if an attached subsystem is being used.

Programmer Response: User determined.

Explanation: The ROLS request was rejected because the PSB contains access to a DEDB, MSDB, or GSAM organization or has access to an attached subsystem.

Programmer Response: The ROLS request is invalid in this environment. The program must either remove the use of the database organization that is preventing the use of the ROLS call or not use the ROLS call.

RX

Explanation: The program issued a REPL that violated the replace rule for that segment.

Programmer Response: Correct the REPL call, or check with the DBA or the equivalent specialist at your installation.

SA

Explanation: On a SETS request, IMS was not able to obtain the storage space for the data in the I/O area.

Programmer Response: Use a larger region size for the job step.

SB

Explanation: The maximum number of levels, nine, of SETS requests were already specified, and this request is attempting to set the tenth.

Programmer Response: Correct the program.

SC

Explanation: A SETS or SETU call was issued with unsupported PCBs (DEDB, MSDB, or GSAM) in the PSB, or the program is using an attached subsystem.

Programmer Response: For a SETS call, the request is rejected. Remove the unsupported PCBs or use the SETU call. For a SETU call, the program can proceed with the knowledge that a ROLS call will not back out changes for the unsupported PCBs. The other option is to not use the SETS or ROLS function.

TA

Explanation: This status code applies to CICS online command-level programs only, and it is returned following a scheduling request. The PSB named in the request is not in the PSB directory.

Programmer Response: Correct the name of the PSB in the scheduling request, or add the PSB name to the PSB directory.

TC

Explanation: This status code applies to CICS online command-level programs only, and it is returned following a scheduling request. It means that you have already scheduled a PSB.

Programmer Response: Correct your program so that you terminate a PSB before scheduling another. If you want to reschedule a PSB, you must have already terminated the PSB.

TE

Explanation: This status code applies to CICS online command-level programs only, and it is returned following a scheduling request. The PSB could not be scheduled because an initialization error occurred.

Programmer Response: See your system programmer or DBA. For information on possible causes for the PSB initialization error, see *IMS/ESA Application Programming: Database Manager*.

TG

Explanation: This status code applies to CICS online command-level programs only, and it is returned following a terminate request. The program issued a terminate request when there was no PSB scheduled.

Programmer Response: This is an information-only status code. If you only wanted to terminate a PSB, continue with processing. If you also wanted to cause a sync point, issue a SYNCPOINT command. (No sync point was caused by the unsuccessful terminate request.)

Rather than having an abnormal termination occur, this status code is returned to the application program that issued the EXEC DLI command.

TH

Explanation: This status code applies to CICS online command-level programs only, and it is returned following a database request or a statistics request. The program attempted to access the database before scheduling a PSB.

Programmer Response: Correct your program, and schedule a PSB before accessing the database.

TI

Explanation: This status code applies to command-level programs only, and it is returned after an `ISRT` command. The `ISRT` command defined an invalid path to the segment. Data must be transferred for all segments between the first named segment and the last named segment.

Programmer Response: Correct the ISRT command, specifying a FROM option for each segment to be transferred.

TJ

Explanation: This status code applies to CICS online command-level programs only, and it can be returned after any command that a CICS online program uses. DL/I is not active.

Programmer Response: Contact your DBA. CICS must be re-initialized with DL/I defined as active in the SIT.

TL

Explanation: This status code applies to CICS online command-level programs only, and it is returned after a scheduling request. A conflict in scheduling intent occurred. (This cannot occur if program isolation has been specified.)

Programmer Response: Specify program isolation in the SIT. If program isolation has not been specified, wait until the PSB is no longer in use, and reschedule it.

TN

Explanation: This status code applies to command-level programs only, and it can be returned after any of the commands. An invalid SDIB exists. An initialization call was not made, or the system's DIB (not the application program's DIB) was overlaid.

Programmer Response: Check your program to make sure that you did not use an entry statement, as you would in a call-level batch program. Also make sure that no addressing errors are in your program that would cause an overlay.

TO

Explanation: This status code applies to command-level programs only, and it is returned following a `REPL` command. A path-replace error occurred. The segments to be replaced are compared to the previous Get command and one of the following situations occurred:

- A segment is named to be replaced that was not retrieved by the Get command.
- Data had not been transferred (no INTO option) for this segment on the Get command.
- The attributes of the data to be transferred do not match the data in the database.

Programmer Response: Correct the program.

TP

Explanation: This status code applies to command-level programs only, and it is returned following any of the database commands, a LOAD command, or a statistics request. The number of the PCB specified in the USING option is higher than the number of PCBs in the PSB being used, or an invalid processing option was specified. For example, the program tried to issue a LOAD command without having the L processing option specified in its PSB.

An EXEC DLI command is being attempted against a GSAM PCB. This is invalid.

Programmer Response: Check the PSB and correct your program.

TR

Explanation: This status code means that the CICS XDLIPRE exit routine returned X'04' in register 15 because the routine determined that the immediately preceding DL/I request should not be executed.

Programmer Response: Contact the CICS system programmer.

TY

Explanation: This status code applies to CICS online command-level programs only, and it is returned following a database or statistics request. The database was not open when the request was issued.

Programmer Response: Contact your DBA or system programmer. The database can be checked and opened by using operator commands.

TZ

Explanation: This status code applies to CICS online command-level programs only, and it is returned following a database or statistics request. The length of the retrieved segment is greater than 64 KB.

Programmer Response: Contact your DBA or system programmer; the database definition might require modification.

U1

Explanation: This status code is returned when the area name specified is not valid.

Programmer Response: Correct the error and run the utility job again.

U9

Explanation: This status code is returned when the area access intent is read or read only. Access intent must be UP or EX.

Programmer Response: Use the /STA DB ACCESS command to set the access intent to UP or EX and run the job again.

UB

Explanation: This status code is returned when IMS is unable to obtain private buffer pool.

Programmer Response: No DFS0535I message is issued if the High Speed Reorganized Utility (HSRE) is being used when this status code is received. See the DFS2712I messages issued at utility termination for the name of the module, abend subcode, Utility High Speed Workarea (UHSW) storage area dump, IOAR (DEDB I/O), and register contents.

If the DBFPAPB0 return code is 08, storage is not available for the private buffer pool.

If the DBFHUSS0 return code is 10, the request for private buffers is for the initial buffer set and the private pool anchor address already exists.

UC

Explanation: This status code is returned for the following reasons:

- For batch programs in which a checkpoint record was written to the UCF journal data set. For information about the Utility Control Facility (UCF), see *IMS/ESA Administration Guide: Database Manager* and *IMS/ESA Utilities Reference: Database Manager*.

  During the processing of an HD reorganization, a reload, or an initial load program under the supervision of the Utility Control Facility (UCF), a checkpoint record was written to the UCF journal data set. IMS returns this status code to indicate that the last `ISRT` call was correct and the initial load program might continue or it might perform a checkpoint procedure before continuing.
- When a connect failed.

Programmer Response: This is an information-only status code for the first status code reason above.

When this status code is issued for a connect failure, see message DFS0535I for more information on how to correct the error.

UP

Explanation: This status code is returned when the UOW requested is greater than the UOW range.

Programmer Response: Correct the error and run the job again.

UR

Explanation: This status code is returned for batch programs only. Your initial load program is being restarted under UCF. For information about the Utility Control Facility (UCF), see *IMS/ESA Administration Guide: Database Manager* and *IMS/ESA Utilities Reference: Database Manager*. The program terminated while executing under UCF. The job was resubmitted with a restart request.

Programmer Response: Ensure that the program is in proper sequence with database loading. The program uses the I/O area and the DB PCB key feedback area to do this.

US

Explanation: This status code is returned for batch programs only. The initial load program is about to stop processing. While processing an HD reorganization reload or user initial load program under the supervision of UCF, the operator replied to the WTOR from UCB and requested the current function to terminate. For information about the Utility Control Facility (UCF), see *IMS/ESA Administration Guide: Database Manager* and *IMS/ESA Utilities Reference: Database Manager*. The last `ISRT` call was processed.

Programmer Response: Ensure that the initial load program performs a checkpoint procedure of its data sets and returns with a nonzero value in register 15.

UW

Explanation: This status code is returned when IMS is unable to obtain a work area.

Programmer Response: Increase the REGION size and run the job again.

UX

Explanation: This status code is returned for batch programs only. A checkpoint record was written, and processing stopped. This is a combination of UC and US status codes.

Programmer Response: See the descriptions of UC and US status codes.

V1

Explanation: The program tried to insert or replace a variable-length segment that is too long. The length of the segment must be equal to or less than the maximum length specified in the DBD. IMS also returns status code V1 when the specified minimum length cannot hold the complete sequence field of the segment type. In this situation, status code V1 results from one of three instances: processing without an edit/compression routine; processing with an edit/compression routine, but not specifying the key compression option; or coding a length field (LL) that is less than the specified minimum length. The length must be long enough to include the entire reference field; if the segment is a logical child, it must include the entire concatenated key of the logical parent and all sequence fields for the paired segment. The program tried to delete a variable-length segment. The copy of this segment in the user's I/O area contains an invalid length field.

IMS also returns this status code when an invalid record length is specified in a GSAM call.

Programmer Response: Correct the program.

V2

Explanation: This status code applies to command-level programs only, and it is returned following a database or LOAD command. The segment length is missing or invalid. The segment length must be a positive integer. For variable-length segments, it is the maximum size acceptable to the program's I/O area.

Programmer Response: Check that the program translated and compiled correctly. The value of any segment length in a path command should not exceed 32 KB, and the sum of the lengths should not exceed 64 KB.

V3

Explanation: This status code applies to command-level programs only, and it is returned following a Get or `ISRT` command. The field length is missing or invalid. The field length must be a positive integer, and it must be specified for each field in a WHERE option.

Programmer Response: Correct the program.

V4

Explanation: This status code applies to command-level programs only, and it is returned following any of the database commands or a LOAD command. The length of a variable-length segment is invalid. The LL field as provided by the program on an `ISRT` or `REPL` command, or as received in the I/O area on a Get command, exceeds the value of SEGLENGTH.

Programmer Response: Correct the program.

V5

Explanation: This status code applies to command-level programs only, and is returned following a Get, `REPL`, or `ISRT` command. The offset is invalid. The offset must be a positive integer and not greater than the segment length.

Programmer Response: Correct the program.

V6

Explanation: This status code applies to command-level programs only, and it is returned following a Get or `ISRT` command using the KEYS option. The concatenated key length is missing or invalid. The length of the concatenated key must be a positive integer.

Programmer Response: Correct the program.

V7

Explanation: This status code applies to command-level programs only, and is returned following a `STAT` command. It means that the statistics area length is either too small or invalid. The length must be a positive integer, and it must be at least 72 bytes for unformatted statistics, 120 bytes for summary statistics, and 360 bytes for formatted statistics.

Programmer Response: Correct the program.

XA

Explanation: The program tried to continue processing the conversation by passing the SPA to another program through a program-to-program

message switch after already responding to the terminal.

Programmer Response: If a response has been sent, the SPA should be returned to IMS. Correct the program.

XB

Explanation: The program has passed the SPA to another program but is trying to respond to the originating terminal.

Programmer Response: No response is allowed by a program that is passed control of the program through a program-to-program message switch.

XC

Explanation: The program inserted a message that has some bits in the Z1 field set. The Z1 field is reserved for IMS.

Programmer Response: Correct the program to prevent it from setting those bits.

XD

Explanation: IMS is terminating by a CHECKPOINT FREEZE or DUMPQ. IMS returns this code to a BMP that has issued a `CHKP` or `SYNC` call. If it is a transaction-oriented BMP, IMS does not return a message.

IMS also returns XD when a batch program issues a `SYNC` call.

Programmer Response: Terminate the program immediately. IMS will terminate the program abnormally if the program issues another call.

XE

Explanation: A program tried to insert a SPA to an alternate express PCB.

Programmer Response: Regenerate the PSB and remove the EXPRESS=YES option from the PCB, or define another non-express PCB to be used in the `ISRT` call.

XF

Explanation: IMS is ignoring the `ISRT` call for the SPA, because the specified alternate PCB had its destination set to a logical terminal but was not defined as ALTRESP=YES during PSB generation.

MSC directed routing does not support a program-to-program switch between conversational transactions.

Programmer Response: Correct the application program or change the PSB generation for that alternate PCB to specify ALTRESP=YES.

XG

Explanation: IMS ignored the `ISRT` call because the current conversation requires a fixed-length SPA, and the ISRT call was to a program with a different length or variable-length SPA, while the source IMS system was earlier than IMS 6.1. If the SPA `ISRT` on a remote system is not going back to the input terminal (IOPCB), the SPA size must be the same as the size of the current one, if the source IMS system is earlier than IMS 6.1.

Programmer Response: Correct the program or the SPA definitions.

XX

Explanation: An error occurred during GSAM initialization or during GSAM call processing. If this status code is in the GSAM PCB before the application program issued the first call, the error was detected during initialization. Possible causes are:

- Insufficient space
- Invalid DBD
- Invalid block size
- Invalid option
- Internal GSAM error

Programmer Response: A subsequent GSAM call will result in an abnormal termination of the program. The program should terminate.

X2

Explanation: The first `ISRT` call to an alternate PCB whose destination is a conversational transaction code is not for the SPA. The SPA must be inserted with the first `ISRT` call.

Programmer Response: Insert the SPA, and then reinsert the message segment.

X3

Explanation: The program modified the first 6 bytes of the SPA; the SPA is now invalid.

Programmer Response: Correct the program and restore the original bytes.

X4

Explanation: The program issued an `ISRT` call to pass the SPA to a nonconversational transaction code. It did this by referencing a PCB whose destination was set for the nonconversational transaction code. You can send the SPA only to transaction codes that are defined as conversational.

Programmer Response: Correct the ISRT call. Send only data segments.

X5

Explanation: The program issued more than one ISRT call to send the SPA to a PCB whose destination is a transaction code. Only one SPA is allowed per message.

Programmer Response: Correct the program.

X6

Explanation: An invalid transaction code name was inserted into the SPA. This will occur if the input is from LU 6.2 (APPC) or OTMA and if a dynamic control block was built for the transaction code.

Programmer Response: Correct the program to set the proper transaction code name.

X7

Explanation: The length of the SPA is incorrect. The program modified the first 6 bytes.

Programmer Response: Correct the SPA and the program.

X8

Explanation: A system or I/O error occurred in a queue attempt for the SPA on a transaction code PCB.

Programmer Response: Terminate the conversation.

---

blanks (bb)

Explanation: The call was completed.

Programmer Response: Proceed with processing.

Chapter 16. DL/I Return and Reason Codes

This chapter contains reference information on the return and reason codes. These codes are returned in the AIB when you use the AIB interface. The information is divided into two parts:

- Return and reason code tables
  - DL/I Return Codes
  - Database Call Return Codes
  - Message Call Return Codes
  - System Service Call Return Codes
- Return and reason code explanations

Return and Reason Code Tables

The tables in this section briefly explain each return and reason code, and list the calls for which these codes can be received.

If you receive the X'0900' return code, you must look at the status code in the PCB. You may also want to look at the status code if you do not receive X'0900' but have a PCB associated with the call. (For detailed explanations of the status codes, see “Chapter 15. DL/I Status Codes” on page 443.) For DL/I return codes, see Table 55, Table 56 on page 476, Table 57 on page 481, and Table 58 on page 485.

For expanded descriptions of the return and reason codes, see “DL/I Return and Reason Code Explanations” on page 491.

Table 55. DL/I Return Codes

Return Code Hex	Description
0000	Call was completed successfully.
0004	Call was completed with information.
0008	Call was completed with warning.
00F0	Reserved within IMS.
0100	Call was completed with exception.
0104	Call incomplete -- application error.
0108	Call incomplete -- system error.
010C	Call incomplete -- unavailable data.
0110	Call incomplete -- authorization error.
0900	PCB status code analysis required.

© Copyright IBM Corp. 1974, 2000

Return and Reason Codes

Table 56. Database Calls

Return Code (Hex)	Reason Code (Hex)	Database Calls	Description
0000	0000	CLSE (GSAM) DEQ GU, GHU GN, GHN GNP, GHNP DLET, REPL ISRT (LOAD) ISRT (ADD) FLD POS OPEN (GSAM)	Call was completed successfully. Proceed.
0004	0004		Segment returned to I/O area. This is the last segment.
0004	0014		No more messages are available. No segment is returned.
0004	0018		No more messages are available. No segment is returned.
0004	0048		AO application waiting for AOI message was posted due to IMS system shutdown. No message segment returned.
0004	004C		AO application waiting for AOI message was posted due to/PSTOP region command. No message segment is returned.
0100	000C		The output area was not large enough to contain all the data. Only partial data was returned.
0104	0204	CIMS TERM	No value specified for IMS DB ID in AIBRSNM2 field.
0104	0208	CLSE (GSAM) DEQ GU, GHU GN, GHN GNP, GHNP DLET, REPL ISRT (LOAD) ISRT (ADD) FLD POS OPEN (GSAM)	An invalid PCB name was passed in the AIB.

Table 56. Database Calls (continued)

Return Code (Hex)	Reason Code (Hex)	Database Calls	Description
0104	020C	CLSE (GSAM) DEQ GU, GHU GN, GHN GNP, GHNP DLET, REPL ISRT (LOAD) ISRT (ADD) FLD POS OPEN (GSAM)	No PSB scheduled. Call before APSB.
0104	0210		The output area length in the AIB was invalid. No data is returned.
0104	0218	CIMS INIT CIMS TERM	Invalid sub-function code.
0104	021C		No AOI token specified in AIB on initial GMSG call.
0104	0220		GMSG call with WAITAOI specified did not specify an AOI token.
0104	0224		AOI token specified on GMSG call contained one or more invalid characters. AOI token may contain characters A-Z, O-9, @, #, and $.
0104	0228	CIMS INIT CIMS TERM CIMS TALL	Invalid AIBLEN must be a minimum of 264 bytes.
0104	0404		Invalid function code.
0104	0440		Invalid call from non-CPI-C application.
0104	0444		Invalid from the specific environment.
0104	0448		Invalid call from CPI-C application.
0104	044C		Invalid call from DEP region type.
0104	0450		No messages to retrieve. AO user exit DFSAOE00 does not exist.
0104	048C		APSB for PSB with no DPSB.
0104	0490		DPSB with no CPI-RR commit.
0104	0494		PSB not allocated.
0104	054C		Invalid output destination.
0104	0550		DEALLOC_ABND issued for destination.
0104	060C		Invalid AUTH class name specified.

Table 56. Database Calls (continued)

Return Code (Hex)	Reason Code (Hex)	Database Calls	Description
0104	0610	CLSE (GSAM) DEQ GU, GHU GN, GHN GNP, GHNP DLET, REPL ISRT (LOAD) ISRT (ADD) FLD POS OPEN (GSAM)	Required I/O area address parameter missing. No data is returned.
0104	0618		I/O area length is longer than the PSB work area defined by the IOASIZE parameter on PSBGEN.
0104	0620		ZZ field of the I/O area is not zero.
0104	0754		The segment length is less than 5.
0104	0810		Required option list parameter is missing.
0104	0920		Queue Space Notification Exit (DFSQSPC0) determined that the application has reached its limit of data that can be inserted to the message queue.
0104	1004		Command as entered not allowed from an application.
0104	1008		Invalid command verb specified in I/O area.
0104	1010		ICMD call was not issued before RCMD call.
0108	0004		Error encountered during PC router processing.
0108	0010	CIMS INIT	Unable to obtain private storage. GETMAIN failure trying to obtain required DRA storage.
0108	001C	CIMS INIT	ATTACH of the DRA TCB failed.
0108	0014		Unable to obtain global storage.
0108	0018		Internal latch error. For GMSG call, no segment is returned.
0108	0020	CIMS INIT	Load of required DRA module failed. The name of the module is returned in AIBSNM2. Or: Load of DRA startup table (DFScccc0) failed. The table name is returned in AIBRSNM2 (only when AIBRSNM2=DBCTLID
0108	0024	CIMS INIT	OPEN of DRA RELIB failed.

Table 56. Database Calls (continued)

Return Code (Hex)	Reason Code (Hex)	Database Calls	Description
0108	002C	CIMS INIT	MVS Name/Token service failed.
0108	0030		Spool API data set CLOSE error.
0108	0034		Spool API data set deallocation (DYNALLOC) error.
0108	0038		Spool API data set allocation (DYNALLOC) error.
0108	003C		Spool API data set OPEN error.
0108	0040		Spool API WRITE to data set error.
0108	0044		Spool API create output descriptor (OUTADD) error.
0108	0048		Spool API delete output descriptor (OUTDEL) error.
0108	0060		DFSLUMIF failure.
0108	0064		DFSYFND failure.
0108	0068		DFSYLUM failure.
0108	0080		RACINIT failure - LU 6.2 originated transaction.
0108	0220		PST 62 extension was not found.
0108	020C		Invalid AWE encountered during command processing.
0108	0214	CIMS INIT	Required DFSPRP parameters not specified correctly.
0108	0224		Security UTOKEN not found.
0108	0240		Internal error. PST posted with invalid post code. No segment returned.
0108	0244		PST posted during abnormal terminate thread processing. No segment returned.
0108	0304		PSB was not found.
0108	0308		PSB authorization failure.
0108	030C		PSB permanently bad.
0108	0310		Fast Path DB stopped/locked.
0108	0314		PSB already scheduled.
0108	0318		PSB stopped/locked.
0108	031C		I/O error reading PSB/DMB.
0108	0320		PSBW/DMB/PSB pool too small.
0108	0324		Invalid option 'L' or 'LS'.
0108	0328		Fast Path buffer fix error.
0108	032C		Invalid processing intent.

Table 56. Database Calls (continued)

Return Code (Hex)	Reason Code (Hex)	Database Calls	Description
0108	0540	CIMS INIT CIMS TERM	No active communication with IMS DB.
0108	0544	CIMS INIT	RRS/MVS is not active. The CIMS INIT c all cannot be issued. This only applies when AIBRSNM2=DBCTLID.
0110	000C		Resource specified was not authorized.
0110	0020		Program not authorized to RACF.
0110	0024		Userid not authorized to RACF.
0110	0028		Program not authorized to issue the command.
0110	002C		Userid not authorized to issue the command.
0110	0030		DFSCCMD0 indicated program was not authorized to issue command.
0110	0034		DFSCCMD0 indicated userid was not authorized to issue command.
0110	0038		ICMD call not authorized from any program (AOIS=N).
0110	003C		RACF not available for authorization check (AOIS=A|R).
0110	0040		DFSCCMD0 not available for authorization check (AOIS=A|C).
0110	0050		A CPI-C driven application issued an APSB call and the userid is not authorized to RACF.
0110	0070	CIMS INIT	The DRA RESLIB is not authorized.
0900		CLSE (GSAM) DEQ GU, GHU GN, GHN GNP, GHNP DLET, REPL ISRT (LOAD) ISRT (ADD) FLD POS OPEN (GSAM)	Reason code unmodified on 0900. PCB status code analysis required.

Table 57. Message Calls

Return Code (Hex)	Reason Code (Hex)	Message Calls	Description
0000	0000	AUTH GU GN ISRT CHNG CMD GCMD PURG SETO	Call was completed successfully. Proceed.
0004	0004		Segment returned to I/O area. This is the last segment.
0004	0014		No more messages are available. No segment is returned.
0004	0018		No more messages are available. No segment is returned.
0004	0048		AO application waiting for AOI message was posted due to IMS system shutdown. No message segment returned.
0004	004C		AO application waiting for AOI message was posted due to /PSTOP region command. No message segment is returned.
0100	000C		The output area was not large enough to contain all the data. Only partial data was returned.
0104	020C	AUTH GU GN ISRT CHNG CMD GCMD PURG SETO	No PSB scheduled. Call before APSB.
0104	0208	AUTH GU GN ISRT CHNG CMD GCMD PURG SETO	An invalid PCB name was passed in the AIB.
0104	0210	AUTH	The output area length in the AIB was invalid. No data is returned.

Table 57. Message Calls (continued)

Return Code (Hex)	Reason Code (Hex)	Message Calls	Description
0104	0218		Subfunction unknown or invalid.
0104	021C		No AOI token specified in AIB on initial GMSG call.
0104	0220		GMSG call with WAITAOI specified did not specify an AOI token.
0104	0224		AOI token specified on GMSG call contained one or more invalid characters. AOI token may contain characters A-Z, O-9, @, #, and $.
0104	0404		Invalid function code.
0104	0440		Invalid call.
0104	0444		Invalid from the specific environment.
0104	0448	SETO	Invalid call from CPI-C application.
0104	044C	SETO	Invalid call from DEP region type.
0104	0450		No messages to retrieve. AO user exit DFSAOE00 does not exist.
0104	048C		APSB for PSB with no DPSB.
0104	0490		DPSB with no CPI-RR commit.
0104	0494		PSB not allocated.
0104	054C	SETO	Invalid output destination.
0104	0550	ISRT SETO	DEALLOC_ABND issued for destination.
0104	060C	AUTH	Invalid AUTH class name specified.
0104	0610	AUTH GU GN ISRT CHNG CMD GCMD PURG SETO	Required I/O area address parameter missing. No data is returned.
0104	0618		I/O area length is longer than the PSB work area defined by the IOASIZE parameter on PSBGEN.
0104	0620	AUTH	ZZ field of I/O area not zero.

Table 57. Message Calls (continued)

Return Code (Hex)	Reason Code (Hex)	Message Calls	Description
0104	0754	ISRT SETO	The segment length is less than 5.
0104	0810	SETO	Required option list parameter is missing.
0104	0920	ISRT SETO	Queue Space Notification Exit (DFSQSPC0) determined the application has reached its limit of data that can be inserted to the message queue.
0104	1004		Command as entered not allowed from an application.
0104	1008		Invalid command verb specified in I/O area.
0104	1010		ICMD call was not issued before RCMD call.
0108	0004		Error encountered during PC router processing.
0108	0010	ISRT CHNG SETO	Unable to obtain private storage.
0108	0014	GU CHNG SETO	Unable to obtain global storage.
0108	0018		Internal latch error. For GMSG call, no segment is returned.
0108	0030	PURG	Spool API data set CLOSE error.
0108	0034	PURG	Spool API data set deallocation (DYNALLOC) error.
0108	0038	ISRT	Spool API data set allocation (DYNALLOC) error.
0108	003C	ISRT	Spool API data set OPEN error.
0108	0040	ISRT	Spool API WRITE to data set error.
0108	0044	CHNG	Spool API create output descriptor (OUTADD) error.
0108	0048	CHNG	Spool API delete output descriptor (OUTDEL) error.
0108	0060	GU ISRT CHNG SETO	DFSLUMIF failure.

Table 57. Message Calls (continued)

Return Code (Hex)	Reason Code (Hex)	Message Calls	Description
0108	0064	ISRT CHNG PURG	DFSYFND failure.
0108	0068		DFSYLUM failure.
0108	0080	AUTH	RACINIT failure - LU 6.2 originated transaction.
0108	020C		Invalid AWE encountered during command processing.
0108	0220	AUTH	PST 62 extension was not found.
0108	0224	AUTH	Security UTOKEN not found.
0108	0240		Internal error, PST posted with invalid post code. No segment returned.
0108	0244		PST posted during abnormal terminate thread processing. No segment returned.
0108	0304		PSB was not found.
0108	0308		PSB authorization failure.
0108	030C		PSB permanently bad.
0108	0310		Fast Path DB stopped/locked.
0108	0314		PSB already scheduled.
0108	0318		PSB stopped/locked.
0108	031C		I/O error reading PSB/DMB.
0108	0320		PSBW/DMB/PSB pool too small.
0108	0324		Invalid option ’L’ or ’LS’.
0108	0328		Fast Path buffer fix error.
0108	032C		Invalid processing intent.
0110	000C	AUTH	Resource specified was not authorized.
0110	0020		Program not authorized to RACF.
0110	0024		Userid not authorized to RACF.
0110	0028		Program not authorized to issue the program.
0110	002C		Userid not authorized to issue the program.
0110	0030		DFSCCMD0 indicated program was not authorized to issue command.

Table 57. Message Calls (continued)

Return Code (Hex)	Reason Code (Hex)	Message Calls	Description
0110	0034		DFSCCMD0 indicated userid was not authorized to issue command.
0110	0038		ICMD call not authorized from any program (AOIS=N).
0110	003C		RACF not available for authorization check (AOIS=A|R).
0110	0040		DFSCCMD0 not available for authorization check (AOIS=A|C).
0110	0050		A CPI-C driven application issued an APSB call and the userid is not authorized to RACF.
0900		AUTH GU GN ISRT CHNG CMD GCMD PURG SETO	Reason code unmodified on 0900. PCB status code analysis required.

Table 58. System Service Calls

Return Code (Hex)	Reason Code (Hex)	System Service Calls	Description
0000	0000	APSB CHKP DPSB GMSG ICMD INIT INQY LOG RCMD ROLB ROLS SETS SETU SNAP[2] STAT[3] SYNC XRST	Call was completed successfully. Proceed.
0004	0004	GMSG ICMD RCMD	Segment returned to I/O area. This is the last segment.
0004	0014	GMSG	No more messages are available. No segment is returned.

Table 58. System Service Calls (continued)

Return Code (Hex)	Reason Code (Hex)	System Service Calls	Description
0004	0018	GMSG RCMD	No more segments are available. No segment is returned.
0004	0048	GMSG	AO application waiting for AOI message was posted due to IMS system shutdown. No message segment returned.
0004	004C	GMSG	AO application waiting for AOI message was posted due to /PSTOP region command. No message segment is returned.
0100	000C	GMSG ICMD INQY RCMD	The output area was not large enough to contain all the data. Only partial data was returned.
0104	020C	CHKP GMSG ICMD INIT INQY LOG RCMD ROLB ROLS SETS SETU SNAP[2] STAT[3] SYNC XRST	No PSB scheduled. Call before APSB.
0104	0204	APSB DPSB	No value specified for IMS DB ID in AIBRSNM2 field.
0104	0208	CHKP INIT INQY LOG ROLB ROLS SETS SETU SNAP[2] STAT[3] SYNC XRST	An invalid PCB name was passed in the AIB. (See note 4 on page 491).
0104	0210	GMSG ICMD INQY LOG RCMD SETS SETU	The output area length in the AIB was invalid. No data is returned.

Table 58. System Service Calls (continued)

Return Code (Hex)	Reason Code (Hex)	System Service Calls	Description
0104	0218	GMSG INQY	Subfunction unknown or invalid.
0104	021C	GMSG	No AOI token specified in AIB on initial GMSG call.
0104	0220	GMSG	GMSG call with WAITAOI specified did not specify an AOI token.
0104	0224	GMSG	AOI token specified on GMSG call contained one or more invalid characters. AOI token may contain characters A-Z, O-9, @, #, and $.
0104	0228	APSB DPSB	Invalid AIBLEN must be a minimum of 264 bytes.
0104	022C	APSB	MAXTHRD value reached.Cannot allocate a thread.
0104	0404	APSB DPSB GMSG ICMD RCMD	Invalid function code.
0104	0440		Invalid call.
0104	0444		Invalid call from the specific environment.
0104	0448	INQY	Invalid call from CPI-C application.
0104	044C		Invalid call from DEP region type.
0104	0450	GMSG	No messages to retrieve. AO user exit DFSAOE00 does not exist.
0104	048C	APSB	APSB for PSB with no DPSB.
0104	0490	APSB DPSB	DPSB with no CPI-RR commit.
0104	0494	DPSB	PSB not allocated.
0104	054C		Invalid output destination.
0104	0550		DEALLOC_ABND issued for destination.
0104	060C		Invalid AUTH class name specified.

Table 58. System Service Calls (continued)

Return Code (Hex)	Reason Code (Hex)	System Service Calls	Description
0104	0610	CHKP GMSG ICMD INIT INQY LOG RCMD ROLB ROLS SETS SETU SNAP[2] STAT[3] SYNC XRST	Required I/O area address parameter missing. No data is returned.
0104	0618	ICMD	I/O area length is longer than the PSB work area defined by the IOASIZE parameter on PSBGEN.
0104	0620	INQY SETS SETU	A ZZ field of I/O area not zero.
0104	0754		The segment length is less than 5.
0104	0810		Required option list parameter is missing.
0104	0920		Queue Space Notification exit (DFSQSPC0) determined the application reached the limit of data that can be inserted to the message queue.
0104	1004	ICMD	Command as entered not allowed from an application.
0104	1008	ICMD	Invalid command verb specified in I/O area.
0104	1010	RCMD	ICMD call was not issued before RCMD call.
0108	0004	GMSG ICMD RCMD	Error encountered during PC router processing.
0108	0010	GMSG ICMD RCMD	Unable to obtain private storage.
0108	0014	GMSG ICMD RCMD	Unable to obtain global storage.
0108	0018	GMSG	Internal latch error. For GMSG call, no segment is returned.
0108	001C	APSB	ATTACH of the DRA TCB failed.

Table 58. System Service Calls (continued)

Return Code (Hex)	Reason Code (Hex)	System Service Calls	Description
0108	0020	APSB	The LOAD of the required DRA module failed or startup table failed.
0108	0024	APSB	OPEN of the DRA RESLIB failed.
0108	0028	APSB	Dynamic allocation of the DRA RESLIB failed.
0108	002C	APSB	MVS Name/Token service failed.
0108	0030		Spool API data set CLOSE error.
0108	0034		Spool API data set deallocation (DYNALLOC) error.
0108	0038		Spool API data set allocation (DYNALLOC) error.
0108	003C		Spool API data set OPEN error.
0108	0040		Spool API WRITE to data set error.
0108	0044		Spool API create output descriptor (OUTADD) error.
0108	0048		Spool API delete output descriptor (OUTDEL) error.
0108	0060	INQY	DFSLUMIF failure.
0108	0064	INQY	DFSYFND failure.
0108	0068	INQY	DFSYLUM failure.
0108	0080		RACINIT failure - LU 6.2 originated transaction.
0108	020C	ICMD	Invalid AWE encountered during command processing.
0108	0214	APSB	The DRA startup table (DFScccccc0) required parameters not specified correctly.
0108	0220		PST 62 extension was not found.
0108	0224		Security UTOKEN not found.
0108	0240	GMSG	Internal error. PST posted with invalid post code. No segment returned.
0108	0244	GMSG	PST posted during abnormal terminate thread processing. No segment returned.

Table 58. System Service Calls (continued)

Return Code (Hex)	Reason Code (Hex)	System Service Calls	Description
0108	0304	APSB	PSB was not found.
0108	0308	APSB	PSB authorization failure.
0108	030C	APSB	PSB permanently bad.
0108	0310	APSB	Fast Path DB stopped/locked.
0108	0314	APSB	PSB already scheduled.
0108	0318	APSB	PSB stopped/locked.
0108	031C	APSB	I/O error reading PSB/DMB.
0108	0320	APSB	PSBW/DMB/PSB pool too small.
0108	0324	APSB	Invalid option 'L' or 'LS'.
0108	0328	APSB	Fast Path buffer fix error.
0108	032C	APSB	Invalid processing intent.
0108	0494	DPSB	No PSB allocated because no valid PSB name is in AIBRSNM1 field.
0108	0540	APSB DPSB	No active communication with IMS DB.
0108	0544	APSB	RRS/MVS is not active. APSB call cannot be issued.
0108	0548	APSB	SWITCH_CONTEXT failed.
0108	054C	APSB	Database stopped for PSB.
0108	0550	APSB	The recovery token is not unique or the OS/390 application region ID is invalid.
0108	0554	APSB DPSB	Recovery token passed is zeros.
0108	0558	APSB	SCHED failed by a /STOP REG command.
0108	0560	APSB	SCHED failed by a /STOP THREAD command.
0108	0564	APSB	SIGNON token not found.
0108	0568	APSB	Mismatch address space connection.
0110	000C		Resource specified was not authorized.
0110	0020	ICMD	Program not authorized to RACF.
0110	0024	ICMD	Userid not authorized to RACF.
0110	0028	ICMD	Program not authorized to issue the command.
0110	002C	ICMD	Userid not authorized to issue the command.

Table 58. System Service Calls (continued)

Return Code (Hex)	Reason Code (Hex)	System Service Calls	Description
0110	0030	ICMD	DFSCCMD0 indicated program was not authorized to issue command.
0110	0034	ICMD	DFSCCMD0 indicated userid was not authorized to issue command.
0110	0038	ICMD	ICMD call not authorized from any program (AOIS=N).
0110	003C	ICMD	RACF not available for authorization check (AOIS=A|R).
0110	0040	ICMD	DFSCCMD0 not available for authorization check (AOIS=A|C).
0110	0050	APSB	A CPI-C driven application issued an APSB call and the userid is not authorized to RACF.
0900	N/A	CHKP INIT INQY LOG ROLB ROLS SETS SETU SNAP[2] STAT[3] SYNC XRST	Reason code unmodified on 0900. PCB status code analysis required.

Notes:

1. GSCD is a Product-sensitive programming interface.
2. SNAP is a Product-sensitive programming interface.
3. STAT is a Product-sensitive programming interface.
4. When using the AIBTDLI interface for functions that do not require a PCB (such as APSB or GMSG), you can receive this return/reason code instead of X'0104'/X'0404'. You can also receive this return/reason code if the function is invalid and no PCB name is passed in the AIB.

DL/I Return and Reason Code Explanations

0000/0000

Explanation: Call was completed successfully.

For GMSG calls, there is at least one more segment to retrieve.

For ICMD calls, when the AIBOAUSE field is zero, no segment was returned and there are no more segments to retrieve. This can occur when the only response to the command is a DFS058 message indicating either COMMAND IN PROGRESS or COMMAND COMPLETE. When the AIBOAUSE field is nonzero, a segment was returned to the I/O area and there is at least one more segment to retrieve.

For RCMD calls, there is at least one more segment to retrieve.

Programmer Response: Proceed with processing. For GMSG, ICMD, and RCMD calls, you may issue another GMSG or RCMD call to retrieve the next segment.

0004/0004

Explanation: A message segment was returned to the I/O area. This is the last segment for the message or command response.

Programmer Response: Proceed with processing.

0004/0014

Explanation: No more messages are available for processing at this time. No segment is returned.

Programmer Response: You may want the AO application to terminate at this time.

0004/0018

Explanation: No more segments are available for this message or command response. No segment is returned.

Programmer Response: Process the message or command response previously returned.

0004/0048

Explanation: The AO application issued a GMSG call with the WAITAOI subfunction specified. While the AO application was waiting for an AOI message, the region was posted due to an IMS system shutdown. No message segment is returned.

Programmer Response: The AO program should terminate.

0004/004C

Explanation: The AO application issued a GMSG call with the WAITAOI subfunction. While the AO application was waiting for an AOI message, the region was posted due to a /PSTOP region command. No message segment is returned.

Programmer Response: The AO application should terminate.

0100/000C

Explanation: The I/O area was too small to contain all the data; partial data was returned. The AIBOAUSE field contains the length required to receive all of the data. The AIBOALEN field contains the actual length of the data.

Programmer Response: For INQY calls, the call may be reissued with a larger I/O area to retrieve all of the output data. If the data being returned is a message segment resulting from an ICMD or RCMD call, you may reissue the ICMD call with a larger I/O area.

0104/020C

Explanation: A CPI-C driven application issued a DL/I call before a PSB was scheduled. The APSB call must be issued to schedule a PSB before issuing any other DL/I call.

Programmer Response: Correct the CPI-C driven application so that it issues an APSB call before issuing any other DL/I call.

0104/0204

Explanation: AIBRSNM2 did not contain the IMS/DBCTL DRA startup table suffix on an ODBA APSB or CIMS subfunction INIT or TERM call.

Programmer Response: Set AIBRSNM2 to the correct 1 to 4 character DRA startup table suffix before issuing the call.

0104/0208

Explanation: One of the following application programming errors occurred:

- An invalid PCB name was passed in the AIB for a DL/I call requiring a PCB.
- An invalid DL/I call function was specified, and the PCB name was either not specified or not found in the scheduled PSB.
- The program issued a valid DL/I call that does not use a PCB. However, the call is not supported in the caller's environment (for example, GMSG call in a batch environment).

Programmer Response: Correct the DL/I call in the program.

0104/0210

Explanation: The I/O area length in the AIB (AIBOALEN field) is zero or negative. A negative field may be caused by the application passing the length as a character string instead of binary data. No data is returned to the application.

Programmer Response: Correct the program to specify the correct I/O area length in the AIB.

0104/0218

Explanation: The subfunction specified in the AIB for the DL/I call was invalid.

Programmer Response: Correct the DL/I call in the program. Not all DL/I calls support subfunctions. Refer to the DL/I call documentation for a list of valid subfunctions for the call.

0104/021C

Explanation: The AOI token was not specified in the AIB (AIBRSNM1 field) on the initial GMSG call for a message. The first GMSG call issued by an AO application must specify an AOI token. The AOI token must contain blanks when the GMSG call is used to get subsequent segments of the current message.

Programmer Response: Correct the AO application to specify an AOI token on the initial GMSG call for a message.

0104/0220

Explanation: GMSG call with WAITAOI subfunction did not specify an AOI token. The WAITAOI subfunction may only be specified when retrieving the first segment of a message. The WAITAOI subfunction requires that an AOI token be specified in the AIB (AIBRSNM1 field).

Programmer Response: Correct the program to specify an AOI token on the initial GMSG call for a message.

0104/0224

Explanation: The AOI token specified on a GMSG call contained one or more invalid characters. An AOI token may contain characters A-Z, 0-9, @, #, and $.

Programmer Response: Correct the program so the AOI token uses only valid characters.

0104/0228

Explanation: The AIB length specified in AIBLEN is less than the minimum required length.

Programmer Response: Response: Insure that the AIB is >= 264 bytes and AIBLEN reflects the correct length.

0104/022C

Explanation: A new ODBA thread request (APSB) cannot be processed because the maximum number of threads are currently allocated. The call is not processed.

Programmer Response: Try your request again, otherwise contact your IMS system programmer.

0104/0404

Explanation: The function code specified on the DL/I call is unknown or invalid. The program may be coded incorrectly, or the program may be executing on a level of IMS that does not support the call.

Programmer Response: Make sure the call is coded correctly and that the level of IMS in which the program is executing supports the call function.

0104/0440

Explanation: A DL/I call that is only valid from a CPI-C driven application was issued by a nonCPI-C driven program.

Programmer Response: Correct the program so it either does not issue the call or executes as a CPI-C driven program.

0104/0440

Explanation: A DL/I call that is only valid from a CPI-C driven application was issued by a nonCPI-C driven program.

Programmer Response: Correct the program so it either does not issue the call or executes as a CPI-C driven program.

0104/0440

Explanation: A DL/I call that is only valid from a CPI-C driven application was issued by a nonCPI-C driven program.

Programmer Response: Correct the program so it either does not issue the call or executes as a CPI-C driven program.

0104/0440

Explanation: A DL/I call that is only valid from a CPI-C driven application was issued by a nonCPI-C driven program.

Programmer Response: Correct the program so it either does not issue the call or executes as a CPI-C driven program.

0104/0444

Explanation: A DL/I call was issued that is only valid from the specific environment.

Programmer Response: Correct the program so it either does not issue the call or executes as a CPI-C driven program.

Correct the program so that it either doesn't issue the call or runs in the correct region type.

0104/0448

Explanation: A DL/I call that is only valid from a nonCPI-C driven program was issued by a CPI-C driven program. Some DL/I calls are allowed from CPI-C driven programs as long as the call does not use the I/O PCB.

Programmer Response: Correct the CPI-C driven program so that it either doesn't issue the call or uses an alternate PCB instead of the I/O PCB.

0104/044C

Explanation: A DL/I call was issued that is invalid from a specific dependent region type.

Programmer Response: Correct the program so that it either doesn't issue the call or runs in the correct region type.

0104/0450

Explanation: There are no messages to retrieve. The AO exit routine DFSAOE00 does not exist in the IMS system.

Programmer Response: The program should terminate. Contact your IMS system programmer to install DFSAOE00 in your system.

0104/048C

Explanation: The CPI-C driven program attempted to allocate a PSB when one was already allocated. This can be caused when a CPI-C driven program does not issue a DPSB to deallocate the PSB allocated by a previous APSB call.

Programmer Response: Correct the program to issue a DPSB call.

0104/0490

Explanation: The CPI-C driven program tried to either allocate or deallocate a PSB without committing resources affected by a previous DL/I or SQL call. When DL/I or SQL calls are issued from a CPI-C driven program, a CPI-RR commit (SRRCMIT) call must be issued to commit the resources before the PSB can be deallocated.

Programmer Response: Correct the program to issue either a CPI-RR commit or backout for the resources before issuing the DPSB call.

0104/0494

Explanation: The CPI-C driven program or ODBA application tried to deallocate a PSB, but the PSB was not allocated. Either there was no PSB allocated or the PSB that was allocated was not the same PSB specified on the DPSB call. The error may be caused by one of the following situations:

- Multiple DPSB calls are issued without intervening APSB calls.
- DPSB call is issued after an APSB call fails.
- DPSB call is issued with an incorrect PSB name.
- AIBRSNM2 does not contain the PSB name.

Programmer Response: Correct the program to issue one DPSB call for each APSB call, and ensure the PSB names are the same on both calls.

0104/054C

Explanation: The DL/I call specified an invalid message queue destination. The error may be caused by one of the following situations:

- The destination is a dynamic terminal but ETO is not activated.
- The program specified the wrong destination name on a CHNG call.
- The LTERM name defined on an alternate PCB is incorrect.

Programmer Response: Correct the program.

0104/0550

Explanation: The program attempted to insert data to the message queue when a DEALLOCATE_ABEND call was previously issued for the destination. Additional ISRT or SETO calls for this destination are rejected until an application sync point is processed.

The error can also be caused when a SETO DEALLOCATE_ABEND call is issued but there were no previous inserts for the destination.

Programmer Response: Analyze the program to determine if the insert logic is in error. A GU or CHKP call may be issued as appropriate to trigger an application sync point and retrieve the next input message; this will allow further inserts to the destination. CPI-C driven programs may issue a CPI-RR commit request to trigger sync point processing.

0104/0610

Explanation: The I/O area address was not specified in the call list. No data is returned to the program.

Programmer Response: Correct the program to specify the I/O area address in the call list.

0104/060C

Explanation: An invalid class name was specified in the I/O area on the AUTH call. Refer to the AUTH call documentation in the Application Programming Transaction manager for a list of valid classes.

Programmer Response: Correct the program.

0104/0618

Explanation: The I/O area length is longer than the PSB work area defined by the IOASIZE parameter on PSBGEN. IMS is not able to copy the contents of the I/O area to the PSB work area.

Programmer Response: Make sure that the I/O area length specified in the AIB (AIBOALEN field) is correct and is not larger than the size specified in the IOASIZE parameter of the PSB.

0104/0620

Explanation: The ZZ field of the I/O area is not zero. The program must not use the ZZ field in the I/O area.

Programmer Response: Correct the program.

0104/0754

Explanation: The LL field of the I/O area is less than 5. The length of the segment, excluding the LLZZ field, must be at least 1, and LL must be at least 5.

Programmer Response: Correct the program.

0104/0810

Explanation: The option list parameter was not specified in the call list.

Programmer Response: Correct the program.

0104/0920

Explanation: The Queue Space Notification exit (DFSQSPC0) determined that the program reached its limit of data that can be inserted to the message queue. The program may be in a loop inserting information to the IMS message queues. Additional ISRT or SETO calls for this destination are rejected until an application sync point is processed.

Programmer Response: Analyze the program to determine if the insert logic is in error. A GU or CHKP call may be issued as appropriate to initiate an application sync point and retrieve the next input message; this will allow further inserts to the destination. CPI-C driven programs may issue a CPI-RR commit request to initiate sync point processing.

0104/1004

Explanation: A valid IMS command was specified in the I/O area of an ICMD call; however, the command as specified is not allowed from an application. For a list of commands that may be specified from an AO application, see *IMS/ESA Operator's Reference*.

Programmer Response: Correct the AO application so only valid automated operator commands are issued.

0104/1008

Explanation: An invalid command verb was specified in the I/O area.

Programmer Response: Correct the AO application.

0104/1010

Explanation: The AO application issued an RCMD call; however, there were no command response segments to retrieve since the AO application had not issued a prior ICMD call. The AO application must issue an IMS command through the ICMD call before the RCMD call is valid.

Programmer Response: Correct the AO application.

0108/0004

Explanation: An error was encountered during PC router processing. This is a system error.

Programmer Response: Contact your IMS system programmer.

0108/0010

Explanation: A storage allocation request failed to obtain private storage. The call is not processed. This is a system error.

0108/0014

Explanation: A storage allocation request failed to obtain global storage. The call is not processed. This is a system error.

Programmer Response: Contact your IMS system programmer.

0108/0018

Explanation: IMS encountered an internal latch error. For GMSG calls, no segment is returned.

Programmer Response: Contact your IMS system programmer.

0108/001C

Explanation: During the processing of an ODBA CIMS AIBSFUNC=INIT or APSB call, the attach of an ODBA TCB failed in the Application address space. The call is not processed. This is a system error.

Programmer Response: Contact your IMS system programmer.

0108/0020

Explanation: The load of a required DRA module or load of the DRA startup table (DFScccc0) failed in the Application address space. The name is returned to the calling program in AIBRSNM2.

Programmer Response: Insure that the named module is linked into the proper library.

0108/0024

Explanation: The OPEN of the DRA library specified in the DRA startup table failed.

Programmer Response: Insure the library specified in the DRA startup table (DFScccc0) is correct and accessible, otherwise contact your IMS system programmer.

0108/0028

Explanation: Dynamic allocation of the DRA reslib (DSNAME=DRA startup table) failed. The call is not processed. This is a system error.

Programmer Response: If the DSNAME= library is correct, contact your IMS system programmer.

0108/002C

Explanation: ODBA encountered an OS/390 Name/Token Service error in the Application address space. The call is not processed. This is a system error.

Programmer Response: Contact your IMS system programmer.

0108/0030

Explanation: During Spool API processing, a data set CLOSE error was encountered. A X'67D0' log record was written to indicate the cause of the failure.

Programmer Response: Contact your IMS system programmer to determine the reason for the CLOSE error.

0108/0034

Explanation: During Spool API processing, a data set deallocation (DYNALLOC) error was encountered. A X'67D0' log record was written to indicate the cause of the failure.

Programmer Response: Contact your IMS system programmer to determine the reason for the DYNALLOC error.

0108/0038

Explanation: During Spool API processing, a data set allocation (DYNALLOC) error was encountered. A X'67D0' log record was written to indicate the cause of the failure.

Programmer Response: Contact your IMS system programmer to determine the reason for the DYNALLOC error.

0108/003C

Explanation: During Spool API processing, a data set OPEN error was encountered. A X'67D0' log record was written to indicate the cause of the failure.

Programmer Response: Contact your IMS system programmer to determine the reason for the OPEN error.

0108/0040

Explanation: During Spool API processing, an error was encountered while trying to write to the data set. A X'67D0' log record was written to indicate the cause of the failure.

Programmer Response: Contact your IMS system programmer to determine the reason for the write error.

0108/0044

Explanation: During Spool API processing, an error was encountered while trying to create an output descriptor (OUTADD). A X'67D0' log record was written to indicate the cause of the failure.

Programmer Response: Contact your IMS system programmer to determine the reason for the OUTADD error.

0108/0048

Explanation: During Spool API processing, an error was encountered while trying to delete an output descriptor (OUTDEL). A X'67D0' log record was written to indicate the cause of the failure.

Programmer Response: Contact your IMS system programmer to determine the reason for the OUTDEL error.

0108/0060

Explanation: During call processing, IMS issued a DFSLUMIF call which failed. This is a system error. A X'67D0' log record was written to indicate the cause of the failure.

Programmer Response: Contact your IMS system programmer.

0108/0064

Explanation: During call processing, IMS issued a DFSYFND call which failed. This is a system error. A X'67D0' log record was written to indicate the cause of the failure.

Programmer Response: Contact your IMS system programmer.

0108/0068

Explanation: During call processing, IMS issued a DFSYLUM call which failed. This is a system error. A X'67D0' log record was written to indicate the cause of the failure.

Programmer Response: Contact your IMS system programmer.

0108/0080

Explanation: A RACF RACINIT failure occurred for an LU 6.2 originated transaction. An ACEE could not be created for the authorization check for the AUTH call.

Programmer Response: Contact your IMS system programmer.

0108/020C

Explanation: While processing the IMS command issued by an AO application, the command processor encountered an invalid AWE.

Programmer Response: Contact your IMS system programmer.

0108/0214

Explanation: A DRA startup table parameter is incorrectly specified. AIBERRXT contains information regarding the error as follows:

- 'MnMx' = MinThrd or MaxThrd is not between 1 and 255.
- 'FPB ' = FPBUF or FPBOF is not between 0 and 999.
- 'CNBA' = CNBA is less than FPBUF.
- The call is not processed

Programmer Response: Correct and rebuild the DRA startup table.

0108/0220

Explanation: The PST LU 6.2 extension address is zero. This address, obtained during message GU processing, is required to obtain the user token for an LU 6.2 originated transaction.

Programmer Response: Contact your IMS system programmer.

0108/0224

Explanation: The user security token (UTOKEN) is zero. The UTOKEN is obtained during the message GU processing and is required to build the ACEE for the authorization check for an LU 6.2 originated transaction.

Programmer Response: Contact your IMS system programmer.

0108/0240

Explanation: The dependent region was posted with an invalid post code. No segment is returned. This is a system error.

Programmer Response: Contact your IMS system programmer.

0108/0244

Explanation: The dependent region was posted unexpectedly during abnormal terminate thread processing. No segment is returned. This is a system error.

Programmer Response: Contact your IMS system programmer.

0108/0304

Explanation: The application attempted to allocate a PSB that IMS could not find. The application program may have specified an incorrect PSB name, or the PSB may not have been generated in the IMS system.

Programmer Response: Analyze the application program to determine if the cause of the error is the PSB. If the PSB name is correct, make sure the PSB is defined to the IMS system.

0108/0308

Explanation: IMS encountered a PSB authorization failure while attempting to allocate the PSB. Security checking used the application group name (AGN) table. Either the AGN table did not exist or the PSB name was not defined in the table.

Programmer Response: Contact your IMS system programmer to determine if the AGN table exists in the IMS system and to ensure the PSB is defined in it.

0108/030C

Explanation: The program attempted to allocate a PSB that is marked permanently bad. IMS cannot allocate the PSB.

Programmer Response: Contact your IMS system programmer.

0108/0310

Explanation: The PSB cannot be allocated due to a stopped or locked database.

Programmer Response: Determine why the database is stopped or locked. If appropriate, the IMS commands /START and /UNLOCK can be used to change the state of the database.

0108/0314

Explanation: The PSB could not be allocated since it was already scheduled and it does not support parallel scheduling.

Programmer Response: If the PSB should support parallel scheduling, contact your IMS system programmer to make sure the APPLCTN macro specifies SCHDTYP=PARALLEL for the PSB.

0108/0318

Explanation: The PSB could not be allocated since it is stopped or locked.

Programmer Response: Determine why the PSB is stopped or locked. If appropriate, the IMS commands /START and /UNLOCK can be used to change the state of the PSB.

0108/031C

Explanation: An I/O error was encountered while reading the PSB or DMB.

Programmer Response: Contact your IMS system programmer.

0108/0320

Explanation: Either the PSBW pool is too small for the PSB or the DLDP pool is too small for the DMB.

Programmer Response: Contact your IMS system programmer.

0108/0324

Explanation: The PSB was generated with a PROCOPT of L or LS in a nonHSAM PCB.

Programmer Response: Contact your IMS system programmer.

0108/0328

Explanation: During PSB allocation, a Fast Path buffer fix error was encountered.

Programmer Response: Contact your IMS system programmer.

0108/032C

Explanation: During PSB allocation, an invalid processing intent was encountered.

Programmer Response: Contact your IMS system programmer.

0108/0540

Explanation: No active communications with IMS or DBCTL. The call is not processed.

Programmer Response: Contact your IMS system programmer.

0108/0544

Explanation: RRS/MVS is not active at the time that ODBA attempts to establish a connection to IMS or DBCTL. RRS/MVS is required for ODBA. The call is not processed.

Programmer Response: Contact your IMS system programmer.

0108/0548

Explanation: The SWITCH_CONTEXT call failed during ODBA APSB call processing. The call is not processed. This is a system error.

Programmer Response: Contact your IMS system programmer.

0108/054C

Explanation: A psuedoabend U0458 was returned on an ODBA APSB call. One of the Fast Path databases used by the PSB named in AIBRSNM1 was stopped or locked by a prior program failure. The call is not processed.

Programmer Response: Determine the reason for the Fast Path database being stopped. After the problem that caused the database to be stopped has been corrected, resubmit the request.

0108/0550

Explanation: The recovery token is not unique or the OS/390 Application address space ID is invalid. The call is not processed. This is a system error.

Programmer Response: Contact your IMS system programmer.

0108/0554

Explanation: The recovery token is zero. The call is not processed. This is a system error.

Programmer Response: Contact your IMS system programmer.

0108/0558

Explanation: An ODBA APSB schedule failed for a /STOP REGION command. The call is not processed.

Programmer Response: None.

0108/0560

Explanation: An ODBA APSB schedule failed for a /STOP REGION ABDUMP command. The call is not processed.

Programmer Response: None.

0108/0564

Explanation: A psuedo abend U0141 was received on an ODBA APSB request. The call is not processed. This is a system error.

Programmer Response: Contact your IMS system programmer.

0108/0568

Explanation: A psuedo abend U0145 was received on an ODBA APSB request. The call is not processed. This is a system error.

Programmer Response: Contact your IMS system programmer.

0110/000C

Explanation: The user is not authorized for access to the resource name in the class requested in the AUTH call. No 'installation data' is returned.

Programmer Response: Check the program to ensure the authorization check was performed on the correct resource. If it was, check with the person who handles security in your installation to determine whether RACF authorization is required for the resource.

0110/0020

Explanation: An AO application tried to issue an IMS command. Security checking determined that the AO application is not authorized to RACF.

Programmer Response: Contact your RACF administrator to authorize your AO application to RACF.

Programmer Response: Contact your IMS system programmer to authorize your AO application to RACF.

0110/0024

Explanation: An AO application tried to issue an IMS command. Security checking determined that the userid is not authorized to RACF.

Programmer Response: Contact your RACF administrator to authorize your userid to RACF.

0110/0028

Explanation: An AO application tried to issue an IMS command. Security checking determined that the AO application is not authorized to issue the command.

Programmer Response: Contact your IMS system programmer.

0110/002C

Explanation: An AO application tried to issue an IMS command. Security checking determined that the userid is not authorized to issue the command.

Programmer Response: Contact your IMS system programmer.

0110/0030

Explanation: An AO application tried to issue an IMS command. The Command Authorization exit routine (DFSCCMD0) indicated that the program was not authorized to issue the command.

Programmer Response: Contact your IMS system programmer to determine why DFSCCMD0 does not recognize your AO application.

0110/0034

Explanation: An AO application tried to issue an IMS command. The Command Authorization exit routine (DFSCCMD0) indicated the userid was not authorized to issue the command.

Programmer Response: Contact your IMS system programmer to determine why DFSCCMD0 does not recognize your userid.

0110/0038

Explanation: An AO application tried to issue an IMS command. The ICMD call is not authorized from any program because the IMS execution parameter AOIS= was specified as N.

Programmer Response: Contact your IMS system programmer to have the AOIS= execution parameter changed to allow commands from an AO application.

0110/003C

Explanation: An AO application tried to issue an IMS command. RACF is not available for the authorization check. The IMS execution parameter AOIS= was specified as either A or R.

Programmer Response: Contact your IMS system programmer to determine why RACF is not available to the IMS system.

0110/0040

Explanation: An AO application tried to issue an IMS command. The Command Authorization exit routine (DFSCCMD0) is not available for the authorization check. The IMS execution parameter AOIS= was specified as either A or C.

Programmer Response: Contact your IMS system programmer to determine why DFSCCMD0 was not included in the IMS system.

0110/0050

Explanation: A CPI-C driven application issued an APSB call. IMS determined that the userid is not authorized by RACF (or equivalent) to use the PSB. Field AIBERRXT of the AIB contains additional information as follows (see the RACF manual, *External Security Interface (RACROUTE) Macro Reference for MVS and VM* for a description of the codes in AIBERRXT, if you use RACF):

- Bytes 0 to 1 - SAF return code
- Byte 2 - RACF or installation exit return code
- Byte 3 - RACF or installation exit reason code

Programmer Response: Contact your Security administrator to authorize your userid to use the PSB.

0108/0070

Explanation: The reslib specified in the DRA startup table (DSNAME=) is not authorized.

Programmer Response: Contact your IMS system programmer.

0900

Explanation: The DL/I call, which requires a PCB for processing, encountered an error during processing. The cause of this error does not currently have a unique AIB return and reason code defined. The AIB (AIBRSA1 field) contains the PCB address. You must analyze the status code in the PCB to determine the cause of the error. The reason code is not modified when the return code is 0900.

Programmer Response: IMS intends to define unique AIB return and reason codes for most if not all DL/I call processing errors. The error that causes a X'0900' return code may become a unique return and reason code in the future.

Part 5. Appendixes

© Copyright IBM Corp. 1974, 2000

Appendix A. Sample Applications

A description of the IVP Sample Application is in the *IMS/ESA Installation Volume 1: Installation and Verification*. The source for the IVP Sample Application is in the IMS.DFSISRC (SMP/E target) library. Two programs are provided in several different languages. The two programs are:

DFSIVA3 A Conversational MPP that accesses an HDAM/VSAM database. Transaction input and output is through MFS screens.

DFSIVA6 A Batch or BMP program that accesses a HIDAM/OSAM database. The program uses GSAM to receive its transaction input and to display its transaction output.

These programs both perform the same application function—a simple phone book application. The programs are available in several language versions. Those available in IMS are listed in Table 59.

Table 59. Program Languages Available for IVP Sample Program

Program	Language	PSB	MFS	Trancode	Comp/Lked
DFSIVA3	Assembler	DFSIVP3	DFSIVF3	IVTCV	n/a
DFSIVA31	Pascal	DFSIVP31	DFSIVF31	IVTCVP	DFSIVJP3
DFSIVA32	C language	DFSIVP32	DFSIVF32	IVTCVC	DFSIVJC3
DFSIVA34	COBOL	DFSIVP34	DFSIVF34	IVTCVB	DFSIVJB3
DFSIVA35	REXX	DFSIVP35	DFSIVF35	IVTCVX	n/a
DFSIVA6	Assembler	DFSIVP6	n/a	n/a	n/a
DFSIVA61	Pascal	DFSIVP61	n/a	n/a	DFSIVJP6
DFSIVA62	C language	DFSIVP62	n/a	n/a	DFSIVJC6
DFSIVA64	COBOL	DFSIVP64	n/a	n/a	DFSIVJB6
DFSIVA65	REXX	DFSIVP65	n/a	n/a	n/a
DFSIVG20	Assembler	n/a	n/a	n/a	DFSIVJG2
DFSIVG30	Assembler	n/a	n/a	n/a	DFSIVJG3

Programs DFSIVA3 and DFSIVA6 are fully installed and executed by the IVP.

The IVP System Definition includes APPLCTN and TRANSACT macros for all language versions that are supported by the type of system being generated. The IVP also performs PSBGEN, ACBGEN, and MFSUTL for the language versions.

Programs DFSIVA31/32/34 and DFSIVA61/62/64 must be compiled by the user. Sample compile/link-edit JCL is provided in the IMS.DFSISRC library.

Programs DFSIVA35 and DFSIVA65 are fully installed by the IVP, but they are not included in the IVP execution instructions. DFSIVA35 can be executed from any 24x80 (3270) MFS device. DFSIVA65 can be executed by modifying the IVP execution JCL for DFSIVA6.

Once compiled and link-edited, programs DFSIVA31/32/34 are executable from any 24×80 (3270) MFS device.

Once compiled and link-edited, programs DFSIVA61/62/64 can be executed by modifying the IVP execution JCL for DFSIVA6.

© Copyright IBM Corp. 1974, 2000

Sample Applications

Programs DFSIVG20 and DFSIVG30 are WTOR subroutines for the Pascal and C language programs, respectively.

The IMS EXEC library also includes the REXX exec named DFSSUT04 EXEC. Use this exec to process any unexpected return codes or status codes.

Appendix B. MFS Definitions for Intersystem Communication

The following prototype MFS definitions can be used in an intersystem communication (ISC) system between IMS and CICS. In this example:

- CICS can request MFS editing with either 8-byte or 4-byte names.
- Messages from CICS can be multiple-page input or single-page input.
- Output to CICS can be one message of one page or multiple pages with one or more segments.
- Demand paged or autopaged output can be requested of IMS.

These formats can also be used by a 3270 terminal operator who wants to send a message to CICS using an IMS message switch. Or, for example, an IMS message switch can be invoked by a user at a 3270 terminal, the message can be switched to CICS, and a reply is returned from CICS to IMS and then to the 3270 terminal. The routing is handled by MFS.

```
FMTDIS    FMT
          DEV    TYPE=3270-A2,FEAT=IGNORE
          DIV    TYPE=INOUT
DFLDIND1  DFLD   LTH=5,POS=(1,2)
DFLDIND2  DFLD   LTH=100,POS=(1,8)
          FMTEND
FMTDP2    FMT
          DEV    TYPE=DPM-B1,FEAT=IGNORE,
                 MODE=RECORD,DSCA=X'00A0'                        X
          DIV    TYPE=OUTPUT,OPTIONS=(MSG,NODNM)
PPAGE1    PPAGE
DFLDOUT1  DFLD   LTH=5
DFLDOUT2  DFLD   LTH=100
          FMTEND
FMTDPM    FMT
          DEV    TYPE=DPM-B1,FEAT=IGNORE,MODE=RECORD
          DIV    TYPE=INPUT,OPTIONS=(DPAGE,NODNM),                        X
                 PRN=DFLDINP3,                                            X
                 RDPN=DFLDINP4,                                           X
                 RPRN=DFLDINP5
PPAGE2    PPAGE
DFLDINP1  DFLD   LTH=5
DFLDINP2  DFLD   LTH=100
          DIV    TYPE=OUTPUT,OPTIONS=(DPAGE,NODNM)
DPAGE2    DPAGE
DPAGE3    PPAGE
DFLDOUT3  DFLD   LTH=5
DFLDOUT4  DFLD   LTH=100
          DFLD   SCA,LTH=2
          FMTEND
MFSMOD1   MSG    TYPE=OUTPUT,SOR=(FMTDP2,IGNORE),                         X
                 NXT=MFSMID1
          SEG
          MFLD   DFLDOUT1,LTH=5
          MFLD   DFLDOUT2,LTH=100
          MSGEND
```

Figure 81. Sample 2—MFS Definition Format

© Copyright IBM Corp. 1974, 2000

```
MFSMODE2  MSG    TYPE=OUTPUT,SOR=(FMTDPM,IGNORE),                    X
                 NXT=MFSMID1
          SEG
          MFLD   DFLDOUT3,LTH=5
          MFLD   DFLDOUT4,LTH=100
          MFLD   (,SCA),LTH=2
          MSGEND
MFSMID1   MSG    TYPE=INPUT,SOR=(FMTDPM,IGNORE),                     X
                 NXT=MFSMODD
          SEG
          MFLD   DFLDINP1,LTH=5
          MFLD   DFLDINP3,LTH=8
          MFLD   DFLDINP4,LTH=8
          MFLD   DFLDINP5,LTH=8
          MFLD   DFLDINP2,LTH=100
          MSGEND
MFSMIDD   MSG    TYPE=INPUT,SOR=(FMTDIS,IGNORE),                     X
                 NXT=MFSMOD1
          SEG
          MFLD   DFLDIND1,LTH=5
          MFLD   DFLDIND2,LTH=100
          MSGEND
MFSMODD   MSG    TYPE=INPUT,SOR=(FMTDIS,IGNORE),
                 NXT=MFSMIDD
          SEG
          MFLD   DFLDIND1,LTH=5
          MFLD   DFLDIND2,LTH=100
          MSGEND
          END
```

Figure 82. Sample 2—MFS Definition Format

Appendix C. Device Compatibility with Previous Versions of MFS

If you choose not to define 3270 devices during IMS system definition using the device type symbolic name (option 1), no changes to device format definitions are needed.

If you choose to define 3270 devices during IMS system definition using a device type symbolic name (3270-An) (options 2, 3, and 4), in some cases you must make changes in your 3270 device format definitions.

The examples in Table 60 include the recommended standard for relating the device type symbolic name to the screen size:

Table 60. MFS Device Definition Compatibility for 3270 Devices

Device and Screen Size	Current IMS System Definition[1]	New IMS System Definition[1]
3275 or 3277 (12X40)	MFS: DEV TYPE= (3270,1) Model 1	MFS: DEV TYPE= 3270-A5 [24]
3275, 3276, 3277, 3278 (24X80)	MFS: DEV TYPE= (3270,2) Model 2	MFS: DEV TYPE= 3270-A2 [24]
3276, 3278 (12X80)	MFS: DEV TYPE= (3270,1) Model 1	MFS: DEV TYPE= 3270-A1 [23]
3276, 3278 (32X80)	MFS: DEV TYPE= (3270,2) Model 2	MFS: DEV TYPE= 3270-A3 [23]
3276, 3278 (43X80)	MFS: DEV TYPE= (3270,2) Model 2	MFS: DEV TYPE= 3270-A4 [23]
3278 (27X132)	MFS: DEV TYPE= (3270,2) Model 2	MFS: DEV TYPE= 3270-A7 [23]

Notes:

1. For screen sizes specified in type or terminal macro.
2. If the format will be used on the new device and will not be used on the old device, change TYPE= (3270,1) or (3270,2) to 3270-An with the corresponding screen size, and recompile.
3. See option 3 in Table 61.
4. See option 4 in Table 61.

Table 61 lists the advantages and disadvantages of selecting a specific option for the larger screen device and the required action if you choose to use existing device formats.

Table 61. Advantages and Disadvantages of Larger Screen Device

Option	Advantage	Disadvantage	Conversion Action Required
1	You can use existing MFS formats unchanged.	You cannot use full screen.	No (Use current formats as shown in Table 60.)
2	You can use full screen.	You must design new device formats.	No (Define new formats.)

© Copyright IBM Corp. 1974, 2000

Table 61. Advantages and Disadvantages of Larger Screen Device (continued)

Option	Advantage	Disadvantage	Conversion Action Required
3	You can use existing formats as a migration path on the new screen device and you can gradually replace them with new device formats.	You must modify existing device formats to use the device symbolic name.	Yes (Refer to Table 60.)
4	Consistency in definition for current and new screen sizes.	You must modify all formats.	Yes (Refer to Table 60.)

Using STACK/UNSTACK to Convert MFS Device Formats to Symbolic Name Formats

The IMS MFS language utility's compilation statements STACK and UNSTACK can be used to convert existing MFS 3270 device formats to the user-defined device type symbolic name formats. The STACK statement is used to delineate one or more SYSIN or SYSLIB records, and to request that those records, once processed, be kept in storage for use at a later time. The UNSTACK statement requests retrieval of a previously processed stack of SYSIN/SYSLIB records.

Example: With the following existing 3270 format definition:

```
label    FMT
         DEV       TYPE=(3270,2), ...
         DIV       TYPE=INOUT
         DPAGE     CURSOR=((2,3))
label    DFLD
label    DFLD
label    DFLD
         FMTEND
```

You can create a user-defined device type symbolic name (via TYPE=3270-An) format for the large screened display devices by using the DEV statement and the compilation statements STACK and UNSTACK as follows:

```
label    FMT
         DEV       TYPE=3270,2,...
         STACK     ON
         DIV       TYPE=INOUT
         DPAGE     CURSOR=((1,2))
label    DFLD
label    DFLD
label    DFLD
         STACK     OFF
         DEV       TYPE=3270-A2,...
         UNSTACK
         FMTEND
```

The UNSTACK statement causes the statements between STACK ON and STACK OFF to be duplicated. In addition to the 3270 model 2 device format, a device format is created for the 3270-A2, which has the same device layout as the 3270 model 2.

3270 Device Format Conversion Example

The following example is provided to clarify Table 60 on page 507. Assume that the installation has 3270 model 1 and model 2 display devices and has installed additional display devices with 12×80, 24×80, 32×80, and 43×80 screen sizes. A new IMS system definition was performed for the additional devices, and the 3270 model 1 and model 2 devices were redefined to specify the device symbolic name.

The IMS system definition specifications for these 3270 displays were as follows:

- TYPE=3270-A1, SIZE=(12x80) for the additional devices with 12x80 screen size.
- TYPE=3270-A2, SIZE=(24x80) for the 3270 model 2 and additional devices with 24x80 screen size.
- TYPE=3270-A3, SIZE=(32x80) for the additional devices with 32x80 screen size.
- TYPE=3270-A4, SIZE=(43x80) for the additional devices with 43x80 screen size.
- TYPE=3270-A5, SIZE=(12x40) for the 3270 model 1 device.

The following MFS changes were required to convert existing 3270 model 1 and 2 device format definitions for use on the 3270 model 1, model 2, and on the additional devices:

Existing Definitions:

```
label    FMT
         DEV       TYPE=(3270,1)
         DIV       TYPE=INOUT
         DPAGE                  ...
label    DFLD                   ...
label    DFLD                   ...
label    DFLD                   ...
         DEV       TYPE=(3270,2)
         DIV       TYPE=INOUT
         DPAGE                  ...
label    DFLD                   ...
label    DFLD                   ...
label    DFLD                   ...
         FMTEND                 ...
```

Changes Applied and Recompiled:

Device Format Conversion

```
label    FMT
         DEV       TYPE=3270-A5 (changed from (3270,1) to 3270
                                display with 12x40 screen size)
         STACK     ON
         DIV       TYPE=INOUT
         DPAGE                  ...
label    DFLD                   ...
label    DFLD                   ...
label    DFLD                   ...
         STACK     OFF
         DEV       TYPE=3270-A1 (3270 display with 12x80 screen
                                    size)
         UNSTACK
         DEV       TYPE-3270-A2 (changed from (3270,2) to 3270
                                    display with 24x80 screen size)
         STACK     ON
         DIV       TYPE=INOUT
         DPAGE                  ...
label    DFLD                   ...
label    DFLD                   ...
label    DFLD                   ...
         STACK     OFF
         DEV       TYPE=3270-A3(3270 display with 32x80 screen
                                    size)
         UNSTACK  ,KEEP
         DEV       TYPE=3270-A4(3270 display with 43x80 screen
                                    size)
         UNSTACK
         FMTEND
```

After the changes are applied and recompiled, the new device formats are designed to take advantage of each screen size, and the previous format definition can be compiled again as follows:

```
label    FMT
         DEV       TYPE=3270-A5
         DIV       TYPE=INOUT
         DPAGE                  ...
label    DFLD                   ...
label    DFLD                   ...
label    DFLD                   ...(existing device fields
                                using 12x40 screen size)
         DEV       TYPE=3270-A1
         DPAGE                  ...
label    DFLD                   ...(new device fields using
                                12x80 screen size)
         .
         .
label    DFLD                   ...
         DEV       TYPE=3270-A2
         DIV       TYPE=INOUT
         DPAGE                  ...
label    DFLD                   ...(existing device fields
                                using 24x80 screen size)
label    DFLD                   ...
label    DFLD                   ...
         DEV       TYPE=3270-A3
         DIV       TYPE=INOUT
         DPAGE                  ...
label    DFLD                   ...(new device fields using
                                32x80 screen size)
         .
         .
label    DFLD                   ...
         DEV       TYPE=3270-A4
         DIV       TYPE=INOUT
         DPAGE                  ...
label    DFLD                   ...(new device fields using
                                43x80 screen size)
         .
         .
label    DFLD                   ...
         FMTEND
```

3270 Printer and SLU 1 Compatibility

To use the extended attributes of color, highlighting, and programmed symbols, or to use the set vertical format or set line density data streams, you might need to modify your application programs.

Additional 3270 printer devices attached to a non-SNA control unit compatible with the currently installed 3270 printer devices use the existing 3270P model 1 or model 2 device formats. For the printer buffer, they use the existing IMS system definition with 480 characters (current model 1) or 1920 characters (current model 2).

MFS users choosing to change device attachment to SLU 1 must change their MFS device format definitions as shown in Table 62.

Table 62. MFS Device Definition Compatibility for 3270 Printers and SLU 1 Devices

Current Device	Current MFS DEV TYPE	New Device or Control Unit	New IMS System Definition	New MFS DEV TYPE	MVS/ESA Changes Required
3284/3286	3270P	3827/3289 attached to a 3274 or 3276 SNA control unit	SLUTYPE1	SCS1	See Note

3270/SLU1 Compatibility

Table 62. MFS Device Definition Compatibility for 3270 Printers and SLU 1 Devices (continued)

Current Device	Current MFS DEV TYPE	New Device or Control Unit	New IMS System Definition	New MFS DEV TYPE	MVS/ESA Changes Required

Note:

Change DEV TYPE=(3270P,n) to DEV TYPE=SCS1 and recompile. Or, if all printers are not changed to the new device or control unit, add the following after DEV TYPE=3270P and recompile:

```
STACK ON
(3270P DPAGE, DFLD statements)
STACK OFF
DEV TYPE=SCS1
UNSTACK
```

SLU P Compatibility

Application programs written for other MFS-supported devices can execute unchanged with SLU P (DPM-An) devices once the DIFs and DOFs appropriate for the DPM devices are defined. Changes might be required if the program depends on unique device-dependent features such as premodified fields on a 3270 display. The program would execute unchanged only if the premodified fields presented to the remote program are returned in the input message. This requires that the remote program properly interpret the attribute bytes of the output message field and handle the indicated device function in a way that satisfies the requirements of the IMS application program.

Existing IMS application programs that do not use MFS might have to be changed to adjust to the appropriate 3600 or 3790 buffer size and to ensure that message text is a compatible subset of the SCS character string.

IBM 3278-52/3283-52 and IBM 5550 Family (as 3270) Compatibility

The message format definitions for the IBM 3278-52/3283-52 are upwardly compatible. However, message formats created with Kanji functions for the 5550 family of devices cannot be used on the IBM 3278-52/3283-52.

Existing 3270 and IBM 5550 Family (as 3270) Compatibility

Note the following when adding field outlining and input control specifications to existing 3270 and 3278-52/3283-52 message formats:

- Field outlining
  - For the 3270 display, left line, right line, overline, and underline do not take up a position in the user field. The application program does not have to be modified unless dynamic modification of extended attributes is performed.
  - For the SCS1 printer, MFS reserves print positions for left and right lines. If a field starts from the leftmost column or ends at the rightmost column, the left or right line is not printed correctly because room is not available. To correct this, modify the application program to truncate 1 byte. If two adjacent fields

are logically one and the overline and underline should connect, the application program does not have to be modified.

In either case, for dynamic modification, the application program must be modified.

- DBCS/EBCDIC mixed fields
  - For 3270 displays, the SO/SI control characters take up 1 byte on the screen. This means that the length on the display is equal to the message format length. Therefore, the existing application program does not have to be changed.

    When assigning DBCS/EBCDIC mixed data to an existing EBCDIC field, the application program must check that the SO and SI are paired, that the EGCS data is of even length, and that neither the SO nor SI is truncated when the MFLD is mapped to the DFLD.
  - For SCS1 printers, MIX/MIXS must be specified when using DBCS/EBCDIC mixed data. In this case, the message length and the length of the output differ and the application program must modify the MFLD according to each field's characteristics.

Appendix D. Spool API

The IMS Spool API support provides feedback to the application program when IMS detects errors in the print data set options of the CHNG and SETO calls. For convenience, your application program can display these errors by sending a message to an IMS printer or by performing another action that lets you examine the parameter lists and feedback area without looking at a dump listing.

This information applies only to the calls as they are used with Spool API support.

Related Reading: For more detailed information on Spool API, see *IMS/ESA JES Spool API User's Guide*.

Understanding Parsing Errors

When you are diagnosing multiple parsing error return codes, the first code returned is usually the most informative.

Keywords

The CHNG and SETO calls have two types of keywords. The type of keyword determines what type of keyword validation IMS should perform. The keyword types are:

- Keywords valid for the calls (for example, IAFP, PRTO, TXTU, and OUTN)
- Keywords valid as operands of the PRTO keyword (for example CLASS and FORMS).

Incorrectly specified length fields can cause errors when IMS checks for valid keywords. When IMS is checking the validity of keywords on the CHNG and SETO calls, one set of keywords is valid. When IMS is checking the validity of keywords on the PRTO keyword, another set of keywords is valid. For this reason, incorrectly specified length fields can cause a scan to terminate prematurely, and keywords that appear to be valid are actually invalid because of where they occur in the call list. IMS might report that a valid keyword is invalid if it detects a keyword with an incorrect length field or a keyword that occurs in the wrong place in the call list.

Status Codes

The status code returned for the call can also suggest the location of the error. Although exceptions exist, generally, an AR status code is returned when the keyword is invalid for the call. An AS status code is returned when the keyword is invalid as a PRTO option.

Error Codes

This section contains information on Spool API error codes that your application program can receive. "Diagnosis Examples" on page 516 contain examples of errors and the resulting error codes provided to the application program.

Error Code	Reason
(0002)	Unrecognized option keyword. Possible reasons for this error are: • The keyword is misspelled. • The keyword is spelled correctly but is followed by an invalid delimiter.

© Copyright IBM Corp. 1974, 2000

- The length specified field representing the PRTO is shorter than the actual length of the options.
- A keyword is not valid for the indicated call.

(0004) Either too few or too many characters were specified in the option variable. An option variable following a keyword in the options list for the call is not within the length limits for the option.

(0006) The length field (LL) in the option variable is too large to be contained in the options list. The options list length field (LL) indicates that the options list ends before the end of the specified option variable.

(0008) The option variable contains an invalid character or does not begin with an alphabetic character.

(000A) A required option keyword was not specified.

Possible reasons for this error are:

- One or more additional keywords are required because one or more keywords were specified in the options list.
- The specified length of the options list is more than zero but the list does not contain any options.

(000C) The specified combination of option keywords is invalid. Possible causes for this error are:

- The keyword is not allowed because of other keywords specified in the options list.
- The option keyword is specified more than once.

(000E) IMS found an error in one or more operands while it was parsing the print data set descriptors. IMS usually uses MVS/ESA services (SJF) to validate the print descriptors (PRTO= option variable). When IMS calls SJF, it requests the same validation as for the TSO OUTDES command. Therefore, IMS is insensitive to changes in output descriptors. Valid descriptors for your system are a function of the MVS/ESA release level. For a list of valid descriptors and proper syntax, use the TSO HELP OUTDES command.

IMS must first establish that the format of the PRTO options is in a format that allows the use of SJF services. If it is not, IMS returns the status code **AS**, the error code (000E), and a descriptive error message. If the error is detected during the SJF process, the error message from SJF will include information of the form (R.C.=xxxx,REAS.=yyyyyyyy), and an error message indicating the error.

Related Reading: For more information on SJF return and reason codes, see *MVS/ESA Application Development Guide: Authorized Assembler Language Programs*.

The range of some variables is controlled by the initialization parameters. Values for the maximum number of copies, allowable remote destination, classes, and form names are examples of variables influenced by the initialization parameters.

Diagnosis Examples

This section contains examples of mistakes that can generate the various Spool API error codes, and diagnosis of the problems. Some length fields are omitted when

they are not necessary to illustrate the example. The feedback and options lists that are shown on multiple lines are contiguous.

Example 1: Error Code (0002)

```
CALL = SETO
OPTIONS LIST = PRTO=04DEST(018),CLASS(A),TXTU=SET1
FEEDBACK = TXTU(0002)
STATUS CODE = AR
```

Explanation: The options list contains both the keywords PRTO and TXTU. The keyword, TXTU, is not valid for the SETO call.

Example 2: Error Code (0002)

```
CALL = CHNG
OPTIONS LIST = IAFP=N0M,PRTO=0FDEST(018),LINECT(200),CLASS(A),
                                COPIES(80),FORMS(ANS)
FEEDBACK = COPIES(0002),FORMS(0002)
STATUS CODE = AR
```

Explanation: The length field of the PRTO options is too short to contain all of the options. This means that IMS finds the COPIES and FORMS keywords outside the PRTO options list area and indicates that they are invalid on the CHNG call.

Example 3;: Error Code (0004)

```
CALL = CHNG
OPTIONS LIST = IAFP=N0M,OUTN=OUTPUTDD1
FEEDBACK = OUTN(0004)
STATUS CODE = AR
```

Explanation: The operand for the OUTN keyword is 9 bytes long and exceeds the maximum value for the OUTPUT JCL statement.

Example 4;: Error Code (0006)

```
CALL = CHNG
               0400              05
OPTIONS LIST = 0800IAFP=N0M,PRTO=0ADEST(018),LINECT(200),CLASS(A),
                                COPIES(3),FORMS(ANS)
FEEDBACK = PRTO(0006),LINECT(0002),CLASS(0002),COPIES(0002),
                     FORMS(0002)
STATUS CODE = AR
```

Explanation: The length of the options list for this call is too short to contain all of the operands of the PRTO keyword.

This example shows an options list that is X'48' bytes long and is the correct length. The length field of the PRTO keyword incorrectly indicates a length of X'5A'. The length of the PRTO options exceeds the length of the entire options list so IMS ignores the PRTO keyword and scans the rest of the options list for valid keywords. The feedback area contains the PRTO(0006) code (indicating a length error) and the (0002) code (indicating that the PRTO keywords are in error). This is because the keywords beyond the first PRTO keyword, up to the length specified in the options list length field, have been scanned in search of valid keywords for the call. The status code of AR indicates that the keywords are considered invalid for the call and not the PRTO keyword.

Example 5;: Error Code (0008)

```
CALL = CHNG
                              00
OPTIONS LIST = IAFP=N0Z,PRTO=0BDEST(018)
FEEDBACK = IAFP(0008) INVALID VARIABLE
STATUS CODE = AR
```

Explanation: The message option of the IAFP keyword is incorrectly specified as “Z”.

Example 6;: Error Code (000A)

```
CALL = CHNG
OPTIONS LIST = TXTU=SET1
FEEDBACK = TXTU(000A)
STATUS CODE = AR
```

Explanation: The valid keyword TXTU is specified but the call also requires that the IAFP keyword be specified if the TXTU keyword is used.

Example 7;: Error Code (000C)

```
CALL = CHNG
                              00
OPTIONS LIST = IAFP=A00,PRTO=0BCOPIES(3),TXTU=SET1
FEEDBACK = TXTU(000C)
STATUS CODE = AR
```

Explanation: The **AR** status code is returned with the (0002) error code. This implies that the problem is with the call options and not with the PRTO options.

The call options list contains the PRTO and TXTU keywords. These options cannot be used in the same options call list.

Example 8;: Error Code (000E)

```
CALL = CHNG
                              01
OPTIONS LIST = IAFP=A00,PRTO=0BCOPIES((3),(8,RG,18,80))
FEEDBACK = PRTO(000E) (R.C.=0004,REAS.=00000204) COPIES/RG VALUE
                      MUST BE NUMERIC CHARACTERS
STATUS CODE = AS
```

Explanation: The COPIES parameter has the incorrect value “RG” specified as one of its operands. The error message indicates that the values for these operands must be numeric.

Example 9;: Error Code (000E)

```
CALL = CHNG
                              00
OPTIONS LIST = IAFP=A00,PRTO=0AXYZ(018)
FEEDBACK = PRTO(000E) (R.C.=0004,REAS.=000000D0) XYZ
STATUS CODE = AS
```

Explanation: This example includes an invalid PRTO operand. The resulting reason code of X'000000D0' indicates that the XYZ operand is invalid.

Understanding Allocation Errors

The IMS Spool API interface defers dynamic allocation of the print data set until data is actually inserted into the data set. Incorrect data set print options on the CHNG or SETO call can cause errors during dynamic allocation. The print data set options can be parsed during the processing of the CHNG and SETO calls but some things, for example the *destination name* parameter, can be validated only during dynamic allocation.

If one of the print options is incorrect and dynamic allocation fails when the IMS performs the first insert for the data set, IMS returns a **AX** status code to the ISRT call. IMA also issues message DFS0013E and writes a diagnostic log record (67D0) that you can use to evaluate the problem. The format of the error message indicates the type of service that was invoked and the return and reason codes that were responsible for the error. The error message can indicate these services:

DYN MVS/ESA dynamic allocation (SVC99)

OPN MVS/ESA data set open

OUT MVS/ESA dynamic output descriptors build (SVC109)

UNA MVS/ESA dynamic unallocation (SVC99)

WRT MVS/ESA BSAM write

If the DFS0013E message indicates an error return code from any of these services, you should consult the corresponding MVS/ESA documentation for more information on the error code. If the service is for dynamic allocation, dynamic unallocation, or dynamic output descriptor build, see *MVS/ESA Programming: Authorized Assembler Services Guide* for the appropriate return and reason codes.

One common mistake is the use of an invalid destination or selection of integrity option 2 (non-selectable destination) when the destination of IMSTEMP has not been defined to JES. If you specify an invalid destination in the *destination name* parameter, the call will result in a dynamic unallocation error when IMS unallocates the print data set.

Understanding Dynamic Output for Print Data Sets

IMS can use the MVS/ESA services for Dynamic Output (SVC109) for print data sets. IMS uses this service to specify the attributes provided by the application for the print data sets being created. The service can be used on the CHNG call with the PRTO, TXTU, and OUTN options.

Related Reading: For more information, see *MVS/ESA Programming: Authorized Assembler Services Guide*.

CHNG Call with PRTO Option

When you use the CHNG call and PRTO option, IMS activates SJF to verify the print options to call MVS/ESA services for Dynamic Output. This creates the output descriptors that are used when the print data set is allocated. This is the simplest way for the application to provide print data set characteristics. However, it also uses the most overhead because parsing must occur for each CHNG call. If your application is WFI or creates multiple data sets with the same print options, use another option to reduce the parsing impact.

CHNG Call with TXTU Option

If your application can manage the text units necessary for Dynamic Output, then you can avoid parsing for many of the print data sets. You can do this in one of two ways:

- The application can build the text unit in the necessary format within the application area and pass these text units to IMS with the CHNG call and TXTU option.
- The application can provide the print options to IMS with a SETO call and provide a work area for the construction of the text units. After MVS/ESA has finished parsing and text construction, the work area passed will contain the text units necessary for Dynamic Output after a successful SETO call. The application must not relocate this work area because the work area contains address sensitive information.

Regardless of the method the application uses to manage the text units, applications that can reuse the text units can often achieve better performance by using the TXTU option on the CHNG call.

CHNG Call with OUTN Option

The dependent region JCL can contain OUTPUT JCL statements. If your application can use this method, you can use the CHNG call and OUTN option to reference OUTPUT JCL statements. When you use the OUTN option, IMS will reference the OUTPUT JCL statements at dynamic allocation. JES will obtain the print data set characteristics from the OUTPUT JCL statement.

Sample Program Using the Spool API

The Spool API provides functions that allow an application program to write data to the IMS Spool using the same techniques for sending data to native IMS printers.

The Spool API provides functions such as error checking for invalid OUTDES parameters. Error checking makes application programs more complex. To simplify these application programs, develop a common routine to manage error information, then make the diagnostic information from the Spool API available for problem determination.

The following sample programs show how DL/I calls can be coded to send data to the IMS Spool. Only the parts of the application program necessary to understand the DL/I call formats are included. The examples are in assembler language.

Application PCB Structure

The application PCBs are as follows:

I/O PCB

ALTPCB1

ALTPCB2

ALTPCB3

ALTPCB4

GU Call to I/O PCB

IMS application programs begin with initialization and a call to the I/O PCB to obtain the input message. The example in Figure 83 shows how to issue a GU call to the I/O PCB:

```
*********************************************************************
*       ISSUE GU ON IOPCB                                           *
*********************************************************************
         L     9,IOPCB             I/O PCB ADDRESS
         LA    9,0(9)
         MVC   FUNC,=CL4'GU'       GU FUNCTION
         CALL  ASMTDLI,(FUNC,(9),IOA1),VL
         BAL   10,STATUS           CHECK STATUS
*        ADDITIONAL PROGRAM LOGIC HERE
FUNC     DC    CL4' '
IOA1     DC    AL2(IOA1LEN),AL2(0)
TRAN     DS    CL8                 TRANSACTION CODE AREA
DATA     DS    CL5                 DATA STARTS HERE
         DC    20F'0'
IOA1LEN  EQU   *-IOA1
```

Figure 83. Issuing a GU Call to the I/O PCB

After completing the GU call to the I/O PCB, the application program prepares output data for the IMS Spool.

CHNG Call to Alternate PCB

In the same way that other programs specify the destination of the output using the CHNG call, this program specifies the IMS Spool as the output destination. For a native IMS printer, the DEST NAME parameter identifies the output LTERM name. When a CHNG call is issued that contains the IAFP= keyword, the DEST NAME parameter is used only if integrity option '2' is specified. If option '2' is not specified, the DEST NAME parameter can be used by the application program to identify something else, such as the routine producing the change call. The destination for the print data set is established using a combination of initialization parameters or OUTDES parameters.

The example in Figure 84 shows how to issue a CHNG call to the alternate modifiable PCB.

```
*********************************************************************
*       ISSUE CHNG ON ALTPCB4                                       *
*********************************************************************
         L     9,ALTPCB4           ALT MODIFIABLE PCB
         LA    9,0(9)              CLEAR HIGH BYTE/BIT
         MVC   FUNC,=CL4'CHNG'     CHNG FUNCTION
         CALL  ASMTDLI,(FUNC,(9),DEST2,OPT1,FBA1),VL
         BAL   10,STATUS           CHECK STATUS OF CALL
*        ADDITIONAL PROGRAM LOGIC HERE
FUNC     DC    CL4' '
DEST2    DC    CL8'IAFP1'          LTERM NAME
*
         DC    C'OPT1'             OPTIONS LIST AREA
OPT1     DC    AL2(OPT1LEN),AL2(0)
         DC    C'IAFP='
OCC      DC    C'M'                DEFAULT TO MACHINE CHAR
OOPT     DC    C'1'                DEFAULT TO HOLD
OMSG     DC    C'M'                DEFAULT TO ISSUE MSG
         DC    C','
         DC    C'PRTO='
PRTO1    EQU   *
         DC    AL2(PRTO1LEN)
         DC    C'COPIES(2),CLASS(T),DEST(RMT003)'
PRTO1LEN EQU   *-PRTO1
         DC    C' '
OPT1LEN  EQU   *-OPT1
*
FBA1     DC    AL2(FBA1LEN),AL2(0)
         DC    CL40' '
FBA1LEN  EQU   *-FBA1
```

Figure 84. Issuing a CHNG Call to the Alternate Modifiable PCB

After the CHNG call is issued, the application program creates the print data set by issuing ISRT calls.

ISRT Call to Alternate PCB

Once the IMS Spool is specified as the destination of the PCB, ISRT calls can be issued against the alternate PCB.

The example in Figure 85 shows how to issue the ISRT call to the alternate modifiable PCB.

```
*********************************************************************
*       ISSUE ISRT TO ALTPCB4                                       *
*********************************************************************
         L     9,ALTPCB4           ALT MODIFIABLE PCB
         LA    9,0(9)              CLEAR HIGH BYTE/BIT
         MVC   FUNC,=CL4'ISRT'     ISRT FUNCTION
         CALL  ASMTDLI,(FUNC,(9),IOA2),VL
         BAL   10,STATUS           CHECK STATUS OF CALL
*        ADDITIONAL PROGRAM LOGIC HERE
FUNC     DC    CL4' '
IOA2     DC    AL2(IOA2LEN),AL2(0)
IOA21    DC    AL2(MSG2LEN),AL2(0)
         DC    C' '                CONTROL CHARACTER
         DC    C'MESSAGE TO SEND TO IMS SPOOL'
MSG2LEN  EQU   *-IOA21
IOA2LEN  EQU   *-IOA2
```

Figure 85. Issuing an ISRT Call to the Alternate Modifiable PCB

The print data streams can be stored in databases or generated by the application, depending on the requirements of the application program and the type of data set being created.

Program Termination

After the calls are issued, the program sends a message back to originating terminal, issues a GU call to the I/O PCB, or terminates normally.

Appendix E. Using the DL/I Test Program (DFSDDLT0)

DFSDDLT0 is an IMS application program test tool that issues calls to IMS based on control statement information. You can use it to verify and debug DL/I calls independently of application programs. You can run DFSDDLT0 using any PSB, including those that use an IMS-supported language. You can also use DFSDDLT0 as a general-purpose database utility program.

The functions that DFSDDLT0 provides include:

- Issuing any valid DL/I call against any database using:
  - Any segment search argument (SSA) or PCB, or both
  - Any SSA or AIB, or both
- Comparing the results of a call to expected results. This includes the contents of selected PCB fields, the data returned in the I/O area, or both.
- Printing the control statements, the results of calls, and the results of comparisons only when the output is useful, such as after an unequal compare.
- Dumping DL/I control blocks, the I/O buffer pool, or the entire batch region.
- Punching selected control statements into an output file to create new test data sets. This simplifies the construction of new test cases.
- Merging multiple input data sets into a single input data set using a SYSIN2 DD statement in the JCL. You can specify the final order of the merged statements in columns 73 to 80 of the DFSDDLT0 control statements.
- Sending messages to the MVS system console (with or without a reply).
- Repeating each call up to 9,999 times.

Control Statements

DFSDDLT0 processes control statements to control the test environment. DFSDDLT0 can issue calls to IMS full-function databases and Fast Path databases, as well as DC calls. Table 63 gives an alphabetical summary of the types of control statements DFSDDLT0 uses. A detailed description of each type of statement follows.

Table 63. Summary of DFSDDLT0 Control Statements

Control Statement	Code	Description
ABEND[1]	ABEND	Causes user abend 252.
CALL	L	• CALL FUNCTION identifies the type of IMS call function to be made and supplies information to be used by the call. • CALL DATA provides IMS with additional information.
COMMENT		There are two types of COMMENT statements:
	T	Conditional allows a limited number of comments that are printed or not depending on how the STATUS statement is coded and the results of the PCB or DATA COMPARE.
	U[11]	Unconditional allows an nlimited number of comments, all of which are printed.

© Copyright IBM Corp. 1974, 2000

Table 63. Summary of DFSDDLT0 Control Statements (continued)

Control Statement	Code	Description
COMPARE	E	There are three types of COMPARE statements: COMPARE DATA verifies that the correct segment was retrieved by comparing the segment returned by IMS with data in this statement. COMPARE AIB compares values that IMS returns to the AIB. COMPARE PCB checks fields in the PCB and calls for a snap dump of the DL/I blocks, the I/O buffer pool, or the batch region if the compare is unequal.
IGNORE	N or .	The program ignores statements that contain N or . (period) in column 1.
OPTION[1]	O	Shows which control blocks are to be dumped, the number of unequal comparisons allowed, whether dumps are produced, number of lines printed per page, and the SPA size.
PUNCH[1]	CTL	PUNCH CTL produces an output data set consisting of the COMPARE PCB statements, the COMPARE AIB statements, the DATA statements, and all other control statements read.
STATUS[1]	S	Establishes print options and selects the PCB or AIB against which subsequent calls are to be issued.
WTO[1]	WTO	Sends a message to the MVS console without waiting for reply.
WTOR[1]	WTOR	Sends a message to the MVS console and waits for a reply before proceeding.

Note:

1. These control statements are acted on immediately when encountered in an input stream. Do not code them where they will interrupt call sequences. (See "Planning the Control Statement Order" on page 527.)

The control statements are further described below:

- The CALL statement is the central DFSDDLT0 statement. The CALL statement has two parts: CALL FUNCTION and CALL DATA. CALL FUNCTION identifies the type of IMS call function and supplies information about segment search arguments (SSAs). CALL DATA provides more information required for the type of call identified by CALL FUNCTION.
- The STATUS statement controls the PCB, AIB, and handling of output.
- The three types of COMPARE statements, DATA, PCB, and AIB, compare different values:
  - If you want specific data from a call, use COMPARE DATA to check the segment data for mismatches when the call is made.
  - Use COMPARE PCB to check status codes, segment levels, and feedback keys. It also indicates mismatches when you specify output.
  - Use COMPARE AIB to compare values that IMS returns to the AIB.
- The two COMMENT statements, Conditional and Unconditional, allow you to set limits on the number of comments on the DFSDDLT0 job stream and to specify whether you want the comments printed.

- The OPTION statement controls several overall functions such as the number of unequal comparisons allowed and the number of lines printed per page.
- The remaining statements, ABEND, IGNORE, CTL, WTO and WTOR, are not as important as the others at first. Read the sections describing these statements so that you can become familiar with the functions they offer.

When you are coding the DFSDDLT0 control statements, keep the following items in mind:

- If you need to temporarily override certain control statements in the DFSDDLT0 streams, read about SYSIN/SYSIN2 processing in the JCL requirements section on page 565.
- You must fill in column 1 of each control statement. If column 1 is blank, the statement type defaults to the prior statement type. DFSDDLT0 attempts to use any remaining characters as it would for the prior statement type.
- Use of reserved fields can produce invalid output and unpredictable results.
- Statement continuations are important, especially for the CALL statement.
- Sequence numbers are not required, but they can be very useful for some DFSDDLT0 functions. Read about the PUNCH CTL statement starting on page 557 and the SYSIN/SYSIN2 processing description in the JCL requirements section on page 565 to understand how to use sequence numbers.
- All codes and fields in the DFSDDLTO statements must be left justified followed by blanks, unless otherwise specified.

Planning the Control Statement Order

The order of control statements is *critical* in constructing a successful call. To avoid unpredictable results, follow these guidelines:

1. If you are using STATUS and OPTION statements, place them somewhere before the calls that are to use them.
2. Both types of COMMENT statements are optional but, if present, must appear before the call they document.
3. You must code CALL FUNCTION statements and any required SSAs consecutively without interruption.
4. CALL DATA statements must immediately follow the last continuation, if any, of the CALL FUNCTION statements.
5. COMPARE statements are optional but must follow the last CALL (FUNCTION or DATA) statement.
6. When CALL FUNCTION statements, CALL DATA statements, COMPARE DATA statements, COMPARE PCB statements, and COMPARE AIB statements are coded together, they form a call sequence. *Do not* interrupt call sequences with other DFSDDLT0 control statements.

   Exception: IGNORE statements are the only exception to this rule.
7. Use IGNORE statements (N or .) to override any statement, regardless of its position in the input stream. You can use IGNORE statements in either SYSIN or SYSIN2 input streams.

ABEND Statement

The ABEND statement causes IMS to issue an abend and terminate DFSDDLT0. Table 64 shows the format of the ABEND statement.

Table 64. ABEND Statement

Column	Function	Code	Description
1-5	Identifies control statement	**ABEND**	Issues abend U252. (No dump is produced unless you code DUMP on the OPTION statement.)
6-72	Reserved	**b**	
73-80	Sequence indication	**nnnnnnnn**	For SYSIN2 statement override.

Examples of ABEND Statement

If you use ABEND in the input stream and want a dump, you must specify DUMP on the OPTION statement. The default on the OPTION statement is NODUMP.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
ABEND                                                                 22100010
```

Dump will be produced; OPTION statement provided requests dump.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
O DUMP                                                                22100010
```

No dump will be produced; OPTION statement provided requests NODUMP.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
O NODUMP                                                              22100010
```

CALL Statement

The CALL control statement has two parts: CALL FUNCTION and CALL DATA.

- The CALL FUNCTION statement supplies the DL/I call function, the segment search arguments (SSAs), and the number of times to repeat the call. SSAs are coded according to IMS standards.
- With the CALL DATA statement you provide any data (database segments, MVS commands, checkpoint IDs) required by the DL/I call specified in the CALL FUNCTION statement. See “CALL DATA Statement” on page 531.

CALL FUNCTION Statement

Table 65 gives the format for CALL FUNCTION statements. This is the preferred format when you are not working with column-specific SSAs.

Table 65. CALL FUNCTION Statement

Column	Function	Code	Description
1	Identifies control statement	L	Issues an IMS call
2	Reserved	**b**	
3	SSA level	**b**	SSA level (optional)
		n	Range of hexademical characters allowed is 1-F
4	Reserved	**b**	

Table 65. CALL FUNCTION Statement (continued)

Column	Function	Code	Description
5-8	Repeat count	**ƀƀƀƀ**	If blank, repeat count defaults to 1.
		nnnn	'nnnn' is the number of times to repeat this call. Range is 1 to 9999, right-justified, with or without leading zeros.
9	Reserved	**ƀ**	ƀ
10-13	Identifies DL/I call function	**ƀƀƀƀ**	If blank, use function from previous CALL statement.
		xxxx	'xxxx' is a DL/I call function.
	Continue SSA	CONT	Continuation indicator for SSAs too long for a single CALL FUNCTION statement. Column 72 of the preceding CALL FUNCTION statement must have an entry. The next CALL statement should have CONT in columns 10 - 13 and the SSA should continue in column 16.
14-15	Reserved	**ƀ**	
16-23 or	SSA name	**xxxxxxxx**	Must be left-justified.
16-23 or	Token	**xxxxxxxx**	Token name (SETS/ROLS).
16-23 or	MOD name	**xxxxxxxx**	Modname (PURG+ISRT).
16-23 or	Subfunction	**xxxxxxxx**	nulls, DBQUERY, FIND, ENVIRON, PROGRAM (INQY).
16-19 and	Statistics type	**xxxx**	DBAS/DBES-OSAM or VBAS/VBES-VSAM (STAT).[2]
20 or	Statistics format	**x**	F - Formatted U- Unformatted S - Summary.
16–19	SETO ID[1]	**SETx**	Where x is 1, 2, or 3. Specified on SETO and CHNG calls as defined in Note.

Table 65. CALL FUNCTION Statement (continued)

Column	Function	Code	Description
21-24	SETO IOAREA SIZE	**nnnn**	Value of 0000 to 8192. If a value greater than 8192 is specified, it defaults to 8192. If no value is specified, the call is made with no SETO size specified.
24–71	Remainder of SSA		Unqualified SSAs must be blank. Qualified search arguments should have either an '*' or a '(' in column 24 and follow IMS SSA coding conventions.
72	Continuation column	**b**	No continuations for this statement.
		x	Alone, it indicates multiple SSAs each beginning in column 16 of successive statements. With CONT in columns 10-13 of the next statement, indicates a single SSA that is continued beginning in column 16 of the following statement.
73-80	Sequence indication	nnnnnnnn	For SYSIN2 statement override.

Note:

1. SETO CALL:

   The SETO ID (SET1, SET2, or SET3) is required on the SETO call if DFSDDLT0 is to keep track of the text unit address returned on the SETO call that would be passed on the CHNG call for option parameter TXTU.

   If the SETO ID is omitted on the SETO call, DFSDDLT0 does not keep track of the data returned and is unable to reference it on a CHNG call.

   CHNG CALL:

   The SETO ID (SET1, SET2, or SET3) is required on the CHNG call if DFSDDLT0 is to place the address of the SETO ID I/O area returned on the SETO call. This is the SETO call of the text unit returned on the SETO call with a matching SETO ID for this CHNG call into the “TXTU=ADDR” field of the option parameter in the CHNG call.

   When the SETO ID is specified on the CHNG call, DFSDDLT0 moves the address of that text unit returned on the SETO call using the same SETO ID.

   Code the OPTION statement parameter TXTU as follows: TXTU=xxxx where xxxx is any valid non-blank character. It cannot be a single quote character.

   Suggested value for xxxx could be SET1, SET2, or SET3. This value is not used by DFSDDLT0.

2. STAT is a Product-sensitive programming interface.

The following information applies to different types of continuations:

- Column 3, the SSA level, is usually blank. If it is blank, the first CALL FUNCTION statement fills SSA 1, and each following CALL FUNCTION statement fills the

next lower SSA. If column 3 is not blank, the statement fills the SSA at that level, and the following CALL FUNCTION statement fills the next lower one.

- Columns 5 through 8 are usually blank, but if used, must be right justified. The same call is repeated as specified by the repeat call function.
- Columns 10 through 13 contain the DL/I call function. The call function is required only for the first CALL FUNCTION statement when multiple SSAs are in a call. If left blank, the call function from the previous CALL FUNCTION statement is used.
- Columns 16 through 23 contain the segment name if the call uses an SSA.
- If the DL/I call contains multiple SSAs, the statement must have a nonblank character in column 72, and the next SSA must start in column 16 of the next statement. The data in columns 1 and 10 through 13 are blank for the second through last SSAs.

  Restriction: On ISRT calls, the last SSA can have only the segment name with no qualification or continuation.
- If a field value extends past column 71, put a nonblank character in column 72. (This character is not read as part of the field value, only as a continuation character.) In the next statement insert the keyword CONT in columns 10 through 13 and continue the field value starting at column 16.
- Maximum length for the field value is 256 bytes, maximum size for an SSA is 290 bytes, and the maximum number of SSAs for this program is 15, which is the same as the IMS limit.
- If columns 5 through 8 in the CALL FUNCTION statement contain a repeat count for the call, the call will terminate when reaching that count, unless it first encounters a GB status code.

Related Reading: See “CALL FUNCTION Statement with Column-Specific SSAs” on page 546 for another format supported by DFSDDLT0.

CALL DATA Statement

CALL DATA statements provide IMS with information normally supplied in the I/O area for that type of call function.

CALL DATA statements must follow the last CALL FUNCTION statement. You must enter an L in column 1, the keyword DATA in columns 10 thru 13, and code the necessary data in columns 16 thru 71. You can continue data by entering a nonblank character in column 72. On the continuation statement, columns 1 thru 15 are blank and the data resumes in column 16. Table 66 shows the format for a CALL DATA statement.

Table 66. CALL DATA Statement

Column	Function	Code	Description
1	Identifies control statement	**L**	CALL DATA statement.
2	Increase segment length	**K**	Adds 2500 bytes to the length of data defined in columns 5 through 8.
3	Propagate remaining I/O indicator	**P**	Causes 50 bytes (columns 16 thru 65) to be propagated through remaining I/O area. **Note:** This must be the last data statement and cannot be continued.
4	Format options	**b**	Not a variable-length segment.

Table 66. CALL DATA Statement (continued)

Column	Function	Code	Description
		V	For the first statement describing the only variable-length segment or the first variable-length segment of multiple variable-length segments, LL field is added before the segment data.
		M	For statements describing the second through the last variable-length segments, LL field is added before the segment data.
		P	For the first statement describing a fixed-length segment in a path call.
		Z	For message segment, LLZZ field is added before the data.
		U	Undefined record format for GSAM records. The length of segment for an ISRT is placed in the DB PCB key feedback area.
5-8	Length of data in segment	**nnnn**	This value must be right justified but need not contain leading zeros. If you do not specify a length, DFSDDLT0 will use the number of DATA statements read multiplied by 56 to derive the length.
9	Reserved	**b**	
10-13	Identifies CALL DATA statement	**DATA**	Identifies this as a DATA statement.
14-15	Reserved	**b**	
16-71 or	Data area	**xxxx**	Data that goes in the I/O area.
16-23 or	Checkpoint ID		Checkpoint ID (SYNC).
16-23 or	Destination name		Destination name (CHNG).
16	DEQ option		DEQ options (A,B,C,D,E,F,G,H,I, or J).
72	Continuation column	**b**	If no more continuations for this segment.
		x	If more data for this segment or more segments.
73-80	Sequence indication	**nnnnnnnn**	For SYSIN2 statement override.

When inserting variable-length segments or including variable-length data for a CHKP or LOG call:

- You must use a V or M in column 4 of the CALL DATA statement.
- Use V if only one variable-length segment is being processed.
- You must enter the length of the data with leading zeros, right justified, in columns 5 through 8. The value is converted to binary and becomes the first 2 bytes of the segment data.
- You can continue a CALL DATA statement into the next CALL DATA statement by entering a nonblank character in column 72. For subsequent statements, leave columns 1 through 15 blank, and start the data in column 16.

If multiple variable-length segments are required (that is, concatenated logical child/logical parent segments, both of which are variable-length) for the first segment:

- You must enter a V in column 4.
- You must enter the length of the first segment in columns 5 through 8.
- If the first segment is longer than 56 bytes, continue the data as described for inserting variable-length segments.

  Exceptions:

  – The last CALL DATA statement to contain data for this segment must have a nonblank character in column 72.

  – The next CALL DATA statement applies to the next variable-length statement and must contain an M in column 4 and the length of the segment in columns 5 through 8.

You can concatenate any number of variable-length segments in this manner. Enter M or V and the length (only in CALL DATA statements that begin data for a variable-length segment).

When a program is inserting or replacing through path calls:

- Enter a P in column 4 to specify that the length field is to be used as the length the segment will occupy in the user I/O area.
- You only need to use P in the first statement of fixed-length-segment CALL DATA statements in path calls that contain both variable- and fixed-length segments.
- You can use V, M, and P in successive CALL DATA statements.

For INIT, SETS, ROLS, and LOG calls:

- The format of the I/O area is

  ```
  LLZZuser-data
  ```

  where LL is the length of the data in the I/O area, including the length of the LLZZ portion.
- If you want the program to use this format for the I/O area, enter a Z in column 4 and the length of the data in columns 5 through 8. The length in columns 5 through 8 is the length of the data, not including the 4-byte length of LLZZ.

OPTION DATA Statement

The OPTION DATA statement contains options as required for SETO and CHNG calls.

Table 67 shows the format for an OPTION DATA statement.

Table 67. OPTION DATA Statement

Column	Function	Code	Description
1	Identifies control statement	**L**	OPTION statement.
2-9	Reserved	**b**	
10-13	Identifies	**OPT**	Identifies this as OPTION statement.
		CONT	Identifies this as a continuation of an option input.
14-15	Reserved	**b**	
16-71	Option area	**xxxx**	Options as defined for SETO and CHNG call.

Table 67. OPTION DATA Statement (continued)

Column	Function	Code	Description
72	Continuation column	**b**	If no more continuations for options.
		x	If more option data exists in following statement.
73-80	Sequence number	**nnnnnnnn**	For SYSIN2 statement override.

FEEDBACK DATA Statement

The FEEDBACK DATA statement defines an area to contain feedback data.

The FEEDBACK DATA statement is optional. However, if the FEEDBACK DATA statement is used, an OPTION DATA statement is required.

Table 68 shows the format for a FEEDBACK DATA statement.

Table 68. FEEDBACK DATA Statement

Column	Function	Code	Description
1	Identifies control statement	**L**	FEEDBACK statement.
2-3	Reserved	**b**	
4	Format option	**b**	Feedback area contains LLZZ.
		Z	Length of feedback area will be computed and the LLZZ will be added to the feedback area.
5-8	Length of feedback area	**nnnn**	This value must be right justified but need not contain leading zeros. If you do not specify a length, DFSDDLT0 uses the number of FDBK inputs read multiplied by 56 to derive the length.
2-9	Reserved	**b**	
10-13	Identifies	**FDBK**	Identifies this as feedback statement and continuation of feedback statement.
14-15	Reserved	**b**	
16-71	Feedback area	**xxxx**	Contains user pre-defined initialized area.
72	Continuation column	**b**	If no more continuations for feedback.
		x	If more feedback data exists in following statement.
73-80	Sequence number	**nnnnnnnn**	For SYSIN2 statement override.

Call Functions

DL/I Call Functions

Table 69 shows the DL/I call functions supported in DFSDDLT0 and which ones require data statements.

Table 69. DL/I Call Functions

Call	AIB Support	PCB Support	Data Stmt[1]	Description
CHKP	yes	yes	R	Checkpoint.

Table 69. DL/I Call Functions (continued)

Call	AIB Support	PCB Support	Data Stmt[1]	Description
CHNG	yes	yes	R	Change alternate PCB.
			R	Contains the alternate PCB name option statement and feedback statement optional.
CMD	yes	yes	R	Issue IMS command. This call defaults to I/O PCB.
DEQ	yes	yes	R	Dequeue segments locked with the Q command code. For full function, this call defaults to the I/O PCB, provided a DATA statement containing the class to dequeue immediately follows the call. For Fast Path, the call is issued against a DEDB PCB. Do not include a DATA statement immediately following the DEQ call.
DLET	yes	yes	O	Delete. If the data statement is present, it is used. If not, the call uses the data from the previous Get Hold Unique (GHU).
FLD	yes	yes	R	Field—for Fast Path MSDB calls using FSAs. This call references MSDBs only. If there is more than one FSA, put a nonblank character in column 34, and put the next FSA in columns 16-34 of the next statement. A DATA statement containing FSA is required.
GCMD	yes	yes	N	Get command response. This call defaults to I/O PCB.
GHN	yes	yes	O[2]	Get Hold Next.
GHNP	yes	yes	O[2]	Get Hold Next in Parent.
GHU	yes	yes	O[2]	Get Hold Unique.
GMSG[3]	yes	no	R	Get Message is used in an automated operator (AO) application program to retrieve a message from AO exit routine DFSAOE00. The DATA statement is required to allow for area in which to return data. The area must be large enough to hold this returned data.
GN	yes	yes	O[2]	Get Next segment.
GNP	yes	yes	O[2]	Get Next in Parent.
GU	yes	yes	O[2]	Get Unique segment.
ICMD[3]	yes	no	R	Issue Command enables an automated operator (AO) application program to issue an IMS command and retrieve the first command response segment. The DATA statement is required to contain the input command and to allow for area in which to return data. The area must be large enough to hold this returned data.
INIT	yes	yes	R	Initialization This call defaults to I/O PCB. A DATA statement is required. Use the LLZZ format.
INQY[3]	yes	no	R	Request environment information using the AIB and the ENVIRON subfunction. The DATA statement is required to allow for area in which to return data. The area must be large enough to hold this returned data.
			R	Request database information using the AIB and the DBQUERY subfunction, which is equivalent to the INIT DBQUERY call. The DATA statement is required to allow for area in which to return data. The area must be large enough to hold this returned data.

Table 69. DL/I Call Functions (continued)

Call	AIB Support	PCB Support	Data Stmt[1]	Description
ISRT	yes	yes		Insert.
			R	DB PCB, DATA statement required.
			O	I/O PCB using I/O area with MOD name, if any, in columns 16-23.
			R	Alt PCB.
LOG	yes	yes	R	Log system request. This call defaults to I/O PCB. DATA statement is required and can be specified in one of two ways.
POS	yes	yes	N	Position - for DEDBs to determine a segment location. This call references DEDBs only.
PURG	yes	yes		Purge.
			R	This call defaults to use I/O PCB. If column 16 is not blank, MOD (message output descriptor) name is used and a DATA statement is required.
			O	If column 16 is blank, the DATA statement is optional.
RCMD[3]	yes	no	R	Retrieve Command enables an automated operator (AO) application program to retrieve the second and subsequent command response segments after an ICMD call. The DATA statement is required to allow for area in which to return data. The area must be large enough to hold this returned data.
REPL	yes	yes	R	Replace—This call references DB PCBs only. The DATA statement is required.
ROLB	yes	yes	O	Roll Back call.
ROLL	no	yes	O	Roll Back call and issue U778 abend.
ROLS	yes	yes	O	Back out updates and issue 3303 abend. Uses the I/O PCB. Can be used with the SETS call function. To issue a ROLS with an I/O area and token as the fourth parameter, specify the 4-byte token in column 16 of the CALL statement. Leaving columns 16-19 blank will cause the call to be made without the I/O area and the token. (To issue a ROLS using the current DB PCB, use ROLX.)
ROLX	yes	yes	O	Roll call against the DB PCB (DFSDDLT0 call function). This call is used to request a Roll Back call to DB PCB, and is changed to ROLS call when making the DL/I call.
SETO	yes	yes	N	Set options. OPTION statement required. FEEDBACK statement optional.
SETS/SETU	yes	yes	O	Create or cancel intermediate backout points. Uses I/O PCB. To issue a SETS with an I/O area and token as the fourth parameter, specify the four-byte token in column 16 of the CALL statement and include a DATA statement. Leaving columns 16-19 blank will cause the call to be made without the I/O area and the token.

Table 69. DL/I Call Functions (continued)

Call	AIB Support	PCB Support	Data Stmt[1]	Description
SNAP[4]	yes	yes	O	Sets the identification and destination for snap dumps. If a SNAP call is issued without a CALL DATA statement, a snap of the I/O buffer pools and control blocks will be taken and sent to LOG if online and to PRINTDD DCB if batch. The SNAP ID will default to SNAPxxxx where xxxx starts at 0000 and is incremented by 1 for every SNAP call without a DATA statement. The SNAP options default to YYYN. If a CALL DATA statement is used, columns 16-23 specify the SNAP destination, columns 24-31 specify the SNAP identification, and columns 32-35 specify the SNAP options. SNAP options are coded using 'Y' to request a snap dump and 'N' to prevent it. Column 32 snaps the I/O buffer pools, columns 33 and 34 snap the IMS control blocks and column 35 snaps the entire region. The SNAP call function is only supported for full-function database PCB.
STAT[5]	yes	yes	O	The STAT call retrieves statistics on the IMS system. This call must reference only full-function DB PCBs. See the examples on 545. Statistics type is coded in columns 16-19 of the CALL FUNCTION statement. **DBAS** For OSAM database buffer pool statistics. **VBAS** For VSAM database subpool statistics. Statistics format is coded in column 20 of the CALL FUNCTION statement. **F** For the full statistics to be formatted if F is specified, the I/O area must be at least 360 bytes. **U** For the full statistics to be unformatted if U is specified, the I/O area must be at least 72 bytes. **S** For a summary of the statistics to be formatted if S is specified, the I/O area must be at least 120 bytes.
SYNC	yes	yes	R	Synchronization.
XRST	yes	yes	R	Restart.

Notes:

1. R = required; O = optional; N = none
2. The data statement is required on the AIB interface.
3. Valid only on the AIB interface.
4. SNAP is a Product-sensitive programming interface.
5. STAT is a Product-sensitive programming interface.

Examples of DL/I Call Functions

Basic CHKP Call: Use a CALL FUNCTION statement to contain the CHKP function and a CALL DATA statement to contain the checkpoint ID.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        CHKP                                                        10101400
L        DATA  TESTCKPT
```

Symbolic CHKP Call with Two Data Areas to Checkpoint: Use a CALL FUNCTION statement to contain the CHKP function, a CALL DATA statement to contain the checkpoint ID data, and two CALL DATA statements to contain the data that you want to checkpoint.

You also need to use an XRST call when you use the symbolic CHKP call. Prior usage of an XRST call is required when using the symbolic CHKP call, as the CHKP call keys on the XRST call for symbolic CHKP.

Recommendation: Issue an XRST call as the first call in the application program.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        XRST
L        .
L        .
L        .
L        CHKP
L        DATA  TSTCHKP2                                                  X
L      8 DATA  STRING2-                                                  X
L     16 DATA  STRING2-STRING2-
U EIGHT BYTES OF DATA (STRING2-) IS CHECKPOINTED AND
U SIXTEEN BYTES OF DATA (STRING2-STRING2-) IS CHECKPOINTED ALSO
```

CHNG Call: Use a CALL FUNCTION statement to contain the CHNG function and a CALL DATA statement to contain the new logical terminal name.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        CHNG SET1
L        OPT  IAFP=A1M,PRTO=LLOPTION1,OPTION2,
L        CONT OPTION4
L Z0023  DATA DESTNAME
```

LL is the hex value of the length of LLOPTION,.....OPTION4.

The following is an example of a CHNG statement using SETO ID SET2, OPTION statement, DATA statement with MODNAME, and FDBK statement.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        CHNG SET2
L        OPT  IAFP=A1M,TXTU=SET2
L Z0023  DATA DESTNAME
L Z0095  FDBK FEEDBACK AREA
```

CMD Call: Use a CALL FUNCTION statement to contain the CMD function and a CALL DATA statement to contain the Command data.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        CMD
L  ZXXXX DATA  COMMAND DATA

WHERE XXXX = THE LENGTH OF THE COMMAND DATA
```

DEQ Call: For full function, use a CALL FUNCTION statement to contain the DEQ function and a CALL DATA statement to contain the DEQ value (A,B,C,D,E,F,G,H,I or J).

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        DEQ
L        DATA  A
```

For Fast Path, use a CALL FUNCTION statement to contain the DEQ function.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        DEQ
```

DLET Call: Use a CALL FUNCTION statement to contain the DLET function. The data statement is optional. If there are intervening calls to other PCBs between the Get Hold call and the DLET call, you must use a data statement to refresh the I/O area with the segment to be deleted.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        DLET
```

FLD Call: Use a CALL FUNCTION statement to contain the FLD function and ROOTSSA, and a CALL DATA statement to contain the FSAs.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        FLD   ROOTA   (KEYA     =ROOTA)
L        DATA  ???????                                              X
L        DATA
```

GCMD Call: Use a CALL FUNCTION statement to contain the GCMD function; no CALL DATA statement is required.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        GCMD
```

GHN Call: Use a CALL FUNCTION statement to contain the GHN function; no CALL DATA statement is required.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        GHN                                                        10103210
```

GHNP Call: Use a CALL FUNCTION statement to contain the GHNP function; no CALL DATA statement is required.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        GHNP                                                       10103210
```

GHU Call with a Continued SSA: Use two CALL FUNCTION statements to contain the single SSA.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        GHU   SEGG    (FILLRG  = G131G131G131G131G131G131G131G131G131G*
         CONT  131G131G131G131G131G131G131        )
```

GMSG Call: Use a CALL FUNCTION statement to contain the GMSG function. Use a CALL DATA statement to retrieve messages from AO exit routine.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        GMSG  TOKEN111 WAITAOI
L  Z0132 DATA
L        GMSG
L  Z0132 DATA
```

GN Call: Use a CALL FUNCTION statement to contain the GN function; no CALL DATA statement is required.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        GN                                                          10103210
```

GNP Call: Use a CALL FUNCTION statement to contain the GNP function; no CALL DATA statement is required.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        GNP                                                         10103210
```

GU Call with a Single SSA and a Relational Operator: Use a CALL FUNCTION statement to contain the GU function; no CALL DATA statement is required. The qualified SSA begins in column 24 and is contained in parentheses.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        GU    SEGF    (KEYF    > F131*KEYF    < F400)
```

GU Call with a Single SSA and a Relational Operator Extended Across Multiple Inputs with Boolean Operators: Use a CALL FUNCTION statement to contain the GU function and three additional continuation of CALL FUNCTION input to continue with Boolean operators. No CALL DATA statement is required. The qualified SSA begins in column 24 and is contained in parentheses. This type of SSA can continue over several statements.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        GU    SEGG    (FILLRG  > G131G131G131G131G131G131G131G131G131G131G*
         CONT  131G131G131G131G131G131G131        &FILLRG  < G400G400G4*
         CONT  00G400G400G400G400G400G400G400G400G400G400G400G400G400  *
         CONT        )
```

GU Path Call: Use a CALL FUNCTION statement to contain the GU function and three additional continuation of CALL function input to continue with two additional SSAs. No CALL DATA statement is required. The call uses command codes in columns 24 and 25 to construct the path call. This type of call cannot be made with the column-specific SSA format.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        GU    SEGA    *D(KEYA    = A200)                              *
               SEGF    *D(KEYF    = F250)                              *
               SEGG    *D(KEYG    = G251)
```

ICMD Call: Use a CALL FUNCTION statement to contain the ICMD function. Use a CALL DATA statement to contain the command.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        ICMD
L  Z0132 DATA  /DIS ACTIVE
```

INIT Call: Use a CALL FUNCTION statement to contain the INIT call and a CALL DATA statement to contain the INIT function DBQUERY, STATUS GROUPA, or STATUS GROUPB.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        INIT                                                        10103210
L  Z0011 DATA  DBQUERY
```

INQY Call: Use a CALL FUNCTION statement to contain the INQY call and either the DBQUERY or ENVIRON subfunction. The subfunctions are in the call input rather than the data input as in the INIT call.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        INQY  ENVIRON                                                  10103210
L  V0256 DATA                                                           10103211
L                                                                       10103212


|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        INQY  DBQUERY                                                  10103210
L  V0088 DATA                                                           10103211
L                                                                       10103212
```

ISRT Call: Use two CALL FUNCTION statements to contain the multiple SSAs and a CALL DATA statement to contain the segment data.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        ISRT  STOCKSEG(NUMFIELD =20011)                              X10103210
              ITEMSSEG                                                  10103211
L  V0018 DATA  3002222222222222                                         10103212
```

ISRT Containing Only One Fixed-Length Segment: Use a CALL FUNCTION statement to contain the ISRT function and segment name, and two CALL DATA statements to contain the fixed-length segment. When inserting only one fixed-length segment, leave columns 4 thru 8 blank and put data in columns 16 thru 71. To continue data, put a nonblank character in column 72, and the continued data in columns 16 thru 71 of the next statement.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        ISRT  JOKESSEG                                                 10103210
L        DATA  THEQUICKBLACKDOGJUMPEDONTOTHECRAZYFOXOOPSTHEQUICKBROWNFO*10103211
              XJUMPEDOVERTHELAZYDOGSIR                                  10103212
```

ISRT Containing Only One Variable-Length Segment: Use a CALL FUNCTION statement to contain the ISRT function and segment name, and two CALL DATA statements to contain the variable-length segment. When only one segment of variable-length is being processed, you must enter a V in column 4, and columns 5 thru 8 must contain the length of the segment data. The length in columns 5 thru 8 is converted to binary and becomes the first two bytes of the segment data. To continue data, put a nonblank character in column 72, and the continued data in columns 16 thru 71 of the next statement.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        ISRT  JOKESSEG                                                 10103210
L  V0080 DATA  THEQUICKBLACKDOGJUMPEDONTOTHECRAZYFOXOOPSTHEQUICKBROWNFO*10103211
              XJUMPEDOVERTHELAZYDOGSIR                                  10103212
```

ISRT Containing Multiple Variable-Length Segments: Use a CALL FUNCTION statement to contain the ISRT function and segment name, and four CALL DATA statements to contain the variable-length segments. For the first segment, you must enter a V in column 4 and the length of the segment data in columns 5 thru 8. If the segment is longer than 56 bytes, put a nonblank character in column 72, and continue data on the next statement as described above. The last statement to contain data for this segment must have a nonblank character in column 72.

The next DATA statement applies to the next variable-length segment and it must contain an M in column 4, the length of the new segment in columns 5 thru 8, and data starting in column 16. Any number of variable-length segments can be concatenated in this manner. If column 72 is blank, the next statement must have the following:

- An L in column 1

- An M in column 4
- The length of the new segment in columns 5 thru 8
- The keyword DATA in columns 10 thru 13
- Data starting in column 16

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        ISRT  AAAAASEG                                                 10103210
L  V0080 DATA  THEQUICKBLACKDOGJUMPEDONTOTHECRAZYFOXOOPSTHEQUICKBROWNFO*10103211
               XJUMPEDOVERTHELAZYDOGSIR                                *10103212
   M0107 DATA  NOWISTHETIMEFORALLGOODMENTOCOMETOTHEAIDOFTHEIRCOUNTRYNOW*10103213
               ISTHETIMEFORALLGOODMENTOCOMETOTHEAIDOFTHEIRCOUNTRY       10103214
```

ISRT Containing Multiple Segments in a PATH CALL: Use a CALL FUNCTION statement to contain the ISRT function and segment name, and seven CALL DATA statements to contain the multiple segments in the PATH CALL.

When DFSDDLT0 is inserting or replacing segments through path calls, you can use V and P in successive statements. The same rules apply for coding multiple variable-length segments, but fixed-length segments must have a P in column 4 of the DATA statement. This causes the length field in columns 5 thru 8 to be used as the length of the segment, and causes the data to be concatenated in the I/O area without including the LL field.

Rules for continuing data in the same segment or starting a new segment in the next statement are the same as those applied to the variable-length segment.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        ISRT  LEV01SEG*D                                             *10103210
               LEV02SEG                                               *10103211
               LEV03SEG                                               *10103212
               LEV04SEG                                                10103213
L  V0080 DATA  THEQUICKBLACKDOGJUMPEDONTOTHECRAZYFOXOOPSTHEQUICKBROWNFO*10103214
               XJUMPEDOVERTHELAZYDOGSIR                                *10103215
   M0107 DATA  NOWISTHETIMEFORALLGOODMENTOCOMETOTHEAIDOFTHEIRCOUNTRYNOW*10103216
               ISTHETIMEFORALLGOODMENTOCOMETOTHEAIDOFTHEIRCOUNTRY      *10103217
L  P0039 DATA  THEQUICKBROWNFOXJUMPEDOVERTHELAZYDOGSIR                 *10103218
L  M0107 DATA  NOWISTHETIMEFORALLGOODMENTOCOMETOTHEAIDOFTHEIRCOUNTRYNOW*10103219
               ISTHETIMEFORALLGOODMENTOCOMETOTHEAIDOFTHEIRCOUNTRY       10103220
```

LOG Call Using an LLZZ Format: Use a CALL FUNCTION statement to contain the LOG function and a CALL DATA statement to contain the LLZZ format of data to be logged.

When you put a Z in column 4, the first word of the record is not coded in the DATA statement. The length specified in columns 5 through 8 must include the 4 bytes for the LLZZ field that is not in the DATA statement.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        LOG                                                           10103210
L  Z0016 DATA  ASEGMENT ONE                                            10103211
```

The A in column 16 becomes the log record ID.

POS Call: Use a CALL FUNCTION statement to contain the POS function and SSA; CALL DATA statement is optional.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        POS   SEGA    (KEYA     =A300)
```

PURG Call with MODNAME and Data: Use a CALL FUNCTION statement to contain the PURG function and MOD name. Use the CALL DATA statement to contain the message data. If MOD name is provided, a DATA statement is required.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        PURG  MODNAME1
L        DATA  FIRST SEGMENT OF NEW MESSAGE
```

PURG Call with Data and no MODNAME: Use a CALL FUNCTION statement to contain the PURG function; a DATA statement is optional.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        PURG
L        DATA  FIRST SEGMENT OF NEW MESSAGE
```

PURG Call without MODNAME or Data: Use a CALL FUNCTION statement to contain the PURG function; CALL DATA statement is optional.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        PURG
```

RCMD Call: Use a CALL FUNCTION statement to contain the RCMD function. Use a CALL DATA statement to retrieve second and subsequent command response segments resulting from an ICMD call.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        RCMD
L  Z0132 DATA
```

REPL Call: Use a CALL FUNCTION statement to contain the REPL function. Use a CALL DATA statement to contain the replacement data.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        REPL
L  V0028 DATA  THIS IS THE REPLACEMENT DATA
```

ROLB Call Requesting Return of First Segment of Current Message: Use a CALL FUNCTION statement to contain the ROLB function. Use the CALL DATA statement to request first segment of current message.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        ROLB
L        DATA  THIS WILL BE OVERLAID WITH FIRST SEGMENT OF MESSAGE
```

ROLB Call Not Requesting Return of First Segment of Current Message: Use a CALL FUNCTION statement to contain the ROLB function. The CALL DATA statement is optional.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        ROLB
```

ROLL Call: Use a CALL FUNCTION statement to contain the ROLL function. The CALL DATA statement is optional.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        ROLL
```

ROLS Call with a Token: Use a CALL FUNCTION statement to contain the ROLS function and token, and the CALL DATA statement to provide the data area that will be overlaid by the data from the SETS call.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        ROLS TOKEN1

L  Z0046 DATA THIS WILL BE OVERLAID WITH DATA FROM SETS
```

ROLS Call without a Token: Use a CALL FUNCTION statement to contain the ROLS function. The CALL DATA statement is optional.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        ROLS
```

ROLX Call: Use a CALL FUNCTION statement to contain the ROLX function. The CALL DATA statement is optional. The ROLX function is treated as a ROLS call with no token.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        ROLX
```

SETO Call: Use a CALL FUNCTION statement to contain the SETO function. The DATA statement is optional; however, if an OPTION statement is passed on the call, the DATA statement is required. Also, if a FEEDBACK statement is passed on the call, then both the DATA and OPTION statements are required. The following is an example of a SETO statement using the OPTION statement and SETO token of SET1.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        SETO  SET1 5000
L        OPT   PRTO=11OPTION1,OPTION2,
L        CONT  OPTION3,
L        CONT  OPTION4
```

11 is the hex value of the length of 11OPTION,.....OPTION4.

The following is an example of a SETO statement using the OPTION statement and SETO token of SET1.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        SETO  SET1 7000
L        OPT   PRTO=11OPTION1,OPTION2,OPTION3,OPTION4
```

11 is the hex value of the length of 11OPTION,.....OPTION4.

The following is an example of a SETO statement using the OPTION statement and SETO token of SET2 and FDBK statement.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        SETO  SET2 5500
L        OPT   PRTO=11OPTION1,OPTION2,OPTION3,OPTION4
L Z0099  FDBK OPTION ERROR FEEDBACK AREA
```

11 is the hex value of the length of 11OPTION,.....OPTION4.

SETS Call with a Token: Use a CALL FUNCTION statement to contain the SETS function and token; use the CALL DATA statement to provide the data that is to be returned to ROLS call.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        SETS  TOKEN1

L  Z0033 DATA  RETURN THIS DATA ON THE ROLS CALL
```

SETS Call without a Token: Use a CALL FUNCTION statement to contain the SETS function; CALL DATA statement is optional.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        SETS
```

This section (SNAP call) contains product-sensitive programming interface information.

SNAP Call: Use a CALL FUNCTION statement to contain the SNAP function and a CALL DATA statement to contain the SNAP data.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        SNAP                                                          10103210
L  V0022 DATA  PRINTDD 22222222                                        10103212
```

This section (STAT call) contains product-sensitive programming interface information.

STAT Call: OSAM statistics require only one STAT call. STAT calls for VSAM statistics retrieve only one subpool at a time, starting with the smallest. See *IMS/ESA Application Programming: Design Guide* for further information about the statistics returned by STAT.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        STAT  DBASF
L        STAT  VBASS
L        STAT  VBASS
L        STAT  VBASS
L        STAT  VBASS
```

SYNC Call: Use a CALL FUNCTION statement to contain the SYNC function. The CALL DATA statement is optional.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        SYNC
```

Initial XRST Call: Use a CALL FUNCTION statement to contain the XRST FUNCTION and a CALL DATA statement that contains a checkpoint ID of blanks to indicate that you are normally starting a program that uses symbolic checkpoints.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        XRST                                                          10101400
L        DATA
L        CKPT
L        DATA  YOURID01
```

CALL Statement

Basic XRST Call: Use a CALL FUNCTION statement to contain the XRST function and a CALL DATA statement to contain the checkpoint ID.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        XRST                                                          10101400
L        DATA  TESTCKPT
```

Symbolic XRST Call: Use a CALL FUNCTION statement to contain the XRST function, a CALL DATA statement to contain the checkpoint ID data, and one or more CALL DATA statements where the data is to be returned.

The XRST call is used with the symbolic CHKP call.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        XRST
L        DATA  TSTCHKP2                                                 X
L      8 DATA  OVERLAY2                                                 X
L     16 DATA  OVERLAY2OVERLAY2
U EIGHT BYTES OF DATA (OVERLAY2) SHOULD BE OVERLAID WITH CHECKPOINTED DATA
U SIXTEEN BYTES OF DATA (OVERLAY2OVERLAY2) IS OVERLAID ALSO
```

CALL FUNCTION Statement with Column-Specific SSAs

In this format, the SSA has intervening blanks between fields. Columns 24, 34, and 37 must contain blanks. Command codes are not permitted. Table 70 gives the format for the CALL FUNCTION statement with column-specific SSAs.

Table 70. CALL FUNCTION Statement (Column-Specific SSAs)

Column	Function	Code and Description	
1	Identifies control statement	**L**	Call statement (see columns 10-13).
2	Reserved	**b**	
3	Reserved	**b**	
4	Reserved	**b**	
5-8	Repeat Count	**b**	If blank, repeat count defaults to 1.
		nnnn	'nnnn' is the number of times to repeat this call. Range 1 to 9999, right-justified but need not contain leading zeros.
10-13	Identifies DL/I call function	**b**	If blank, use function from previous CALL statement.
		xxxx	'xxxx' is a DL/I call function.
		CONT	Continuation indicator for SSAs too long for a single CALL FUNCTION statement. Column 72 of preceding CALL FUNCTION statement must contain a nonblank character. The next CALL statement should have CONT in columns 10 through 13 and the SSA should continue in column 16.
14-15	Reserved	**b**	
16-23	SSA name	**s-name**	Required if call contains SSA.
24	Reserved	**b**	Separator field.
25	Start character for SSA	**(**	Required if segment is qualified.
26-33	SSA field name	**f-name**	Required if segment is qualified.
34	Reserved	**b**	Separator field.
35-36	DL/I call operator(s)	**name**	Required if segment is qualified.

Table 70. CALL FUNCTION Statement (Column-Specific SSAs) (continued)

Column	Function	Code and Description	
37	Reserved	**b**	Separator field.
38-nn	Field value	**nnnnn**	Required if segment is qualified. **Note:** Do not use '5D' or ')' in field value.
nn+1	End character for SSA	**)**	Required if segment is qualified.
72	Continuation column	**b**	No continuations for this statement.
		x	Alone, it indicates multiple SSAs each beginning in column 16 of successive statements. With CONT in columns 10-13 of the next statement, indicates a single SSA that is continued beginning in column 16 of the following statement
73-80	Sequence indication	**nnnnnnnn**	For SYSIN2 statement override.

If a CALL FUNCTION statement contains multiple SSAs, the statement must have a nonblank character in column 72 and the next SSA must start in column 16 of the next statement. If a field value extends past column 71, put a nonblank character in column 72. In the next statement insert the keyword CONT in columns 10 through 13 and continue the field value starting at column 16. Maximum length for field value is 256 bytes, maximum size for an SSA is 290 bytes, and the maximum number of SSAs for this program is 15, which is the same as the IMS limit.

DFSDDLT0 Call Functions

The DFSDDLT0 call functions were created for DFSDDLT0. They do not represent “valid” IMS calls and are not punched as output if DFSDDLT0 encounters them while a CTL (PUNCH) statement is active. Table 71 shows the special call functions of the CALL FUNCTION statement. Descriptions and examples of these special functions follow.

Table 71. CALL FUNCTION Statement with DFSDDLT0 Call Functions

Column	Function	Code	Description
1	Identifies control statement	**L**	Call statement.
2-4	Reserved	**b**	
5-8	Repeat count	**b**	If blank, repeat count defaults to 1.
		nnnn	'nnnn' is the number of times to repeat this call. Range is 1 to 9999, right-justified but need not contain leading zeros.
9	Reserved	**b**	
10-15	Special call function	**STAKb**	Stack control statements for later execution.
		ENDbb	Stop stacking and begin execution.
		SKIPb	Skip statements until START function is encountered.
		START	Start processing statements again.
73-80	Sequence indication	**nnnnnnnn**	For SYSIN2 statement override.

- STAK/END(stacking control statements)

With the STAK statement, you repeat a series of statements that were read from SYSIN and held in memory. All control statements between the STAK statement and the END statement are read and saved. When DFSDDLT0 encounters the END statement, it executes the series of calls as many times as specified in columns 5 through 8 of the STAK statement. STAK calls imbedded within another STAK cause the outer STAK call to be abnormally terminated.

- SKIP/START (skipping control statements)

  With the SKIP and START statements, you identify groups of statements that you do not want DFSDDLT0 to process. These functions are normally read from SYSIN2 and provide a temporary override to an established SYSIN input stream. DFSDDLT0 reads all control statements occurring between the SKIP and START statements, but takes no action. When DFSDDLT0 encounters the START statement, it reads and processes the next statement normally.

Examples of DFSDDLT0 Call Functions

STAK/END Call: The following example shows the STAK and END call functions.

```
//BATCH.SYSIN DD *                                                      10000700
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
O SNAP= ,ABORT=0                                                        10000800
S 1 1 1 1 1                                                             10001000
L        GU    SEGA    (KEYA     =A300)                                 10001100
L   0003 STAK                                                           10001150
WTO THIS IS PART OF THE STAK                                            10001200
T THIS COMMENT IS PART OF THE STAK                                      10001300
L        GN                                                             10001400
L        END                                                            10001450
U THIS COMMENT SHOULD GET PRINTED AFTER THE STAK IS DONE 3 TIMES        10001500
L   0020 GN                                                            10001600
/*
```

SKIP/START Call: The following example demonstrates the use of the SKIP and START call functions in SYSIN2 to override and stop the processing of the STAK and END call functions in SYSIN. DFSDDLT0 executes the GU call function in SYSIN, skips the processing of STACK, WTO, T comment, GN, and END in SYSIN, and goes to the COMMENT.

```
//BATCH.SYSIN DD *                                                      10000700
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
O SNAP= ,ABORT=0                                                        10000800
S 1 1 1 1 1                                                             10001000
L        GU    SEGA    (KEYA     =A300)                                 10001100
L   0003 STAK                                                           10001150
WTO THIS IS PART OF THE STAK                                            10001200
T THIS COMMENT IS PART OF THE STAK                                      10001300
L        GN                                                             10001400
L        END                                                            10001450
U THIS COMMENT SHOULD GET PRINTED AFTER THE STAK IS DONE 3 TIMES        10001500
L   0020 GN                                                            10001600
/*
//BATCH.SYSIN2 DD *
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        SKIP                                                           10001150
L        START                                                          10001450
U THIS COMMENT SHOULD REPLACE THE STAK COMMENT                          10001500
U ********THIS COMMENT SHOULD GET PRINTED BECAUSE OF SYSIN2*********    10001650
/*
```

COMMENT Statement

Use the COMMENT statement to print comments in the output data. The two types of COMMENT statements, conditional and unconditional, are described below. Table 72 shows the format of the COMMENT statement.

Conditional COMMENT Statement

You can use up to five conditional COMMENT statements per call; no continuation mark is required in column 72. Code the statements in the DFSDDLT0 stream before the call they are to document. Conditional COMMENTS are read and held until a CALL is read and executed. (If a COMPARE statement follows the CALL, conditional COMMENTS are held until after the comparison is completed.) You control whether the conditional comments are printed with column 3 of the STATUS statement. DFSDDLT0 prints the statements according to the STATUS statement in the following order: conditional COMMENTS, the CALL, and the COMPARE(s). The time and date are also printed with each conditional COMMENT statement.

Unconditional COMMENT Statement

You can use any number of unconditional COMMENT statements. Code them in the DFSDDLT0 stream before the call they are to document. The time and date are printed with each unconditional COMMENT statement.

Table 72. COMMENT Statement

Column	Function	Code	Description
1	Identifies control statement	**T**	Conditional comment statement.
		U	Unconditional comment statement.
2-72	Comment data		Any relevant comment.
73-80	Sequence indication	**nnnnnnnn**	For SYSIN2 statement override.

Example of COMMENT Statement

T/U Comment Calls: The following example shows the T and U comment calls.

```
//BATCH.SYSIN DD *                                                    10000700
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
O SNAP= ,ABORT=0                                                     10000800
S 1 1 1 1 1                                                          10001000
L        GU    SEGB    (KEYA     =A400)                             10001100
T THIS COMMENT IS A CONDITIONAL COMMENT FOR THE FIRST GN              10001300
L        GN                                                          10001400
U THIS COMMENT IS AN UNCONDITIONAL COMMENT FOR THE SECOND GN          10001500
L   0020 GN                                                          10001600
/*
```

COMPARE Statement

The COMPARE statement compares the actual results of a call with the expected results. The three types of COMPARE statements are the COMPARE PCB, COMPARE DATA, and COMPARE AIB.

When you use the COMPARE PCB, COMPARE DATA, and COMPARE AIB statements you must:

- Code COMPARE statements in the DFSDDLT0 stream immediately after either the last continuation, if any, of the CALL DATA statement or another COMPARE statement.
- Specify the print option for the COMPARE statements in column 7 of the STATUS statement.

For all three COMPARE statements:

- The condition code returned for a COMPARE gives the total number of unequal comparisons.
- For single fixed-length segments, DFSDDLT0 uses the comparison length to perform comparisons if you provide a length. The length comparison option (column 3) is not applicable.

Product-sensitive programming interface

When you use the COMPARE PCB statement and you want a snap dump when there is an unequal comparison, request it on the COMPARE PCB statement. A snap dump to a log with SNAP ID COMPxxxx is issued along with the snap dump options specified in column 3 of the COMPARE PCB statement.

The numeric part of the SNAP ID is initially set to 0000 and is incremented by 1 for each SNAP resulting from an unequal comparison.

End of Product-sensitive programming interface

COMPARE DATA Statement

The COMPARE DATA statement is optional. It compares the segment returned by IMS to the data in the statement to verify that the correct segment was retrieved. Table 73 gives the format of the COMPARE DATA statement.

Table 73. COMPARE DATA Statement

Column	Function	Code	Description
1	Identifies control statement	**E**	COMPARE statement.
2	Reserved	**b**	
3	Length comparison option	**b**	For fixed-length segments or if the LL field of the segment is not included in the comparison; only the data is compared.
		L	The length in columns 5-8 is converted to binary and compared against the LL field of the segment.
4	Segment length option	**b**	
		V	For a variable-length segment only, or for the first variable-length segment of multiple variable-length segments in a path call, or for a concatenated logical child/logical parent segment.
		M	For the second or subsequent variable-length segment of a path call, or for a concatenated logical child/logical parent segment.

Table 73. COMPARE DATA Statement (continued)

Column	Function	Code	Description
		P	For fixed-length segments in path calls.
		Z	For message segment.
5-8	Comparison length	**nnnn**	Length to be used for comparison. (Required for length options V, M, and P if L is coded in column 3.)
9	Reserved	**b**	
10-13	Identifies type of statement	**DATA**	Required for the first I/O COMPARE statement and the first statement of a new segment if data from previous I/O COMPARE statement is not continued.
14-15	Reserved	**b**	
16-71	String of data		Data against which the segment in the I/O area is to be compared.
72	Continuation column	**b**	If blank, data is NOT continued.
		x	If not blank, data will be continued, starting in columns 16-71 of the subsequent statements for a maximum of 3840 bytes.
73-80	Sequence indication	**nnnnnnnn**	For SYSIN2 statement override.

Notes:

- If you code an L in column 3, the value in columns 5 through 8 is converted to binary and compared against the LL field of the returned segment. If you leave column 3 blank and the segment is not in a path call, then the value in columns 5 through 8 is used as the length of the comparison.
- If you code column 4 with a V, P, or M, you must enter a value in columns 5 through 8.
- If this is a path call comparison, code a P in column 4. The value in columns 5 through 8 must be the exact length of the fixed segment used in the path call.
- If you specify the length of the segment, this length is used in the COMPARE and in the display. If you do not specify a length, DFSDDLT0 uses the shorter of the following for the length of the comparison and display:
  - The length of data supplied in the I/O area by IMS
  - The number of DATA statements read times 56

COMPARE AIB Statement

The COMPARE AIB statement is optional. You can use it to compare values returned to the AIB by IMS. Table 74 shows the format of the COMPARE AIB statement.

Table 74. COMPARE AIB Statement

Column	Function	Code	Description
1	Identifies control statement	**E**	COMPARE statement.
2	Hold compare option	**H**	Hold COMPARE statement; see the paragraph below for details.
		b	Reset hold condition for a single COMPARE statement.
3	Reserved	**b**	
4-6	AIB compare	**AIB**	Identifies an AIB compare.

Table 74. COMPARE AIB Statement (continued)

Column	Function	Code	Description
7	Reserved	**b**	
8-11	Return code	**xxxx**	Allow specified return code only.
12	Reserved		
13-16	Reason code	**xxxx**	Allow specified reason code only.
17-72	Reserved	**b**	b
73-80	Sequence indication	**nnnnnnnn**	For SYSIN2 statement override.

To execute the same COMPARE AIB after a series of calls, put an H in column 2. When you specify an H, the COMPARE statement executes after each call. The H COMPARE statement is particularly useful when comparing with the same status code on repeated calls. The H COMPARE statement stays in effect until another COMPARE AIB statement is read.

COMPARE PCB Statement

The COMPARE PCB statement is optional. You can use it to compare values returned to the PCB by IMS or to print blocks or buffer pool. Table 75 shows the format of the COMPARE PCB statement.

Table 75. COMPARE PCB Statement

Column	Function	Code	Description
1	Identifies control statement	**E**	COMPARE statement.
2	Hold compare option	**H**	Hold compare statement.
		b	Reset hold condition for a single COMPARE statement.
3	Snap dump options (if compare was unequal)	**b**	Use default value. (You can change the default value or turn off the option by coding the value in an OPTION statement.)
		1	The complete I/O buffer pool.
		2	The entire region (batch regions only).
		4	The DL/I blocks.
		8	Terminate the job step on miscompare of DATA or PCB.
		S	To SNAP subpools 0 through 127. Requests for multiple SNAP dump options can be obtained by summing their respective hexadecimal values. If anything other than a blank, 1-9, A-F, or S is coded in column 3, the SNAP dump option is ignored.
4	Extended SNAP[1] options	**b**	Ignore extended option.
		P	SNAP the complete buffer pool (batch).

Table 75. COMPARE PCB Statement (continued)

Column	Function	Code	Description
		S	SNAP subpools 0 through 127 (batch). An area is never snapped twice. The SNAP option is a combination of columns 3 (SNAP dump option) and 4 (extended SNAP option).
5-6	Segment level	**nn**	'nn' is the segment level for COMPARE PCB. A leading zero is required.
7	Reserved	**b**	
8-9	Status code	**b**	Allow blank status code only.
		xx	Allow specified status code only.
		XX	Do not check status code.
		OK	Allow the following: blank, GA, GC, or GK.
10	Reserved	**b**	
11-18	Segment name User Identification	**xxxxxxxx**	Segment name for DB PCB compare.
			Logical terminal for I/O.
			Destination for ALT PCB.
19	Reserved	**b**	
20-23	Length of key	**nnnn**	'nnnn' is length of the feedback key.
24-71 or	Concatenated key		Concatenated key feedback for DB PCB compare.
24-31	User ID		User identification for TP PCB.
72	Continuation column	**b**	If blank, key feedback is *not* continued.
		x	If not blank, key feedback is continued, starting in columns 16-71 of subsequent statements.
73-80	Sequence indication	**nnnnnnnn**	For SYSIN2 statement override.
Note: 1. SNAP is a Product-sensitive programming interface.			

Blank fields are not compared to the corresponding field in the PCB, except for the status code field. (Blanks represent a valid status code.) To accept the status codes blank, GA, GC, or GK as a group, put OK in columns 8 and 9. To stop comparisons of status codes, put XX in columns 8 and 9.

To execute the same compare after a series of calls, put an H in column 2. This executes the COMPARE statement after each call. This is particularly useful to compare to a blank status code only when loading a database. The H COMPARE statement stays in effect until another COMPARE PCB statement is read.

Examples of COMPARE DATA and PCB Statements

COMPARE PCB Statement for Blank Status Code: The COMPARE PCB statement is coded blank. It checks a blank status code for the GU.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        GU                                                              10101100
E                                                                        10101200
```

COMPARE PCB Statement for SSA Level, Status Code, Segment Name, Concatenated Key Length, and Concatenated Key: The COMPARE PCB statement is a request to compare the SSA level, a status code of OK (which includes blank, GA, GC, and GK), segment name of SEGA, concatenated key length of 0004, and a concatenated key of A100.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        GU
E   01 OK SEGA     0004A100
```

COMPARE PCB Statement for SSA Level, Status Code, Segment Name, Concatenated Key Length, and Concatenated Key: The COMPARE PCB statement causes the job step to terminate based on the 8 in column 3 when any of the fields in the COMPARE PCB statement are not equal to the corresponding field in the PCB.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        GU                                                              10105100
E 8 01 OK SEGK     0004A100                                              10105200
```

COMPARE PCB Statement for Status Code with Hold Compare: The COMPARE PCB statement is a request to compare the status code of OK (which includes blank, GA, GC, and GK) and hold that compare until the next COMPARE PCB statement. The compare of OK is used on GN following GU and is also used on a GN that has a request to be repeated six times.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        GU    SEGA    (KEYA    = A300)                                  20201100
L        GN                                                              20201300
EH     OK                                                                20201400
L   0006 GN                                                              20201500
```

COMPARE DATA Statement for Fixed-Length Segment: The COMPARE DATA statement is a request to compare the data returned. 72 bytes of data are compared.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        GU
E        DATA  A100A100A100A100A100A100A100A100A100A100A100A100A100X10102200
E              A100A100A100A100                                          10102300
```

COMPARE DATA Statement for Fixed-Length Data for 64 Bytes: The COMPARE DATA statement is a request to compare 64 bytes of the data against the data returned.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        GU                                                              10101600
E   0064 DATA  A100A100A100A100A100A100A100A100A100A100A100A100A100X10101700
E              A100A100B111B111                                          10101800
```

COMPARE DATA Statement for Fixed-Length Data for 72 Bytes: The COMPARE DATA statement is a request to compare 72 bytes of the data against the data returned.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        GU                                                                10103900
E LP0072 DATA  A100A100A100A100A100A100A100A100A100A100A100A100A100A100X10104000
E              A100A100A100A100                                            10104100
```

COMPARE DATA Statement for Variable-Length Data of Multiple-Segments Data and Length Fields: The COMPARE DATA statement is a request to compare 36 bytes of the data against the data returned for segment 1 and 16 bytes of data for segment 2. It compares the length fields of both segments.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        ISRT  D        (DSS      =  DSS01)                              X38005500
L              DJ       (DJSS     =  DJSS01)                             X38005600
L              QAJAXQAJ                                                   38005700
L  V0036 DATA  QSS02QASS02QAJSS01QAJASS97*IQAJA**                        *38005800
L  M0016 DATA  QAJSS01*IQAJ**                                             38005850
L        GHU   D        (DSS      =  DSS01)                              X38006000
               DJ       (DJSS     =  DJSS01)                             X38006100
               QAJAXQAJ (QAJASS   =  QAJASS97)                           38006200
E LV0036 DATA  QSS02QASS02QAJSS01QAJASS97*IQAJA**                        *38006300
E LM0016 DATA  QAJSS01*2QAJ**                                             38006350
```

COMPARE DATA Statement for Variable-Length Data of Multiple Segments with no Length Field COMPARE: The COMPARE DATA statement is a request to compare 36 bytes of the data against the data returned for segment 1 and 16 bytes of data for segment 2 with no length field compares of either segment.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        ISRT  D        (DSS      =  DSS01)                              X38005500
L              DJ       (DJSS     =  DJSS01)                             X38005600
L              QAJAXQAJ                                                   38005700
L  V0036 DATA  QSS02QASS02QAJSS01QAJASS97*IQAJA**                        *38005800
L  M0016 DATA  QAJSS01*IQAJ**                                             38005850
L        GHU   D        (DSS      =  DSS01)                              X38006000
               DJ       (DJSS     =  DJSS01)                             X38006100
               QAJAXQAJ (QAJASS   =  QAJASS97)                           38006200
E  V0036 DATA  QSS02QASS02QAJSS01QAJASS97*IQAJA**                        *38006300
   M0016 DATA  QAJSS01*2QAJ**                                             38006350
```

COMPARE DATA Statement for Variable-Length Data of Multiple Segments and One Length Field COMPARE: The COMPARE DATA statement is a request to compare 36 bytes of the data against the data returned for segment 1 and 16 bytes of data for segment 2. It compares the length field of segment 1 only.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
L        ISRT  D        (DSS      =  DSS01)                              X38005500
L              DJ       (DJSS     =  DJSS01)                             X38005600
L              QAJAXQAJ                                                   38005700
L  V0036 DATA  QSS02QASS02QAJSS01QAJASS97*IQAJA**                        *38005800
L  M0016 DATA  QAJSS01*IQAJ**                                             38005850
L        GHU   D        (DSS      =  DSS01)                              X38006000
               DJ       (DJSS     =  DJSS01)                             X38006100
               QAJAXQAJ (QAJASS   =  QAJASS97)                           38006200
E LV0036 DATA  QSS02QASS02QAJSS01QAJASS97*IQAJA**                        *38006300
   M0016 DATA  QAJSS01*2QAJ**                                             38006350
```

IGNORE Statement

DFSDDLT0 ignores any statement with an N or a period (.) in column 1. You can use the N or . (period) to comment out a statement in either the SYSIN or SYSIN2 input streams. Using an N or . (period) in a SYSIN2 input stream causes the SYSIN input stream to be ignored as well. See “SYSIN2 DD Statement” on page 565 for information on how to override SYSIN input. Table 76 gives the format of the IGNORE statement. An example of the statement follows.

Table 76. IGNORE Statement

Column	Function	Code	Description
1	Identifies control statement	N or .	IGNORE statement.
2-72	Ignored		
73-80	Sequence indication	nnnnnnnn	For SYSIN2 statement override.

Example of IGNORE (N or .)

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
. NOTHING IN THIS AREA WILL BE PROCESSED.  ONLY THE SEQUENCE NUMBER     67101010
N WILL BE USED IF READ FROM SYSIN2 OR SYSIN.                            67101020
```

OPTION Statement

Use the OPTION statement to override various default options. Use multiple OPTION statements if you cannot fit all the options you want in one statement. No continuation character is necessary. Once you set an option, it remains in effect until you specify another OPTION statement to change the first parameter. Table 77 shows the format of the OPTION statement. An example follows.

Table 77. OPTION Statement

Column	Function	Code	Description
1	Identifies control statement	O	OPTION statement (free-form parameter fields).
2	Reserved	b	b
3-72	Keyword parameters:		
	ABORT=	• 0 • 1 to 9999	• Turns the ABORT parameter off. • Number of unequal compares before aborting job. Initial default is 5.
	LINECNT=	10 to 99	Number of lines printed per page. Must be filled with zeros. Initial default 54.
	SNAP[1]	x	SNAP option default, when results of compare are unequal. To turn the SNAP option off, code 'SNAP='. See “COMPARE PCB Statement” on page 552 for the appropriate values for this parameter. (Initial default is 5 if this option is not coded. This causes the I/O buffer pool and the DL/I blocks to be dumped with a SNAP call.)
	DUMP/NODUMP		Produce/do not produce dump if job abends. Default is NODUMP.

Table 77. OPTION Statement (continued)

Column	Function	Code	Description
	LCASE=	• H • C	• Hexadecimal representation for lower case characters. This is the initial default. • Character representation for lower case characters.
	STATCD/NOSTATCD		Issue/do not issue an error message for the internal, end-of-job stat call that does not receive a blank or GA status code. NOSTATCD is the default.
	ABU249/NOABU249		Issue/do not issue a DFSDDLT0 ABENDU0249 when an invalid status code is returned for any of the internal end-of-job stat calls in a batch environment. NOABU249 is the default.
73 - 80	Sequence indication	nnnnnnnn	For SYSIN2 statement override.
Note:			
1. SNAP is a Product-sensitive programming interface.			

OPTION statement parameters can be separated by commas.

Example of OPTION Control Statement

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
O ABORT=5,DUMP,LINECNT=54,SPA=4096,SNAP=5                              67101010
```

PUNCH Statement

The PUNCH CTL statement allows you to produce an output data set consisting of COMPARE PCB statements, COMPARE DATA statements, COMPARE AIB statements, other control statements (with the exceptions noted below), or combinations of the above. Table 78 shows the format and keyword parameters for the PUNCH CTL statement.

Table 78. PUNCH CTL Statement

Column	Function	Description	Description
1-3	Identifies control statement	**CTL**	PUNCH statement.
4-9	Reserved	**b**	
10-13	Punch control	**PUNC**	Begin punching (no default values).
		NPUN	Stop punching (default value).
14-15	Reserved	**b**	
16-72	Keyword parameters:		

Table 78. PUNCH CTL Statement (continued)

Column	Function	Description	Description
	OTHER		Reproduces all input control statements except: • CTL (PUNCH) statements. • N or . (IGNORE) statements. • COMPARE statements. • CALL statements with functions of SKIP and START. Any control statements that appear between SKIP and START CALLs are not punched. (See SKIP/START 548). • CALL statements with functions of STAK and END. Control statements that appear between STAK and END CALLS are saved and then punched the number of times indicated in the STAK CALL. (See STAK/END 547).
	DATAL		Create a full data COMPARE using all of the data returned to the I/O area. Multiple COMPARE statements and continuations are produced as needed.
	DATAS		Create a single data COMPARE statement using only the first 56 bytes of data returned to the I/O area.
	PCBL		Create a full PCB COMPARE using the complete key feedback area returned in the PCB. Multiple COMPARE statements and continuations are produced as needed.
	PCBS		Create a single PCB COMPARE statement using only the first 48 bytes of the key feedback area returned in the PCB.
	SYNC/NOSYNC		If a GB status code is returned on a Fast Path call while in STAK, but prior to exiting STAK, this function issues or does not issue SYNC.
	START=		00000001 to 99999999. This is the starting sequence number to be used for the punched statements. Eight numeric bytes must be coded.

Table 78. PUNCH CTL Statement (continued)

Column	Function	Description	Description
	INCR=		1 to 9999. Increment the sequence number of each punched statement by this value. Leading zeros are not required.
	AIB		Create an AIB COMPARE statement.
73-80	Sequence indication	**nnnnnnnn**	For SYSIN2 statement override.

To change the punch control options while processing a single DFSDDLT0 input stream, always use PUNCH CTL statements in pairs of PUNC and NPUN.

One way to use the PUNCH CTL statement is as follows:

1. Code only the CALL statements for a new test. Do not code the COMPARE statements.
2. Verify that each call was executed correctly.
3. Make another run using the PUNCH CTL statement to have DFSDDLT0merge the proper COMPARE statements and produce a new output data set that can be used as input for subsequent regression tests.

You can also use PUNCH CTL if segments in an existing database are changed. The control statement causes DFSDDLT0 to produce a new test data set that has the correct COMPARE statements rather than you having to manually change the COMPARE statements.

Parameters in the CTL statement must be the same length as described in Table 78, and they must be separated by commas.

Example of PUNCH CTL Statement

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
CTL      PUNC  PCBS,DATAS,OTHER,START=00000010,INCR=0010               33212010
CTL      NPUN                                                          33212020
```

The DD statement for the output data set is labeled PUNCHDD. The data sets are fixed block with LRECL=80. Block size as specified on the DD statement is used. If not specified, the block size is set to 80. If the program is unable to open PUNCHDD, DFSDDLT0 issues abend 251.

Example of PUNCH CTL Statement for All Parameters

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
CTL      PUNC  OTHER,DATAL,PCBL,START=00000001,INCR=1000,AIB           33212010
```

STATUS Statement

With the STATUS statement, you establish print options and name the PCB that you want subsequent calls to be issued against. Table 79 on page 560 shows the format of the STATUS statement.

STATUS Statement

Table 79. STATUS Statement

Column	Function	Code	Description
1	Identifies control statement	**S**	STATUS statement.
2	Output device option	**b**	Use PRINTDD when in a DL/I region; use I/O PCB in MPP region.
		1	Use PRINTDD in MPP region if DD statement is provided; otherwise, use I/O PCB.
		A	Same as if 1, and disregard all other fields in this STATUS statement.
3	Print comment option	**b**	Do not print.
		1	Print for each call.
		2	Print only if compare done and unequal.
4	Print AIB option	**b**	Do not print.
		1	Print for each call.
		2	Print only if compare done and unequal.
5	Print call option	**b**	Do not print.
		1	Print for each call.
		2	Print only if compare done and unequal.
6	Reserved	**b**	
7	Print compare option	**b**	Do not print.
		1	Print for each call.
		2	Print only if compare done and unequal.
8	Reserved	**b**	
9	Print PCB option	**b**	Do not print.
		1	Print for each call.
		2	Print only if compare done and unequal.
10	Reserved	**b**	
11	Print segment option	**b**	Do not print.
		1	Print for each call.
		2	Print only if compare done and unequal.
12	Set task and real time	**b**	Do not time
		1	Time each call.
		2	Time each call if compare done and unequal.
13-14	Reserved	**b**	

Table 79. STATUS Statement (continued)

Column	Function	Code	Description
15	PCB selection option	**1**	PCB name passed in columns 16-23 (use option 1).
		2	DBD name passed in columns 16-23 (use option 2).
		3	Relative DB PCB passed in columns 16-23 (use option 3).
		4	Relative PCB passed in columns 16-23 (use option 4).
		5	$LISTALL passed in columns 16-23 (use option 5).
		b	If column 15 is blank, DFSDDLT0 selects options 2 through 5 based on content of columns 16-23.
Opt. 1 16-23	PCB selection PCB name	**alpha**	These columns must contain the name of the label on the PCB at PSBGEN, or the name specified on the PCBNAME= operand for the PCB at PSBGEN time.
Opt. 2 16-23	PCB selection DBD name	**b**	
		alpha	The default PCB is the first database PCB in the PSB. If columns 16-23 are blank, current PCB is used. If DBD name is specified, this must be the name of a database DBD in the PSB.
Opt. 3 16-18 19-23	PCB selection Relative position of PCB in PSB	**b**	
		numeric	When columns 16 through 18 are blank, columns (19-23) of this field are interpreted as the relative number of the DB PCB in the PSB. This number must be right-justified to column 23, but need not contain leading zeros.
Opt. 4 16-18 19-23	PCB selection I/O PCB Relative position of PCB in PSB	**TPb**	
		numeric	When columns 16 through 18 = 'TPb', columns (19-23) of this field are interpreted as the relative number of the PCB from the start of the PCB list. This number must be right-justified to column 23, but need not contain leading zeros. I/O PCB is always the first PCB in the PCB list in this program.
Opt. 5 16-23	List all PCBs in the PSB	**$LISTALL**	Prints out all PCBs in the PSB for test script.
24	Print status option	**b**	Use print options to print this STATUS statement.
		1	Do not use print options in this statement; print this STATUS statement.
		2	Do not print this STATUS statement but use print options in this statement.
		3	Do not print this STATUS statement and do not use print options in this statement.
25-28	PCB processing option	**xxxx**	This is optional and is only used when two PCBs have the same name but different processing options. If not blank, it is used in addition to the PCB name in columns 16 through 23 to select which PCB in the PSB to use.
29	Reserved	**b**	

STATUS Statement

Table 79. STATUS Statement (continued)

Column	Function	Code	Description
30-32	AIB interface	**AIB**	Indicates that the AIB interface is used and the AIB is passed rather than passing the PCB. (Passing the PCB is the default.) **Note:** When the AIB interface is used, the PCB must be defined at PSBGEN with PCBNAME=name. IOPCB is the PCB name used for all I/O PCBs. DFSDDLT0 recognizes that name when column 15 contains a 1 and columns 16 through 23 contain IOPCB.
33	Reserved		
37-72	Reserved		
73-80	Sequence indication	**nnnnnnnn**	For SYSIN2 statement override.

If DFSDDLT0 does not encounter a STATUS statement, all default print options (columns 3 through 12) are 2 and the default output device option (column 2) is 1. You can code a STATUS statement before any call sequence in the input stream, changing either the PCB to be referenced or the print options.

The referenced PCB stays in effect until a subsequent STATUS statement selects another PCB. However, a call that must be issued against an I/O PCB (such as LOG) uses the I/O PCB for that call. After the call, the PCB changes back to the original PCB.

Examples of STATUS Statement

To Print Each CALL Statement: The following STATUS statement tells DFSDDLT0 to print these options: COMMENTS, CALL, COMPARE, PCB, and SEGMENT DATA for all calls.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
S 1 1 1 1 1
```

To Print Each CALL Statement, Select a PCB: The following STATUS statements tell DFSDDLT0 to print the COMMENTS, CALL, COMPARE, PCB, and SEGMENT DATA options for all calls, and select a PCB.

The 1 in column 15 is required for PCBNAME. If omitted, the PCBNAME is treated as a DBDNAME.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
S 1 1 1 1 1   1PCBNAME


|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
S 1 1 1 1 1   1PCBNAME       AIBb
```

To print each CALL statement, select PCB based on a DBD name: The following STATUS statements tell DFSDDLT0 to print the COMMENTS, CALL, COMPARE, PCB, and SEGMENT DATA options for all calls, and select a PCB by a DBD name.

The 2 in column 15 is optional.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
S 1 1 1 1 1   2DBDNAME

|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
S 1 1 1 1 1   2DBDNAME       AIBb
```

If you do not use the AIB interface, you do not need to change STATUS statement input to existing streams; existing call functions will work just as they have in the past. However, if you want to use the AIB interface, you must change the STATUS statement input to existing streams to include AIB in columns 30 through 32. The existing DBD name, Relative DB PCB, and Relative PCB will work if columns 30 through 32 contain AIB and the PCB has been defined at PSBGEN with PCBNAME=name.

WTO Statement

The WTO (Write to Operator) statement sends a message to the MVS console without waiting for a reply. Table 80 shows the format for the WTO statement.

Table 80. WTO Statement

Column	Function	Code	Description
1-3	Identifies control statement	**WTO**	WTO statement.
4	Reserved	**b**	
5-72	Message to send		Message to be written to the system console.
73-80	Sequence indication	**nnnnnnnn**	For SYSIN2 statement override.

Example of WTO Statement

This WTO statement sends a message to the MVS console and continues the test stream.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
WTO AT A "WTO" WITHIN TEST STREAM --WTO NUMBER 1-- TEST STARTED
```

WTOR Statement

The WTOR (Write to Operator with Reply) statement sends a message to the MVS system console and waits for a reply. Table 81 shows the format of the WTO statement.

Table 81. WTOR Statement

Column	Function	Code	Description
1-4	Identifies control statement	**WTOR**	WTOR statement.
5	Reserved	**b**	
6-72	Message to send		Message to be written to the system console.
73-80	Sequence indication	**nnnnnnnn**	For SYSIN2 statement override.

Example of WTOR Statement

This WTOR statement causes the test stream to hole until DFSDDLT0 receives a response from the MVS console operator. Any response is valid.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
WTOR AT A "WTOR" WITHIN TEST STREAM - ANY RESPONSE WILL CONTINUE
```

JCL Requirements

This section defines the DD statements that DFSDDLT0 uses. Execution JCL depends on the installation data set naming standards as well as the IMS environment (batch or online). See Figure 86.

```
//SAMPLE JOB ACCOUNTING,NAME,MSGLEVEL=(1,1),MSGCLASS=3,PRTY=8           33001100
//GET EXEC PGM=DFSRRC00,PARM='DLI,DFSDDLT0,PSBNAME'                     33001200
//STEPLIB  DD DSN=IMS.RESLIB,DISP=SHR                                   33001300
//IMS      DD DSN=IMS2.PSBLIB,DISP=(SHR,PASS)                           33001400
//         DD DSN=IMS2.DBDLIB,DISP=(SHR,PASS)                           33001500
//DDCARD   DD DSN=DATASET,DISP=(OLD,KEEP)                               33001600
//IEFRDER  DD DUMMY                                                     33001700
//PRINTDD  DD SYSOUT=A                                                  33001800
//SYSUDUMP DD SYSOUT=A                                                  33001900
//SYSIN    DD *                                                         33002000
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
U THIS IS PART OF AN EXAMPLE                                            33002100
S 1 1 1 1 1    PCB-NAME                                                 33002200
L        GU                                                            33002300
/*
//SYSIN2   DD *
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
ABEND                                                                   33002300
/*
```

Figure 86. Example JCL Code for DD Statement Definition

Figure 87 is an example of coding JCL for DFSDDLT0 in a BMP. Use of a procedure is optional and is only shown here as an example.

```
//SAMPLE JOB ACCOUNTING,NAME,MSGLEVEL=(1,1),MSGCLASS=A                  00010047
//*************************************************************
//*  BATCH DL/I JOB TO RUN FOR RSR TESTING                     *
//*************************************************************
//BMP   EXEC IMSBATCH,MBR=DFSDDLT0,PSB=PSBNAME
//BMP.PRINTDD DD SYSOUT=A
//BMP.PUNCHDD DD SYSOUT=B
//BMP.SYSIN DD *
U ***THIS IS PART OF AN EXAMPLE OF SYSIN DATA                           00010000
S 1 1 1 1 1          1                                                  00030000
L        GU                                                            00040000
L   0099 GN                                                            00050000
/*
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
//BMP.SYSIN2 DD *
U ***THIS IS PART OF AN EXAMPLE OF SYSIN2 DATA  ******************     00020000
ABEND                                                                  00050000
/*
```

Figure 87. Example JCL Code for DFSDDLT0 in a BMP

SYSIN DD Statement

The data set specified by the SYSIN DD statement is the normal input data set for DFSDDLT0. When processing input data that is on direct-access or tape, you may want to override certain control statements in the SYSIN input stream or to add other control statements to it. You do this with a SYSIN2 DD statement and the control statement sequence numbers.

Sequence numbers in columns 73 to 80 for SYSIN data are optional unless a SYSIN2 override is used. If a SYSIN2 override is used, follow the directions for using sequence numbers as described in "SYSIN2 DD Statement".

SYSIN2 DD Statement

DFSDDLT0 does not require the SYSIN2 DD statement, but if it is present in the JCL, DFSDDLT0 will read and process the specified data sets. When using SYSIN2, the following items apply:

- The SYSIN DD data set is the primary input. DFSDDLT0 attempts to insert the SYSIN2 control statements into the SYSIN DD data set.
- You must code the control groups and sequence numbers properly in columns 73 to 80 or the merging process will not work.
- Columns 73 and 74 indicate the control group of the statement.
- Columns 75 to 80 indicate the sequence number of the statement.
- Sequence numbers *must* be in numeric order within their control group.
- Control groups in SYSIN2 must match the SYSIN control groups, although SYSIN2 does not have to use all the control groups used in SYSIN. DFSDDLT0 does not require that control groups be in numerical order, but the control groups in SYSIN2 must be in the same order as those in SYSIN.
- When DFSDDLT0 matches a control group in SYSIN and SYSIN2, it processes the statements by sequence number. SYSIN2 statements falling before or after a SYSIN statement are merged accordingly.
- If the sequence number of a SYSIN2 statement matches the sequence number of a SYSIN statement in its control group, the SYSIN2 overrides the SYSIN.
- If the program reaches the end of SYSIN before it reaches the end of SYSIN2, it processes the records of SYSIN2 as if they were an extension of SYSIN.
- Replacement or merging occurs only during the current run. The original SYSIN data is not changed.
- During merge, if one of the control statements contains blanks in columns 73 through 80, DFSDDLT0 discards the statement containing blanks, sends a message to PRINTDD, and continues the merge until end-of-file is reached.

PRINTDD DD Statement

The PRINTDD DD statement defines output data set for DFSDDLT0, including displays of control blocks using the SNAP call. It must conform to the MVS/ESA SNAP data set requirements.

PUNCHDD DD Statement

The DD statement for the output data set is labeled PUNCHDD. The data sets are fixed block with LRECL=80. Block size as specified on the DD statement is used; if not specified, the block size is set to 80. If the program is unable to open PUNCHDD, DFSDDLT0 issues abend 251. Here is an example of the PUNCHDD DD statement.

```
//PUNCHDD DD SYSOUT=B
```

Using the PREINIT Parameter for DFSDDLT0 Input Restart

You use the DFSDDLT0 restart function to restart a DFSDDLT0 input stream within the same dependent region. The PREINIT parameter in the EXEC statement invokes the restart function. Code the PREINIT parameter of DFSMPR as `PREINIT=xx`, where `xx` is the two-character suffix of the DFSINTxx proclib member. (PREINIT=DL refers to the default proclib member.)

The PREINIT process establishes a checkpoint field for each active IMS region. This field is updated with the sequence number of each GU call to an I/O PCB as it is processed. For this reason, sequence numbers are required for all such GU calls that are used. On a restart, if the checkpoint field contains a sequence number, the DFSDDLT0 stream starts at the next GU call to an I/O PCB following the sequence number in the checkpoint field; otherwise the DFSDDLT0 stream starts from the beginning.

The DFSDDLSI module and the default IMS.PROCLIB member, DFSINTDL, are shipped with IMS and are installed as part of normal IMS installation.

The following code shows examples of SYSIN/SYSIN2 and PREINIT.

```
//TSTPGM   JOB CARD
//DDLTTST  EXEC DFSMPR,PREINIT=DL
//MPP.SYSIN DD *
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
S11 1 1 1 1    TP     1                                                  01000000
OPTIONS SNAP= ,ABORT=9999                                                01000010
U*********************************************************************** 01000040
S11 1 1 1 1    TP     1                                                  01000050
L        GU                                                              01000060
E      OK                                                                01000070
S11 1 1 1 1    DBPCBXXX                                                  01000080
L        GU                                                              01000090
E        DATA    A   INIT-LOAD UOW                                       01000100
E   01    ROOTSEG1 0008A  0004D                                          01000110
S11 1 1 1 1    TP     1                                                  01000120
L        ISRT                                                            01000130
L  Z0080 DATA  -SYNC INTERVAL 1    SEG 1 -MESSAGE    1                 X01000140
L P      DATA  1111111111111111111111111111111111111111111111111111     01000150
L        ISRT                                                            01000160
L  Z0080 DATA  -SYNC INTERVAL 1    SEG 2 -END  EOM   1                 X01000170
L P      DATA  1111111111111111111111111111111111111111111111111111     01000180
U*********************************************************************** 01000190
U*   ENDING FIRST SYNC INTERVAL                                          01000200
U*********************************************************************** 01000210
L        GU                                                              01000220
E      QC                                                                01000230
L        GU                                                              01000240
E      OK                                                                01000250
S11 1 1 1 1    DBPCBXXX                                                  01000260
WTO GETTING DATA BASE SEGMENT 1 FROM DBPCBXXX                            01000270
L  U     GHU                                                             01000280
E        DATA  INIT-LOAD UOW. 1 A.P. 1                                   01000290
E      OK                                                                01000300
L  U0003 GN                                                              01000310
E      OK                                                                01000320
S11 1 1 1 1    TP     1                                                  01000330
L        ISRT                                                            01000340
L  Z0080 DATA  -SYNC INTERVAL 2    SEG 1 -MESSAGE    1                 X01000350
L P      DATA  2222222222222222222222222222222222222222222222211        01000360
L        ISRT                                                            01000370
L  Z0080 DATA  -SYNC INTERVAL 2    SEG 2 -END  EOM   1                 X01000380
```

```
L P      DATA  2222222222222222222222222222222222222222222211          01000390
U*************************************************************************** 01000400
U*   ENDING SECOND SYNC INTERVAL                                           01000410
U*************************************************************************** 01000420
L        GU                                                               01000430
E      QC                                                                 01000440
L        GU                                                               01000450
E      OK                                                                 01000460
S11 1 1 1 1    DBPCBXXX                                                   01000470
S11 1 1 1 1    TP     1                                                   01000480
L        ISRT                                                             01000490
L  Z0080 DATA  -SYNC INTERVAL 3    SEG 1 -MESSAGE    1                 X01000500
L P      DATA  333333333333333333333333333333333333333333333311          01000510
L        ISRT                                                             01000520
L  Z0080 DATA  -SYNC INTERVAL 3    SEG 2 -END  EOM   1                 X01000530
L P      DATA  333333333333333333333333333333333333333333333311          01000580
U*************************************************************************** 01000590
U*   ENDING THIRD SYNC INTERVAL                                            01000600
U*************************************************************************** 01000610
L        GU                                                               01000620
E      QC                                                                 01000630
//MPP.SYSIN2 DD *
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----<
ABEND                                                                     01000430
/*
```

Notes for the SYSIN/SYSIN2 and PREINIT examples shown above:

1. The PREINIT= parameter coded in the EXEC statement invokes the restart process.
2. When DFSDDLT0 starts processing, it substitutes the SYSIN2 ABEND statement for the statement in SYSIN with the same sequence number. (It is the GU call with sequence number 01000430.)
3. DFSDDLT0 begins with statement 01000000 and processes until it encounters the ABEND statement (statement number 01000430). The GU calls to the I/O PCB have already been tracked in the checkpoint field (statements 01000060, 01000220, and 01000240).
4. When DFSDDLT0 is rescheduled, it examines the checkpoint field and finds 01000240. DFSDDLT0 begins processing at the next GU call to the I/O PCB, statement 01000450.

   If the statement currently numbered 01000240 did not have a sequence number, DFSDDLT0 would restart from statement 01000000 when it was rescheduled.

Execution of DFSDDLT0 in IMS Regions

DFSDDLT0 is designed to operate in a DL/I or BMP region but can be executed in an IFP or MPP region. In a BMP or DL/I region, the EXEC statement allows the program name to be different from the PSB name. There is no problem executing calls against any database in a BMP or DL/I region.

In an MPP region, the program name must be the same as the PSB name. To execute a DFSDDLT0 program in an MPP region, you must give DFSDDLT0 the PSB name or an alias of the PSB named in the IMS definition. You can use a temporary step library.

In an MPP region or a BMP region with an input transaction code specified in the EXEC statement, DFSDDLT0 normally gets input by issuing a GU and GNs to the I/O PCB. DFSDDLT0 issues GU and GN calls until it receives the “No More Messages” status code, QC. If there is a SYSIN DD statement and a PRINTDD DD

statement in the dependent region, DFSDDLT0 reads input from SYSIN and SYSIN2, if present, and sends output to the PRINTDD. If the dependent region is an MPP region and the input stream does not cause a GU to be issued to the I/O PCB before encountering end-of-file from SYSIN, the program will implicitly do a GU to the I/O PCB to get the message that caused the program to be scheduled. If the input stream causes a GU to the I/O PCB and a “No More Messages” status code is received, this is treated as the end of file. When input is from the I/O PCB, you can send output to PRINTDD by coding a 1 or an A in column 2 of the STATUS statement.

Because the input is in fixed form, it is difficult to key it from a terminal. To use DFSDDLT0 to test DL/I in a message region, execute another message program that reads control statements stored as a member of a partitioned set. Insert these control statements to an input transaction queue. IMS then schedules the program to process the transactions. This method allows you to use the same control statements to execute in any region type.

Explanation of DFSDDLT0 Return Codes

A non-zero return code from DFSDDLT0 indicates the number of unequal comparisons that occurred during that time.

A return code of 0 (zero) from DFSDDLTO does not necessarily mean that DFSDDLT0 executed without errors. There are several messages issued by DSFDDLT0 that do not change the return code, but do indicate some sort of error condition. This preserves the return code field for the unequal comparison count.

If an error message was issued during the run, a message `ERRORS WERE DETECTED WITHIN THE INPUT STREAM. REVIEW OUTPUT TO DETERMINE ERRORS.` appears at the end of the DFSDDLT0 output. You must examine the output to ensure DFSDDLT0 executed as expected.

Hints on Using DFSDDLT0

This section describes loading a database, printing, retrieving, replacing, and deleting segments, regression testing, using a debugging aid, and verifying how a call is executed.

To Load a Database

Use DFSDDLT0 for loading only very small databases because you must to provide all the calls and data rather than have them generated. The following example shows CALL FUNCTION and CALL DATA statements that are used to load a database.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7---+-----<
O SNAP= ,ABORT=0
S 1 2 2 1 1
L        ISRT COURSE
L        DATA FRENCH
L        ISRT COURSE
L        DATA COBOL
L        ISRT CLASS
L        DATA 12
L        ISRT CLASS
L        DATA 27
L        ISRT STUDENT
L        DATA SMITH          THERESE
L        ISRT STUDENT
L        DATA GRABOWSKY      MARION
```

To Print the Segments in a Database

Use either of the following sequences of control statements to print the segments in a database.

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7---+-----<
.* Use PRINTDD, print call, compare, and PCB if compare unequal
.* Do 1 Get Unique call
.* Hold PCB compare, End step if status code is not blank, GA, GC, GK
.* Do 9,999 Get Next calls
S   2 2 2 1    DBDNAME
L        GU
EH8    OK
L   9999 GN
```

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7---+-----<
.* Use PRINTDD, print call, compare, and PCB if compare unequal
.* Do 1 Get Unique call
.* Hold PCB compare, Halt GN calls when status code is GB.
.* Do 9,999 Get Next calls
S   2 2 2 1    DBDNAME
L        GU
EH     OK
L   9999 GN
```

Both of the above examples request the GN to be repeated 9999 times. Note that the first example uses a COMPARE PCB of EH8 while the second uses a COMPARE PCB of EH.

The difference between these two examples is that the first halts the job step the first time the status code is not blank, GA, GC, or GK. The second example halts repeating the GN and goes on to process any remaining DFSDDLT0 control statements when a GB status code is returned or the GN has been repeated 9999 times.

To Retrieve and Replace a Segment

Use the following sequence of control statements to retrieve and replace a segment.

```
|----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----
 S 1 1 1 1 1    COURSEDB
 L        GHU   COURSE  (TYPE     =FRENCH)                              X
               CLASS   (WEEK     =27)                                  X
               STUDENT (NAME     =SMITH)
 L        REPL
 L        DATA  SMITH            THERESE
```

To Delete a Segment

Use the following sequence of control statements to delete a segment.

```
|----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----
 S 1 1 1 1 1          4
 L        GHU   COURSE  (TYPE     =FRENCH)                            X
               CLASS   *L                                            X
               INSTRUC (NUMBER   =444)
 L        DLET
```

To Do Regression Testing

DFSDDLT0 is ideal for doing regression testing. By using a known database, DFSDDLT0 can issue calls and then compare the results of the call to expected results using COMPARE statements. The program then can determine if DL/I calls are executed correctly. If you code all the print options as 2's (print only if comparisons done and unequal), only the calls not properly satisfied are displayed.

To Use as a Debugging Aid

When debugging a program, you usually need a print of the DL/I blocks. You can snap the blocks to a log data set at appropriate times by using a COMPARE statement that has an unequal compare in it. You can then print the blocks from the log. If you need the blocks even though the call executed correctly, such as for the call before the failing call, insert a SNAP function in the CALL statement in the input stream.

To Verify How a Call Is Executed

Because it is very easy to execute a particular call, you can use DFSDDLT0 to verify how a particular call is handled. This can be of value if you suspect DL/I is not operating correctly in a specific situation. You can issue the calls suspected of not executing properly and examine the results.

Bibliography

This bibliography includes all the publications cited in this book, including the publications in the IMS library.

Common Programming Interface Communications Reference, SC26-4399

IBM DATABASE 2 Application Programming and SQL Guide, SC26-4377

Language Environment for MVS & VM Installation and Customization, SC26-4817

Language Environment for MVS & VM Programming Guide, SC26-4818

MVS/ESA Application Development Guide: Authorized Assembler Programming, GC28-1645

MVS/ESA JES3 Conversion Notebook, GC23-0079

MVS/ESA System Programming Library: Application Development Guide, GC28-1852

MVS/XA Data Administration Guide, GC26-4140

OS PL/I Version 2 Programming Guide, SC26-4307

System Application Architecture Common Programming Interface: Resource Recovery Reference, SC31-6821

TSO/E Version 2 Procedures Language MVS/REXX Reference, SC28-1883

IMS/ESA Version 6 Library

SC26-8725	ADB	Administration Guide: Database Manager
SC26-8730	AS	Administration Guide: System
SC26-8731	ATM	Administration Guide: Transaction Manager
SC26-8727	APDB	Application Programming: Database Manager
SC26-8728	APDG	Application Programming: Design Guide
SC26-8726	APCICS	Application Programming: EXEC DLI Commands for CICS and IMS
SC26-8729	APTM	Application Programming: Transaction Manager
SC26-8732	CG	Customization Guide
SC26-9517	CQS	Common Queue Server Reference
SC26-8733	DBRC	Database Recovery Control Guide and Reference
LY37-3731	DGR	Diagnosis Guide and Reference
LY37-3732	FAST	Failure Analysis Structure Tables (FAST) for Dump Analysis
GC26-8736	IIV	Installation Volume 1: Installation and Verification
GC26-8737	ISDT	Installation Volume 2: System Definition and Tailoring
SC26-8740	MIG	Master Index and Glossary
GC26-8739	MC	Messages and Codes
SC26-8743	OTMA	Open Transaction Manager Access Guide
SC26-8741	OG	Operations Guide
SC26-8742	OR	Operator's Reference
GC26-8744	RPG	Release Planning Guide
SC26-8767	SOP	Sample Operating Procedures
SC26-8769	URDB	Utilities Reference: Database Manager
SC26-8770	URS	Utilities Reference: System
SC26-8771	URTM	Utilities Reference: Transaction Manager

Supplementary Publications

GC26-8738	LPS	Licensed Program Specifications
SC26-8766	SOC	Summary of Operator Commmands

Online Softcopy Publications

LK3T-2326	CDROM	IMS/ESA Version 6 Softcopy Library
SK2T-0730	CDROM	IBM Online Library: Transaction Processing and Data
SK2T-0710	CDROM	MVS Collection
SK2T-6700	CDROM	OS/390 Collection

© Copyright IBM Corp. 1974, 2000

Index

Special Characters

. (period) usage
  null or void placeholder 409
  parsing, transparent additions 409
  REXX 407
$$IMSDIR
  effect on performance 265
  updating 316
/DISPLAY command 247
/DISPLAY POOL command 265
&DPN= operand (DIV statement), specifying 302
/FORMAT command 179, 246
*mapname 413
/RDISPLAY command 248
/RESET command 293
/SET command 185
/TEST MFS command 179
!token
  IMSQUERY function 418
  STORAGE command 416

Numerics

12-byte time stamp, field in I/O PCB 48
274X
  defining to operate with MFS 180
  entering and exiting formatted mode 183, 184
  operating with MFS
    FTABs 194
    input modes 194
3180
  in partitioned format mode
    clearing the display 246
    paging 246
    restrictions 245
    scrolling 246
3270 Information Display System
  compatibility with 5550 512
  copy function
    bit 4 of SCA, byte 1 279
    description 237
    remote terminals 377
  defining system message field 221
  entering and exiting formatted mode 184
  increasing performance 266
  master terminal format
    display area 248
    literals defined for PF keys 248
    warning message area 248
    your input area 248
  multiple physical page input 200
  PA (program access) key, control functions 237
  printed page format control 221
  screen formatting 261
  selector pen
    effect on input fields 274
    for control functions 237
3270 Information Display System *(continued)*
  selector pen *(continued)*
    pen detect byte 383
    specifying 347, 377
  specifying attributes 328
3270 operator identification card reader
  application program device-dependent
    information 275
  CARD= operand (DEV statement) 345
  CARD= operand (DIV statement) 347
  system message field 221
3270P Printer
  defining to operate with MFS 180
  printed page format control 223
3275/3277 Display Station
  physical paging 208
  using default formats with 259
3276 Control Unit/Display Station
  physical paging 208
  using default formats with 259
3278 Display Station
  compatibility with 5550 512
  physical paging 208
  using default formats with 259
3279 Display Station, default formats 259
3290 Display Panel 180
  defining to operate with MFS 180
  in partitioned format mode 220
  in standard format mode 264
  screen formatting 263
3601 workstation, defining to operate with MFS 180
3770 Data Communication System
  defining to operate with MFS 180
  entering and exiting formatted mode 183
  printed page format control 222
3790 Communication System
  defining to operate with MFS 180
  operating with MFS
    FTABs 194
    input modes 194
5550 Family (as 3270) 214
  compatibility with other devices 512
  field outlining 382
  using DBCS/EBCDIC fields 214
  using DBCS fields 214
6670 Printer, defining to operate with MFS 180

A

abend statement 528
ACTVPID= operand (DPAGE statement)
  cursor positioning (3290 only) 220
  specifying 310, 366
  use 244
addressing environments 400, 405
addressing mode (AMODE) 57
AIB (application interface block) 11
  address return 54

© Copyright IBM Corp. 1974, 2000

AIB (application interface block) 11 *(continued)*
AIB identifier (AIBID) 49
AIBERRXT (reason code) 50
AIBOALEN (maximum output area length) 50
AIBOAUSE (used output area length) 50
AIBREASN (reason code) 50
AIBRSA1 (resource address) 50
AIBRSNM1 (resource name) 50
AIBSFUNC (subfunction code) 50
and program entry statement 54
defining storage 52
description 11, 51
DFSAIB allocated length (AIBLEN) 50
fields 49
interface, REXX 405
language interfaces, relationship with 12
mask 49, 50
return and reason codes 475
specifying 49
subfunction, setting 415
AIBERRXT (reason code) 50
AIBID (AIB identifier) field, AIB mask 49
AIBLEN (DFSAIB allocated length) field, AIB mask 50
AIBOALEN (maximum output area length) field, AIB mask 50
AIBOAUSE (used output area length) field, AIB mask 50
AIBREASN (reason code)
AIB mask, field 50
AIBREASN (reason code) field, AIB mask 50
AIBRSA1 (resource address) field, AIB mask 50
AIBRSNM1 (resource name) field, AIB mask 50
AIBSFUNC (subfunction code) field, AIB mask 50
AIBTDLI interface 51
allocate PSB call 90
alpha character generation 391
ALPHA statement (language utility) 311, 391
alternate destinations, sending messages to 124
alternate PCB
express 123
modifiable
changing the destination 125
description 123
use 124
using the CHNG call with 125
response 136
SAMETRM=YES 136
sending messages to other terminals 124
types and uses 48
use with program-to-program message switching 126
using the PURG call with 124
alternate PCB mask
description 48
format 48
alternate terminals, responding to 124
AO (automated operator) application
after status codes
GCMD call 75
GCMD call
status codes 75
AO (automated operator) application *(continued)*
GMSG call 94
ICMD call 97
RCMD call 111
API (application programming interface)
description 7
APPC conversational program
CPI-C driven 143
ending the conversation 144
message switching 139
modified IMS application 142
APPC environment 400
application interface block 11
application interface block (AIB)
See AIB 475
application programs 32
assembler language 32
C language 34
COBOL 37
Pascal 39
PL/I 42
scheduling 10
application view (diagram) 9
applications, sample 503
APSB call 90
description 90
format 90
parameters 90
restrictions 91
summary 439
usage 90
assembler language
application programming for 32
DL/I call formats 34
MPP coding 153
program entry 52
register 1 at program entry 52
skeleton MPP 153
ATTACH FM header 228, 301, 358
ATTACH manager
blocking algorithms 229
deblocking algorithms 202
ATTR= operand (DFLD statement)
parameters
ALPHA|NUM 377
nn 379
NO 379
NODET/DET/IDET 377
NODISP|HI 377
NOMOD|MOD 377
NOPROT/PROT 377
STRIP|NOSTRIP 378
YES 378
YES, nn 379
specifying 376
with copy lock 376
ATTR= operand (MFLD statement)
example 287
specifying 328
use 211

attribute data
  defaults 376
  input message fields
    ATTR= operand (MFLD statement) 328
    description 193
  output device fields
    ATTR= operand (MFLD statement) 328
    description 210
    dynamic modification 281
    for cursor positioning 220, 280
    specifying 376
attribute simulation
  description 212
  restrictions 281
  specifying 376
AUTH call
  description 59
  format 59
  I/O area format 60
  parameters 60
  restrictions 64
  summary 439
  usage 63
authorization call 59

B

backout point
  description 145
  intermediate (SETS/SETU) 149
  ROLB, ROLL, ROLS 145
basic checkpoint call 91
Basic CHKP call 91
  description 91
  format 91
  parameters 91
  restrictions 92
  summary 439
  usage 92
Basic edit
  IMS TM 169, 203
Basic Edit
  input message 20
  output message 21
  translation to uppercase 20
batch programs
  overview 10
  structure 10
block error message format 247
BOUND= operand (DO statement), specifying 368
BSAM (basic sequential access method)
  using with Spool API 79

C

C language
  __pcblist 52
  application programming 34
  DL/I call formats 37
  entry statement 52
  exit 52
  longjmp 52
C language *(continued)*
  passing PCBs 52
  return 52
  skeleton MPP 153
  system function 53
call functions, DL/I 537
CALL statement
  CALL DATA 531
  CALL FUNCTION 528
call summary, transaction management 439
CARD= operand (DEV statement)
  specifying 347
CARD= operand (DEV statement), specifying 345
categories, status codes 443
CEETDLI
  address return 54
  program entry statement 54
CELLSIZE= operand (PD statement), specifying 388
change call 64, 125
checkpoint call 534
checkpoint call, basic 91
checkpoint call, symbolic 92
CHKP call function 534
CHNG call
  and OTMA environment 67
  description 64, 125
  format 64
  parameters 65
  restrictions 72
  summary 439
  usage 66, 515
  using PURG with 125
  with directed routing 130
CHNG call function 535
CLEAR key 261
CLEAR PARTITION key 261
CMD call
  description 72
  examples 73
  format 72
  parameters 72
  restrictions 73
  summary 439
  usage 73
CMD call function 535
COBOL
  application programming 37
  DL/I call formats 39
  skeleton MPP 155
codes, return and reason
  reference 475, 491
codes, status
  explanations 452
  tables 443
coding DC calls and data areas 152
  in assembler language 153
  in C Language 153
  in COBOL 155
  in Pascal 157
  in PL/I 160
  skeleton MPP 152, 153, 155, 157, 160

Command (CMD) call
See CMD call 72
COMMENT statement
conditional (T) 549
unconditional (U) 549
commit point 145
communicating with other IMS TM systems 128
COMPARE statement
COMPARE AIB 551
COMPARE DATA 550
COMPARE PCB 552
introduction 549
compatibility
3270 printer and SLU 1 511
converting device definitions to SLU P 512
SLU P 512
compilation statements
ALPHA 311, 391
COPY 311, 391
EJECT 396
END 396
EQU 392
PRINT 395
RESCAN 393
SPACE 395
STACK 394
summary of statements 315
syntax 311
syntax errors 313
SYSIN 311
SYSLIB 311
SYSPRINT 311
TITLE 395
UNSTACK 394
COMPR= operand (DIV statement)
specifying 359
COMPR= operand (DIV statement), specifying 303
concatenated equates 393
COND= operand
DPAGE statement
specifying 363
COND= operand (DPAGE statement), specifying 307
COND= operand (LPAGE statement), specifying 319
control blocks, MFS 171
chained control blocks 251
summary 171
testing 179
conversational processing
coding necessary information 144
continuing the conversation 136
ending the conversation and passing control 139
example 131
for APPC/IMS 141
message formats 136
overview 131
passing control and continuing the conversation 137
replying to the terminal 136
restrictions 135, 137
ROLB call 133
ROLL call 133
conversational processing *(continued)*
ROLS call 133
steps in a conversational program 134
structure 132
conversion
3270 device format, example 509
device formats 508
copy function
bit 4 of SCA, byte 1 279
description 237
remote terminals 377
COPY statement (language utility) 311, 391
CTL (PUNCH) statement 557
CURSOR= operand
DPAGE statement 364
CURSOR= operand (DPAGE statement), specifying 308
cursor position input 191
cursor positioning
for input messages 220, 273
for output messages
CURSOR operand 308
dynamic 220
in DPAGE statement 364
specifying attributes 280

D

data capture 51
data mapping, define with MAXDEF command 410
database recovery
backing out 146
DB/DC environment
application view (diagram) 8
programming considerations 8
DB2 (DATABASE 2)
with IMS TM 28
DBCS (double byte character set)
definition 214
types of fields 214
DBCS/EBCDIC mixed fields
description 214
horizontal tab (SCS1 device) 218
input control 218
SO/SI control characters in 214
DBCS/EBCDIC mixed literals
description 216
DCCTL environment
programming considerations 8
deallocate PSB call 93
debugging, IMSRXTRC 410
default system control area 209
deferred program switch 133
define a data mapping with MAXDEF command 410
delete call 535
delete record 342
DEQ call function 535
design objectives, application 251
designator character 274
destination of modifiable alternate PCBs 125
DEV statement 195

DEV statement 195 *(continued)*
CARD= operand 347
DSCA= operand 343
FEAT= operand 258, 345
FORMS= operand 348
FORS= operand 226
FTAB= operand 195, 341
HTAB= operand 222, 348
LDEL= operand 342
MODE= operand 341
PAGE= operand 221, 342
PDB= operand 351
PEN= operand 347
PFK= operand 346
SLDI= operand 350
SLDP= operand 350
SLDx= operand 223
specifying 330
SUB= operand 201, 351
SYSMSG= operand 348
TYPE= operand 258, 339
VERSID= operand 350
VT= operand 223, 349
VTAB= operand 223, 349
WIDTH= operand 222, 348
device characteristics table 339
device control characters 204
device feature selection 258
device format conversion 508
device formats, default 259
device input format 171, 250
device output format 171, 250
device page 191
devices supported by MFS 180
DFLD (device field statement) 324
iterative processing 324, 369
LTH= operand 375
OPCTL= operand 384
PASSWORD parameter 374
PEN= operand 383
POS= operand 375
printing generated DFLD statements 368
SCA parameter 375
SLDI= operand 384
SLDP= operand 384
DFS.EDT 292, 293
DFS.EDTN 292, 293
DFS057I block error message 247
DFSAPPC 139
format 140
message switching 139
option keywords 140
DFSDDLT0 (DL/I Test Program) 525
DFSDF1 246
DFSDF2 246
DFSDF4 246
DFSDSP01 247
DFSIGNI 246, 247
DFSIGNJ 246, 247
DFSIGNN 246, 247
DFSIGNP 246, 247
DFSM0 248
DFSM01 246
DFSM02 247
DFSM03 247
DFSM04 247
DFSM05 246
DFSMI1 246
DFSMI2 246
DFSMI4 247
DFSREXXU (Example User Exit Routine)
description 421
DFSSAM01 (Loads the Database) 431
DFSUDT0x (device characteristics table) 179
description 183
MFS Device Characteristics Table utility 249
specifying screen size 339
diagnosing multiple parsing error return codes 515
DIF (device input format) 171
definition 250
description 171
input formatting functions 186
language statements used to create 294
DEV 330
DFLD 370
DIV 294, 351
DO 367
DPAGE 304, 360
ENDDO 385
FMT 330
FMTEND 385
PPAGE 366
RCD 369
summary 314
relationship to other control blocks 251
selection 258
directed routing 128
distributed presentation management 182
DIV statement 196
&DPN= operand 302
COMPR= operand 303, 359
DPN= operand 358
HDRCTL= operand 225, 357
NOSPAN= operand 298
NULL= operand 196, 298, 355
OFTAB= operand 358
output mode 229
specifying 302
variable-length output data stream 230
OPTIONS= operand 224, 299, 355
PRN= operand 302, 358
RCDCTL= operand 224, 298, 354
RDPN= operand 301, 358
RPRN= operand 302, 358
SPAN= operand 298
TYPE= operand 297, 354
DL/I call functions 534, 537
special DFSDDLT0
END 547
SKIP 547
STAK 547
START 547

DL/I call functions 534, 537 *(continued)*
supported
CHKP 534
CHNG 535
CMD 535
DEQ 535
DLET 535
FLD 535
GCMD 535
GHN 535
GHNP 535
GHU 535
GMSG 535
GN 535
GNP 535
GU 535
ICMD 535
INIT 535
INQY 535
ISRT 536
LOG 536
POS 536
PURG 536
RCMD 536
REPL 536
ROLB 536
ROLL 536
ROLS 536
ROLX 536
SETO 536
SETS 536
SNAP 537
STAT 537
SYNC 537
XRST 537
DL/I calls 34
relationships to PCB types
ALT PCBs 44
I/O PCBs 44
sample call formats 34
assembler language 34
C language 37
COBOL 39
Pascal 42
PL/I 44
usage 12
DL/I calls (general information)
REXXTDLI 404
DL/I calls for transaction management
AUTH call 59
call summary 439
CHNG call 64
CMD call 72
GCMD call 74
general description 12
GN call 75
GU call 76
ISRT call 78
PURG call 80
SETO call 82
summary 12
DL/I language interfaces 31
overview 31
supported interfaces 31
DL/I program structure 10
DL/I return codes (REXX) 405
DL/I system service calls 89
APSB call 90, 91
Basic CHKP call 91, 92
call summary 440
DPSB call 93, 94
GSCD Call 96, 97
INIT call 99, 101
INQY call 102, 109
LOG call 109, 111
ROLB call 112, 113
ROLL Call 114
ROLS call 114, 116
SETS call 116, 117
SETU call 116, 117
Symbolic CHKP call 92, 93
SYNC call 118
XRST call 118
DL/I Test Program (DFSDDLT0)
control statements 525, 564
execution in IMS regions 567, 568
explanation of return codes 568
hints on usage 568, 570
JCL requirements 564, 567
overview 525
restarting input stream 566
DLET call function 535
DLIINFO
. (period) usage 409
REXX extended command 408, 409
DLITCBL 53
DLITPLI 53
DO statement
BOUND= operand 368
SUF= operand 323, 368
DOCMD exec 432
DOF (device output format) 171
associated MFS functions 204
definition 250
description 171
language statements used to create 294
DEV 330
DFLD 370
DIV 294, 351
DO 367
DPAGE 304, 360
ENDDO 385
FMT 330
FMTEND 385
PPAGE 366
RCD 369
summary 314
relationship to other control blocks 251
selection 258
double byte character set 214
DPAGE 191
ACTVPID= operand 244, 310, 366

DPAGE 191 *(continued)*
COND= operand 307, 363
CURSOR= operand 308, 364
FILL= operand 307, 363
input 191
MULT= operand 308, 364
OFTAB= operand 365
output mode 229
specifying 309
variable-length output data stream 230
ORIGIN= operand 309, 365
output 206
PD= operand 310, 366
SELECT= operand 310, 366
selection, with conditional data 201
DPM (distributed presentation management)
control character translation 205, 278
deleting nulls on input 196
increasing performance 267
naming conventions 227
output message header examples 225
using 182
version identification 250
with ISC 182
DPN= operand (DIV statement)
specifying 358
DPN field
control block linkages 258
DIV statement 302
literal specification 358
MFS formatting 185
DPSB call 93
description 93
format 93
parameters 94
restrictions 94
summary 439
usage 94
DSCA (default system control area) 209
autopaged output 228
description 209
destroying screen format 221
ERASE/DO NOT ERASE option 279
use 242
DSCA= operand (DEV statement), specifying 343
DSN (data structure name) 235
dynamic attribute modification, output message formats
default attributes 211
specifying attributes 280
specifying extended field attributes 282
dynamic modification of EGCS data 289

E

E (COMPARE) statement 549
EATTR= operand (DFLD statement)
example 287
specifying 379
use 211
edit routines
Basic Edit 20
ISC 20
edit routines *(continued)*
MFS 20
edit routines, IMS-supplied
field edit routine 192
segment edit routine 192
EGCS (extended graphic character set) 212
/EBCDIC data, dynamic modification 289
description 212
SO/SI framing characters 213
specifying 381
use with selector pen 275
EJECT statement (language utility) 396
END call function 547
end multiple page input request 237
END statement (language utility) 396
ENDDO statement
specifying to terminate DFLD statements 385
specifying to terminate MFLD statements 329
ending a conversation and passing control to another program 139
ENDMPPI request 237
specifying 384
specifying PF key function 347
entry point
assembler language 52
C language 52
COBOL 53
overview 52
Pascal 53
PL/I 53
environment (REXX)
address 400, 405
determining 408
extended 405
EQU statement (language utility statement) 392
equate processing 392
erase all unprotected option (SCA/DSCA) 262
ERROR key 201
error routines
I/O errors 14
programming errors 14
system errors 14
types of errors 14
examples
conversational processing 131
DFSDDLT0 statements
COMMENT 549
DATA/PCB COMPARE 554
DD 565
DL/I call functions 537
IGNORE 556
OPTION 557
PUNCH 559
STATUS 562
SYSIN, SYSIN2, and PREINIT 566
WTO 563
WTOR 564
exceptional conditions 14
EXEC statement, compilation control 312
EXEC statement, operands
DEVCHAR= 249, 317

EXEC statement, operands *(continued)*
DIRUPDT= 316
LINECNT= 316
STOPRC= 316
EXEC statement, parameters
COMP/NOCOMP 316
COMPRESS/NOCOMPRESS 316
DIAG/NODIAG 316
SUBS/NOSUBS 316
XREF/NOXREF 316
EXECIO
example 431
managing resources 400
exit routines, specifying IMS-provided field edit 329
express alternate PCB 123
extended attribute data 193
input message fields 193
output devices, dynamic modification 211
extended commands 408
extended environment 405
extended functions 418
extended graphic character set 212
extended recovery facility 221

F

Fast Path, with MFS 185
FEAT= operand (DEV statement), specifying 258, 345
field edit exit routine
specifying 329
use 192
field format
input message 273
output message 277
field tab 341
example 195
forced FTABs, FORCE parameter (FTAB= operand) 341
mixed FTABs 342
specifying 341
file I/O 431
FILL= operand
DPAGE statement
specifying 363
DPAGE statement, specifying 307
MFLD statement, specifying 329
MSG statement, specifying 318
fill characters
input message fields
MFS treatment 194
specifying 329
output device fields
MFS treatment 208
specifying 307, 318, 363
Fill=NULL 191
FIN (Finance Communication System)
defining to operate with MFS 180
workstation
entering and exiting formatted mode 184
FTABs 194
input modes 194
operating with MFS 194
FIN (Finance Communication System) *(continued)*
workstation *(continued)*
physical page positioning 309, 365
Finance Communication System 184
FLD call function 535
FMT statement, specifying 330
FMTEND statement, specifying 385
force format write option (SCA/DSCA) 262
format, message 183
input 183
device-dependent considerations 273, 278
output 265
output device-dependent considerations 275, 278
format library member selection 258
format set
IMS-provided format sets 246
testing
/FORMAT command 179
/TEST MFS command 179
FORMS= operand (DEV statement), specifying 348
forms control (FIJP, FIPB, FIFP, SCS1) 342
FORS= operand (DEV statement), use for DPM 226
framing characters (SO/SI) 213
FTAB= operand (DEV statement)
ALL parameter 196
description 194
forced FTABs, FORCE parameter 195
mixed FTABs, MIX parameter 195
specifying 341
full format write 261

G

GCMD call
description 74
format 74
parameters 74
restrictions 75
status codes 75
summary 439
usage 74
GCMD call function 535
Get calls
function 535
Get Command (GCMD) call
See GCMD call 74
Get Message (GMSG) call
See GMSG call 94
get next call 75
get system contents directory call 96
get unique call 76
GMSG call 96
description 94
format 94
parameters 94
restrictions 96
use 95
GN call
description 75
format 75
parameters 75
restrictions 76

GN call *(continued)*
summary 439
usage 76
GPSB (generated program specification block), format 54
GRAPHIC= operand (SEG statement)
specifying 322
use 205, 278
group name, field in I/O PCB 47
GSCD call
description 96
format 96
parameters 96
restrictions 97
summary 439
usage 97
GU call
description 76
format 76
parameters 76
restrictions 78
summary 439
usage 77

H

HDRCTL= operand
DIV statement
specifying 357
HDRCTL= operand (DIV statement), use 225
HTAB= operand (DEV statement)
specifying 348
use 222

I

I/O area
for XRST 120
in C language 36
specifying 51
I/O area format, AUTH call 60
I/O PCB mask
12-byte time stamp 48
description 11
general description 45
group name field 47
input message sequence number 47
logical terminal name field 46
message output descriptor name 47
specifying 45
status code field 46
userid field 47
ICMD call 99
commands that can be issued 99
description 97
format 97
parameters 97
restrictions 99
use 99
IF statement
parameters
DATA 390
ENDMPPI 390
LENGTH 390
NEXTLP 390
NEXTMSG 390
NEXTMSGP 390
NEXTPP 390
NOFUNC 390
PAGREQ 390
specifying 389
IGNORE (N or .) statement 556
immediate program switch 134
IMS application programs, standard 142
IMS.FORMAT 179
compression 178
member selection 258
use 179
IMS password
PASSWORD statement 320
specifying 374
IMS-provided formats
/DISPLAY command format 247
DFS057I block error message format 247
multisegment format 247
multisegment system message format 246
output message default format 246
system message format 246
IMS.REFERAL 179
compression 178
handling of ITBs 179
IMS.RESLIB 249
IMS.TFORMAT, use 178
IMS TM
DB2 considerations 28
password 193
IMSQUERY extended function
arguments 419
usage 418
IMSRXTRC command 408, 410
INDEX function (service utility), overview 179
infinite loop, stopping 404
INIT call
description 99
determining data availability 100
format 100
parameters 100
performance considerations 101
summary 439
usage 100
INIT call function 535
initialize call 99
input message 186
field attribute data 193
field tab (FTAB) 194
fill characters 194
format 15
formatting options 186
IMS TM password 193
input modes 194

input message 186 *(continued)*
input substitution character 201
literal fields 192
MFS 22
with multiple physical pages 200, 237
input message format
device-dependent information 273, 278
field and segment format 273
formatting options, examples 187
input message sequence number, field in I/O PCB 47
input modes
record mode
description 194
treatment of nulls 197
with ISC 202
specifying 341
stream mode
description 194
treatment of nulls 197
with ISC 202
inquiry call 102
INQY call
description 102
format 102
parameters 102
querying
data availability 106
environment 107
PCB, using null subfunction 103
PCB address 108
restrictions 109
return and reason codes 109
summary 439
usage 103
INQY call function 535
insert call 78
intersystem communication 185
ISC (intersystem communication)
ATTACH FM header 228, 301
blocking algorithms 229
defining to operate with MFS 182
editing input messages 21
editing output messages 21
entering and exiting formatted modes 185
increasing performance 267
input format control
input modes 201
paging requests 201
message waiting system literal 327
MFS definitions 505
output format control
data structure name 235
for paging messages 228
trailing blank compression 232
variable length output 230
output modes 229
subsystem definition 185
use of DPN field 185, 258
use of RDPN field 185, 258
ISC (Intersystem Communication)
ATTACH FM header 358
ISRT call
description 78
format 78
message call
in conversational programs 136
use with SPAs 136
parameters 78
restrictions 80
Spool API functions 79
summary 439
usage 79
ISRT call function 536
Issue Command (ICMD) call
See ICMD call 97
ITB (intermediate text block), relationship between ITBs and control blocks 178
iterative processing (MFLD/DFLD)
DO statement 324, 367
ENDDO statement 329, 385
PRINT GEN effects 395
RCD statement with DFLD 369
restrictions 368
IVPREXX exec 436
IVPREXX sample application 403

J

JCL (job control language), requirements 564, 567
JUST= operand (MFLD statement), specifying 328
justification
of input messages 186
specifying 328

L

L (CALL) statement 528
LANG= Option on PSBGEN for PL/I Compatibility with Language Environment 56
Language Environment
characteristics of CEETDLI 55
LANG = option for PL/I compatibility 56
supported languages 55
Language Environment for MVS & VM, with IMS 55
language independent interfaces 12
language unique interfaces 12
language utility 178
LDEL= operand (DEV statement), specifying 342
length field 189
line width 348
literal fields
input message, default literals 192
output message
length, padding to maximum 325
length, password parameter 374
specifying length 327
system literals 210
truncating literals 369
with ISC 327
LL field 134
in input message 15
in output message 16
LOG call 109

LOG call 109 *(continued)*
description 109
examples 111
format 109
on LOG I/O area 111
parameters 109
restrictions 111
restrictions on I/O area 111
summary 439
usage 111
LOG call function 536
logical page 191
selection
conditional 360
logical page advance request 237
logical page request 236
logical terminal name field, I/O PCB 46
LPAGE 191
input 191
input, conditional LPAGE selection 304
operands 319
COND= 319
NXT= 320
PROMPT= 320
SOR= 319
output 205
conditional selection 319
format 275
LTH= operand (DFLD statement), specifying 375
LTH= operand (MFLD statement), specifying 327
LU 6.2
application programs 9
conversations 141
LU 6.2 User Edit Exit 28
LUDEFN= operand (PDB statement), specifying 387
LUSIZE= operand (PDB statement), specifying 386

M

MAP definition (MAPDEF) 408, 410
map name 413
MAP reading (MAPGET) 408, 413
MAP writing (MAPPUT) 408, 413
mapping
MAPDEF 410
MAPGET 413
MAPPUT 413
MDT (modified data tag) 221
message
editing
description 19
input message 20
LU 6.2 devices 28
output 21
skipping line 20
using MFS 22
from terminals 14
I/O PCB 19
in conversations 136
input 15, 22
LU 6.2 devices 28
output 16, 27
message *(continued)*
printing 20
processing 17
receiving by program 14
result 19
sending to other application programs 126
using LU 6.2 User Edit Exit 28
message advance protect 237
message advance request 237
message calls
call summary 439
summary 12
message-driven BMP
retrieving messages from host 77
message editor 179
message format service 169
message formatting options
input
description 186
examples 187
performance factors 265
output
description 204
effects on segments 277
performance factors 265
specifying 318
message formatting service 20
message input descriptor 171
message output descriptor 171
message output header 357
message processing program 153
MFLD (message field statement) 186
ATTR= operand 328
FILL= operand 329
function 186
iterative processing 322, 324
JUST= operand 328
LTH= operand 327
printing generated MFLD statements 323
MFS (message format service)
components 177
editing message 20
editing output messages 21
example 174
input message 22
introduction 169
message editor 177
online performance 170
option 1 format 24
option 2 format 25
option 3 format 26
output message 27
pool manager 178
remote programs 181
supported devices 180
MFS bypass
printer byte restriction 292
protected and unprotected messages 243
specifying for 3270 or SLU 2 292
specifying for 3290 with partitioning 294

MFS Device Characteristics table (DFSUDT0x), description 249
MFS language utility 178
compilation statements 311
ALPHA 311, 391
COPY 311, 391
EJECT 396
END 396
EQU 392
invalid statement sequence 313
PRINT 395
RESCAN 393
SPACE 395
STACK 394
summary 315
syntax 311
syntax errors 313
SYSIN 311
SYSLIB 311
SYSPRINT 311
TITLE 395
UNSTACK 394
construction of member names 258
functions 178
modes 178
statistics maintained 178
treatment of EGCS input/output 213
use of MFS libraries 178
MFS libraries 178
IMS.TFORMAT 178
online change 178
MFS message editor 179
MFS pool manager
storage management 179
MFS service utility, INDEX function 179
MFSTEST procedure (language utility)
pool manager 178, 180
use of IMS.TFORMAT library 178
MID (message input descriptor) 171
description 171
input formatting functions 186
language statements used to create 314
DO 322
ENDDO 329
LPAGE 319
MFLD 323
MSG 317
MSGEND 330
PASSWORD 320
SEG 321
summary 314
relationship to other control blocks 251
MIDs and MODs, chaining with NXT= operand (MSG statement) 318
mixed FTABs, MIX parameter (FTAB= operand) 342
mixed-language programming 57
MOD (message output descriptor) 171
associated MFS functions 204
description 171
language statements used to create 314
DO 322
MOD (message output descriptor) 171 *(continued)*
language statements used to create 314 *(continued)*
ENDDO 329
LPAGE 319
MFLD 323
MSG 317
MSGEND 330
PASSWORD 320
SEG 321
summary 314
name specification 291
relationship to other control blocks 251
MODE= operand (DEV statement), specifying 341
modifiable alternate PCBs
changing the destination 125
CHNG call 125
description 124
modified application program
MSC 143
remote execution, MSC 143
modified data tag (MDT) 221, 377
MPP (message processing program)
coding in assembler language 153
coding in C language 153
coding in COBOL 155
coding in Pascal 157
coding in PL/I 160
coding necessary information 152
input 152
skeleton MPP 152
MSC (multiple systems coupling)
conversational programming 138
description 128
directed routing 128
receiving messages from other IMS TM systems 129
sending messages to other IMS TM systems 130
MSG statement
FILL= operand 318
NXT= operand 318
OPT= operand 318
PAGE= operand 318
SOR= operand 318
TYPE= operand 317
MSGEND statement
specifying 330
MULT= operand (DPAGE statement)
specifying 364
MULT= operand (DPAGE statement), specifying 308
multiple physical pages
input messages
specifying 364
multiple physical pages, input messages
description 200
specifying 308
terminating (ENDMPPI request) 237
multiple systems coupling 128
multisegment format 247
MVS environment 400

MVS/ESA, extended addressing capabilities
addressing mode (AMODE) 57
DCCTL environment 57
preloaded program 57
residency mode (RMODE) 57

N

NEXTLP request
description 237
operator control table function 236
specifying 384
specifying PF key function 347
NEXTMSG request
description 237
specifying 384
specifying PF key function 347
NEXTMSGP request
description 237
specifying 384
specifying PF key function 347
NEXTPP request 237
specifying 384
specifying PF key function 347
use 237
nonstandard character 312, 391
NTO (Network Terminal Operations) 180
null
coding in COBOL 277
compression
example 189
prevention 322
specifying 303, 359
deleting on input (DPM) 196
fill character
input message fields 186, 329
output device fields 209, 318
segment, output 276
NULL= operand (DIV statement)
example 196
options 196
specifying 298, 355
NXT= operand (LPAGE statement), specifying 320
NXT= operand (MSG statement), specifying 318

O

O (OPTION) Statement 556
OFTAB= operand
DIV statement
specifying 358
DPAGE statement
specifying 365
OFTAB= operand (DIV statement), specifying 230, 302
OFTAB= operand (DPAGE statement), specifying 230, 309
OID 221
online change (utility), description 178
online performance 170, 265
OPCTL= operand (DFLD statement), specifying 384
Open Transaction Manager Access
CHNG call 67
PURG call 81
SETO call 85
operator control of MFS 235
operator control tables
functions
ENDMPPI request 237
NEXTLP request 237
NEXTMSG request 237
NEXTMSGP request 237
NEXTPP request 237
PAGEREQ request 237
language statements used to create
IF 389
TABLE 389
TABLEEND 391
OPCTL= operand (DFLD statement) 384
operator logical paging
description 207, 236
format design considerations 236
in partitioned format mode, 3180 245
in partitioned format mode, 3290 244
specifying 318
transaction codes and page requests 236
OPT= operand (MSG statement), specifying 318
OPTION statement 556
OPTIONS= operand
DIV statement
specifying 355
OPTIONS= operand (DIV statement)
effects on performance 267
specifying 299
use 224
use with ISC 228
options list parameter 67
CHNG call 67
advanced print function 67
APPC 69
SETO call 85
advanced print function 85
APPC 85
ORIGIN= operand
DPAGE statement
specifying 365
ORIGIN= operand (DPAGE statement), specifying 309
OTMA, processing conversations with 145
outlining values
overline 382
underline 382
vertical line 382
output field tab separator, rules for inserting 230
output message 185
cursor positioning 220
default system control area 209
device field attributes 210
extended field attributes for devices 211
extended graphic character set (EGCS) 212
fill characters for device fields 208
format 16
formatting options 204
description 204

output message 185 *(continued)*
specifying 318
header 185
length 354
structure and content 357
how MFS formats messages 204
how MFS is selected 203
literal fields 210
logical paging 205
mixed DBCS/EBCDIC fields 214
operator logical paging 207
physical paging 207
printing 20
prompt facility 221
sending 127
sequence number 325
system control area (SCA) 209
to other application programs 126
to other IMS TM systems 130
truncation 204
using Basic Edit 21
using MFS 27
with directed routing 130
output message format
default 246
device-dependent information 275, 278
overline, on fields 382

P

PAGE= operand (DEV statement)
specifying 342
use 221
PAGE= operand (MSG statement), specifying 318
page advance request 237
page bit 206
PAGEREQ function 236
paging, operator logical
description 236
format design considerations 236
in partitioned format mode, 3180 245
in partitioned format mode, 3290 244
specifying 318
transaction codes and page requests 236
PAGINGOP= operand (PDB statement), use 244
parmcount 161
parsing error return codes 515
PART exec 429
partition
activating 220
considerations for defining 263
defining 257
descriptor (PD) 257
descriptor block (PDB) 257
initialization options
for the 3180 245
for the 3290 244
uses 263
partition set, description 257
partition set, language statements used to create
PD 387
PDB 386
partition set, language statements used to create *(continued)*
PDBEND 389
PARTNAME exec 431
PARTNUM exec 430
Pascal
application programming 39
DL/I call formats 42
entry statement 53
passing PCBs 53
skeleton MPP 157
passing a conversation to another IMS TM system 138
passing control
restrictions 137
to a conversational program 137
to another program in a conversation 137
password, IMS
description 193
specifying 374
PASSWORD parameter (DFLD statement), specifying 374
PASSWORD statement, specifying 320
PCB, express alternate 123
PCB (program communication block)
DLIINFO call 409
language interfaces, relationship with 12
mask 11
masks
I/O PCB 45
types 55
PCB lists 54
PCB parameter list in assembler language MPPs 153
PCBINFO exec 426
PCBs, alternate 125
PCBs, modifiable 125
PD= operand
DPAGE statement
specifying 366
PD= operand (DPAGE statement), specifying 310
PD statement (partition definition)
CELLSIZE= operand 388
PID= operand 387
PRESPACE= operand 388
SCROLLI= operand 388
specifying 387
use 257
VIEWLOC= operand 388
VIEWPORT= operand 387
WINDOWOF= operand 388
PDB= operand (DEV statement), specifying 351
PDB (partition descriptor block)
function 257
language statements used to create
PD 257
PDBEND 315
summary 315
LUDEFN= operand 387
LUSIZE= operand 386
PAGINGOP= operand 244
PDB (partition descriptor block),
SYSMSG= operand 386

PDBEND statement, specifying 389
PEN= operand (DEV statement), specifying 347
PEN= operand (DFLD statement), specifying 383
performance factors
  3270 or SLU 2 266
  all devices 265
  large screen 3270 or SLU 2 devices 267
period usage 407
PFK= operand (DEV statement), specifying 346
physical page positioning (FIN) 309, 365
physical paging
  description 207
  POS= operand (DFLD statement) 376
  specifying multiple input pages 308, 364
PID= operand (PD statement), specifying 387
PL/I
  application programming 42
  DL/I call formats 44
  entry statement restrictions 160
  MPP coding notes 160
  optimizing compiler 160
  passing PCBs 53
  pointers in entry statement 53
  skeleton MPP 160
pool manager 178
  MFS 178
    buffer pool 179
    control block management 179
    description 178
  MFSTEST, description 178, 180
POS= operand (DFLD statement), specifying 375
POS call function 536
PPAGE statement, specifying 366
PREINIT parameter, input restart 564
preset destination mode 185
PRESPACE= operand (PD statement), specifying 388
print mode 222
PRINT statement (language utility) 395
printed page format control
  bottom margin 223, 349
  horizontal tabbing 222
  left margin position 222, 349
  line density 223, 350
  line width 222
  number of lines per page 342
  page depth 222
  top margin 223, 349
  vertical tabbing 223
PRN= operand
  DIV statement
    specifying 358
PRN= operand (DIV statement), specifying 302
processing a message 17
program communication block 11, 45
program function keys (3270)
  literals for master terminal format 248
  specifying 346
program structure
  batch 10
  conversational 132
program tab function
  3270 or SLU 2 209
  fill character 262, 318
program-to-program message switching
  conversational 137
  nonconversational 126
  restrictions 126
  security checks 126
programmed symbol
  buffers 268
  feature 211
  solving problems 269
  specifying local ID 380
PROMPT= operand (LPAGE statement), specifying 320
prompt facility for output messages 221
protecting the screen
  PROTECT option 243
  specifying parameter on DLFD statement 377
PSB (program specification block), format 54
PT (program tab) function
  3270 or SLU 2 209
  fill character 262, 318
PUNCH statement 557
PURG call
  and OTMA environment 81
  description 80, 124
  parameters 80
  restrictions 82
  Spool API 82
  summary 440
  usage 81
  using CHNG with 125
PURG call function 536
purge call 80

Q

queues 327

R

RACF signon security 47
RACROUTE SAF 48
randomizing routine, FM status code 462
RCD statement, specifying 369
RCDCTL= operand
  DIV statement
    specifying 354
RCDCTL= operand (DIV statement)
  specifying 298
  use 224
RCMD call 112
  description 111
  format 111
  parameters 111
  restrictions 112
  use 112
RDPN= operand
  DIV statement
    specifying 358
RDPN= operand (DIV statement), specifying 301

RDPN (return destination process name)
in input message MFLD 358
specifying in MFLD statement 301
use on Finance or SLU P workstations 258
use with ISC subsystem communication 185
reason code, checking 13
reason codes, AIB 475
receiving messages, from other IMS TM systems 129
record mode
description 194
input example 198
specifying 341
treatment of nulls 197
with ISC 202
remote programs, defining 181
REPL call function 536
replying to one alternate terminal 124
replying to the terminal in a conversation 136
RESCAN statement (language utility) 393
residency mode (RMODE) 57
restart call 118
restarting your program
XRST call 120
retrieval call, status code 14
Retrieve Command (RCMD) call
See RCMD call 111
return and reason codes
quick reference tables 475
return code, checking 13
return codes
AIB 475
REXX
. (period) usage 407
calls
return codes 405
summary 405
syntax 405
commands
DL/I calls 404
summary 404
DL/I calls, example 407
execs
DFSSAM01 431
DOCMD 432
IVPREXX 436
PART 429
PARTNAME 431
PARTNUM 430
PCBINFO 426
SAY 425
IMSRXTRC, trace output 410
REXX, IMS adapter
. (period) usage 409
address environment 400
description 399
DFSREXX0 program 399, 403
DFSREXX1 399
DFSREXXU user exit 399
DFSRRC00 403
diagram 402
DL/I parameters 406
REXX, IMS adapter *(continued)*
entry parameters 421
environment 408, 421
example execs 425
exec name, choosing 421
feedback processing 406
I/O area 406
installation 399, 421
IVPREXX exec 403
IVPREXX PSB 400
IVPREXX setup 400
LLZZ processing 406
LNKED requirements 399
non-TSO/E 399
programs 399
PSB requirements 399
sample generation 400
sample JCL 400
SPA processing 406
SRRBACK 399
SRRCMIT 399
SYSEXEC DD 399, 400
system environment 399, 400
SYSTSIN DD 400
SYSTSPRT DD 399, 400
TSO/E restrictions 399
TSO environment 399
user exit routine (DFSREXXU) 421
ZZ processing 406
REXXIMS commands 410, 413, 418
DLIINFO 408, 409
IMSRXTRC 408, 410
MAPDEF 408
MAPGET 408
MAPPUT 408, 413
SET 408, 414
SRRBACK 408, 415
SRRCMIT 408, 415
STORAGE 408, 416
WTL 408, 417
WTO 408, 417
WTOR 408, 418
WTP 408, 417
REXXTDLI commands 404
RMODE 24, AMODE 31, running user modules in 192
ROLB call 112
comparison to ROLL and ROLS call 145
description 112, 146
format 112
parameters 112
restrictions 113
summary 439
usage 113
use in conversations 133
ROLB call function 536
roll back point 145
roll back to SETS/SETU call 114
ROLL call 114
comparison to ROLB and ROLS call 145
description 114, 146
format 114

ROLL call 114 *(continued)*
parameters 114
restrictions 114
summary 439
usage 114
use in conversations 133
ROLL call function 536
rollback call 112
ROLS call 114
comparison to ROLL and ROLB call 145
description 114
format 115
parameters 115
restrictions 116
Spool API functions 116
summary 439
usage 115
use in conversations 133
with LU 6.2 148
with TOKEN 148
without TOKEN 148
ROLS call function 536
ROLX call function 536
routine, error 14
RPRN= operand (DIV statement)
specifying 358
RPRN= operand (DIV statement), specifying 302
RPRN (return primary resource name) 302, 358

S

S (STATUS) statement 559
SAMETRM=YES 136
sample JCL 564
SAY exec 425
SCA (system control area) 209
description 209
device-dependent information 278
specifying 278, 327
use 242
SCA parameter (DFLD statement), specifying 375
screen formatting
3270 or SLU 2
erase all unprotected option 262
3290
logical units 263
specifying screen size 339
SCROLLI= operand (PD statement), specifying 388
SCS1 devices
CARD= operand (DIV statement) 347
DEV statement 297
DEV statement keywords 354
meaning of designation 181
SCS2 devices
DEV statement keywords 354
meaning of designation 181
specifying line width 348
secondary logical unit 180
security checks in program-to-program switching 126
SEG statement
EXIT= operand 321
GRAPHIC= operand 321
segment edit routine
specifying 321
use 192
segment format, output message 276
SELECT= operand (DPAGE statement)
specifying 366
SELECT= operand (DPAGE statement), specifying 310
selector pen, 3270
application program device-dependent information 274
effect on input fields 274
PEN= operand (DFLD statement) 383
specifying 347
specifying field detectability 377
sending messages
overview 14
to alternate destinations 124
to other application programs 126
to other IMS TM systems 128, 130
to several alternate destinations 124
using alternate PCBs 124
using the PURG call 124
sequence, indication for statements 564
service utility 179
set backout point call 116
set backout point unconditional call 116
SET command (REXX) 408, 414, 415
set options call 82
SET SUBFUNC command (REXX) 415
SET ZZ 415
SETO, DFSDDLT0
description 528
SETO call
and OTMA environment 85
description 82
format 82
parameters 82
restrictions 87
summary 439
usage 84, 515
SETO call function 536
SETS call 116
description 116, 149
format 116
parameters 116
restrictions 117
Spool API functions 117
summary 439
usage 117
SETS call function 536
SETU call 116
description 116
restrictions 117
Spool API functions 117
summary 439
shift in (SI) control character 214
shift in (SI) framing character 213
shift out (SO) control character 214
shift out (SO) framing character 213
signon security, RACF 47

skeleton programs 160
assembler language 153
C language 153
COBOL 155
Pascal 157
PL/I 160
SKIP call function 547
SLDI= operand (DEV statement), specifying 350
SLDI= operand (DFLD statement), specifying 384
SLDP= operand (DEV statement), specifying 350
SLDP= operand (DFLD statement), specifying 384
SLDx= operand (DEV statement), use 223
SLU 180
type 1, defining to operate with MFS 180
type 2, defining to operate with MFS 180
copy function 377
type 4, defining to operate with MFS 180
type 6.1, defining to operate with MFS 180
type P, defining to operate with MFS 180
SNAP call function 537
SO/SI control characters
blank suppress option 215
hex representation 215
pair verification 217
processing by MFS 215
use in mixed data field 214
SO/SI framing characters 213
SOR= operand (LPAGE statement), specifying 319
SOR= operand (MSG statement), specifying 318
SPA (scratchpad area) 134, 136
and program-to-program switches 137
contents 134
format 134
inserting 136
restrictions on using 135
saving information 136
SPACE statement (language utility) 395
Spool API
CHNG call, keywords 515
error codes
description 515
diagnosis, examples 516
functions 79
ISRT call 79
parsing errors
diagnosis, examples 516
error codes 515
status codes 515
print data set characteristics 515
SETO call, keywords 515
status codes 515
STORAGE command example 417
SRRBACK command (REXX)
description 408
format, usage 415
SRRCMIT command (REXX)
description 408
format, usage 415
STACK statement (language utility) 394, 508
staging library 179
STAK call function 547
standard application programs and MSC 142
standard character 391
START call function 547
STAT call function 537
status codes
blank 13
categories 443
checking 13
database calls 452
error routine 14
exception conditions 14
field in I/O PCB 46
message calls, table 452
retrieval call 14
system service calls, table 452
STATUS statement 559
storage
!token 416
STORAGE command 416
STORAGE command (REXX)
description 408
format, usage 416
stream mode
description 194
input example 199
repetitive DFLD generation 370
specifying 341
treatment of nulls 197
with ISC 202
SUB= operand (DEV statement)
specifying 351
use 201
substitution character 201
SUF= operand (DO statement), specifying 323, 368
summary of return codes
explanations 475
symbolic checkpoint call 92
Symbolic CHKP call
description 92
format 92
parameters 92
restrictions 93
summary 439
usage 93
SYNC call
description 118
format 118
parameters 118
restrictions 118
summary 439
usage 118
SYNC call function 537
synchronization call 118
synchronization point 145
syntax
control statements 311
errors 313
SYSIN input 564
SYSIN/SYSLIB record stacking and unstacking
description 311
STACK 394

SYSIN/SYSLIB record stacking and unstacking *(continued)*
UNSTACK 394
SYSIN2 input processing 564
SYSMSG= operand (DEV statement), specifying 348
SYSMSG= operand (PDB statement), specifying 386
SYSPRINT listing control
compilation statements 311
EJECT statement 396
PRINT statement 395
SPACE statement 395
TITLE statement 395
system contents directory 96
system control area 209
system definition
3270 master terminal format support 247
considerations, with MFS 224
system literals
date formats 325
description 210
other formats, CA parameter (MFLD statement) 327
time formats 325
system message field, specifying 348
system message format, IMS-provided 246
system message partition, specifying 386
system service calls
general description 13
ROLB call 133, 146
ROLL call 133, 146
ROLS call 133
summary 13

T

T (Comment) statement 549
tabbing
control characters 223
field tabs 194, 341
horizontal 222, 348
vertical 223, 349
TABLE statement, specifying 389
TABLEEND statement, specifying 391
test program 525
TITLE statement (language utility) 395
TM Batch, programming considerations 9
trailing blank compression 232
transaction code 236
translation, character
alpha character generation 391
for input messages
specifying 351
using XX'3F' 201
for output messages
device control characters 204
for DPM 204
nongraphic characters 204
GRAPHIC= operand (SEG statement) 322
SUB= operand (DEV statement) 201
transmission chains 202, 228
truncation
literal fields 325
truncation *(continued)*
of input messages 186
of output fields 204
TSO/E REXX 399
TYPE= operand
DIV statement
specifying 354
TYPE= operand (DEV statement), specifying 258, 339
TYPE= operand (DIV statement)
specifying 297
TYPE= operand (MSG statement), specifying 317

U

U (Comment) statement 549
underline, on fields 382
unprotecting the screen
specifying parameter on DLFD statement 377
UNPROTECT option 243
UNSTACK statement (language utility) 394, 508
uppercase, using Basic Edit 20
userid, field in I/O PCB 47
Utility Control Facility (UCF) 471
utility control statements 311

V

variable length output data stream 230
VERSID= operand (DEV statement), specifying 350
version identification
description 235
for DPM formats 250
for SLU P 184
specifying 350
vertical line, on fields 382
VIEWLOC= operand (PD statement), specifying 388
VIEWPORT= operand (PD statement), specifying 387
VT= operand (DEV statement)
specifying 349
use 223
VTAB= operand (DEV statement)
specifying 349
use 223

W

WIDTH= operand (DEV statement)
specifying 348
use 222
WINDOWOF= operand (PD statement), specifying 388
writing application programs, environmental
summary 7
WTL command (REXX)
description 408
format, usage 417
WTO command (REXX)
description 408
format, usage 417
WTO statement 563
WTOR command (REXX)
description 408

WTOR command (REXX) *(continued)*
format, usage 418
WTOR statement 563
WTP command (REXX)
description 408
format, usage 417

X

XRF (extended recovery facility), message format after takeover 221
XRST call 118
description 118
format 119
parameters 119
restrictions 121
summary 439
usage 120
XRST call function 537

Z

Z1 field 16
Z2 field 16
ZZ field
in input message 15
in output message 16

Readers' Comments — We'd Like to Hear from You

IMS/ESA
Application Programming:Transaction Manager
Version 6

Publication No. SC26-8729-05

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	☐	☐	☐	☐	☐

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? ☐ Yes ☐ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Name _____ Address _____

Company or Organization _____

Phone No. _____

Readers' Comments — We'd Like to Hear from You
SC26-8729-05

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International
Business Machines Corporation
Department BWE/H3
P.O. Box 49023
San Jose, CA
95161-9945

Fold and Tape | **Please do not staple** | Fold and Tape

Cut or Fold
Along Line

Program Number: 5655-158

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC26-8729-05