

IMS  
Version 13

*Application Programming APIs*  
*(November 5, 2018 edition)*





IMS  
Version 13

*Application Programming APIs*  
(November 5, 2018 edition)



**Note**

Before you use this information and the product it supports, read the information in “Notices” on page 691.

November 5, 2018 edition.

This edition applies to IMS Version 13 (program number 5635-A04), IMS Database Value Unit Edition, V13.1 (program number 5655-DSM), IMS Transaction Manager Value Unit Edition, V13.1 (program number 5655-TM2), and to all subsequent releases and modifications until otherwise indicated in new editions.

© **Copyright IBM Corporation 1974, 2018.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

# Contents

## About this information . . . . . vii

Prerequisite knowledge . . . . .	vii
IMS function names used in this information . . . . .	viii
How new and changed information is identified . . . . .	viii
How to read syntax diagrams . . . . .	viii
Accessibility features for IMS Version 13 . . . . .	x
How to send your comments . . . . .	x

## Chapter 1. DL/I calls reference . . . . . 1

Database management . . . . .	1
DL/I calls for database management . . . . .	1
DL/I calls for IMS DB system services . . . . .	35
Transaction management . . . . .	81
DL/I calls for transaction management . . . . .	81
DL/I calls for IMS TM system services . . . . .	123
EXEC DLI commands . . . . .	162
Summary of EXEC DLI commands . . . . .	163
ACCEPT command . . . . .	164
CHKP command . . . . .	165
DEQ command . . . . .	166
DLET command . . . . .	167
GN command . . . . .	169
GNP command . . . . .	174
GU command . . . . .	180
ISRT command . . . . .	185
LOAD command . . . . .	190
LOG command . . . . .	191
POS command . . . . .	192
QUERY command . . . . .	193
REFRESH command . . . . .	194
REPL command . . . . .	195
RETRIEVE command . . . . .	199
ROLB command . . . . .	200
ROLL command . . . . .	201
ROLS command . . . . .	202
SCHD command . . . . .	204
SETS command . . . . .	205
SETU command . . . . .	206
STAT command . . . . .	207
SYMCHKP command . . . . .	208
TERM command . . . . .	209
XRST command . . . . .	210
Command code reference . . . . .	212
A command code . . . . .	214
C command code . . . . .	214
D command code . . . . .	215
F command code . . . . .	217
G command code . . . . .	218
L command code . . . . .	218
N command code . . . . .	219
O command code . . . . .	220
P command code . . . . .	221
Q command code . . . . .	221
U command code . . . . .	224
V command code . . . . .	225

NULL command code . . . . .	226
DEDB command codes for DL/I . . . . .	226
Relationship between calls, AIBs, and PCBs . . . . .	234
DL/I test program (DFSDDLTO) reference . . . . .	235
Control statements . . . . .	235
ABEND statement . . . . .	236
CALL statement . . . . .	237
COMMENT statement . . . . .	258
COMPARE statement . . . . .	259
IGNORE statement . . . . .	265
OPTION statement . . . . .	265
PUNCH CTL statement . . . . .	267
STATUS statement . . . . .	269
WTO statement . . . . .	272
WTOR statement . . . . .	272
JCL requirements for the DL/I test program (DFSDDLTO) . . . . .	273
Execution of DFSDDLTO in IMS regions . . . . .	276
Explanation of DFSDDLTO return codes . . . . .	277
DFSDDLTO operations . . . . .	277

## Chapter 2. DRDA DDM command architecture reference. . . . . 281

Overview of the syntax for DDM terms supported by IMS . . . . .	281
DSSHDR syntax . . . . .	282
DDM commit and rollback processing . . . . .	282
DDM commands and command objects . . . . .	283
ACCRDB command (X'2001') . . . . .	283
ACCSEC command (X'106D') . . . . .	285
CLSQRY command (X'2005') . . . . .	286
CNTQRY command (X'2006') . . . . .	287
DEALLOCDB command (X'C801') . . . . .	289
DLIFUNC command object (X'CC05') . . . . .	290
DLIFUNCFLG command object (X'CC09') . . . . .	291
EXCSAT command (X'1041') . . . . .	292
EXCSQLIMM command (X'200A') . . . . .	294
EXCSQLSET command (X'2014') . . . . .	298
FLDENTRY command object (X'CC03') . . . . .	300
FLDENTRYREL command object (X'CC0C') . . . . .	301
IMSCALL command (X'C803') . . . . .	301
INAIB command object (X'CC01') . . . . .	302
MONITORRD command (X'1C00') . . . . .	303
OPNQRY command (X'200C') . . . . .	304
PRPSQLSTT command (X'200D') . . . . .	308
RLSE command (X'C802') . . . . .	310
RTRVFLD command object (X'CC04') . . . . .	312
RTRVFLDREL command object (X'CC0B') . . . . .	312
SECCHK command (X'106E') . . . . .	313
SEGMLIST command object (X'CC0A') . . . . .	314
SQLATTR command (X'2450') . . . . .	315
SQLCARD command (X'2408') . . . . .	316
SQLDARD command (X'2411') . . . . .	318
SQLDTA command (X'2412') . . . . .	322
SQLSTT command (X'2414') . . . . .	324

SSALIST command object (X'CC06')	325
DDM reply messages and reply objects	326
ABNUOWRM reply message (X'220D')	326
ACCRDBRM reply message (X'2201')	327
ACCSECRD reply object (X'14AC')	329
AGNPRMRM reply message (X'1232')	330
CMDVLTRM reply message (X'221D')	330
DEALLOCDBRM reply message (X'CA01')	331
ENDQRYRM reply message (X'220B')	332
ENDUOWRM reply message (X'220C')	333
EXCSATRD reply object (X'1443')	334
IMSCALLRM reply message (X'CA04')	336
OPNQFLRM reply message (X'2212')	337
OPNQRYRM reply message (X'2205')	338
QRYDSC reply object (X'241A')	339
QRYDTA reply object (X'241B')	340
QRYPOPRM reply message (X'220F')	341
RDBAFLRM reply message (X'221A')	342
RDBATHRM reply message (X'2203')	343
RDBNACRM reply message (X'2204')	344
RDBNFNRM reply message (X'2211')	345
RDBUPDRM reply message (X'2218')	346
RLSERM reply message (X'CA03')	347
RSCLMTRM reply message (X'1233')	348
SECCHKRM reply message (X'1219')	349
SQLERRRM reply message (X'2213')	350
DDM parameters used by IMS	350
AIBOALEN parameter (X'C904')	351
AIBRSNM1 parameter (X'C901')	351
AIBRSNM2 parameter (X'C902')	351
AIBSFUNC parameter (X'C903')	352
aibStream data structure	352
dbpcbStream data structure	353
iopcbStream data structure	354
OUTAIBDBPCB parameter (X'CC02')	355
OUTAIBIOPCB parameter (X'CC08')	356
RDBNAM parameter (X'2110')	357
SSA parameter (X'C906')	357
SSACOUNT parameter (X'C905')	358
UPDCNT parameter (X'C90A')	358

### Chapter 3. IMS Adapter for REXX reference . . . . . 359

IMS Adapter for REXX overview	360
Sample exit routine (DFSREXXU)	361
Addressing other environments	361
REXX transaction programs	361
REXXTDLI commands	363
REXXTDLI calls	364
REXXIMS extended commands	369
DLIINFO	369
IMSRXTRC	370
MAPDEF	371
MAPGET	373
MAPPUT	374
SET	375
SRRBACK and SRRCMIT	376
STORAGE	377
WTO, WTP, and WTL	378
WTOR	379
IMSQUERY extended functions	379

Sample execs using REXXTDLI	381
SAY exec: for expression evaluation	382
PCBINFO exec: display available PCBs in current PSB	383
PART execs: database access examples	384
DOCMD: IMS commands front end	387
IVPREXX sample application	391

### Chapter 4. Java programming reference . . . . . 395

IMS Universal drivers support for JDBC	395
javax.sql.Clob methods supported	395
java.sql.Connection methods supported	395
java.sql.DatabaseMetaData methods supported	397
javax.sql.DataSource methods supported	401
java.sql.Driver methods supported	401
java.sql.ParameterMetaData methods supported	402
java.sql.PreparedStatement methods supported	402
java.sql.Statement methods supported	403
java.sql.ResultSet methods supported	405
java.sql.ResultSetMetaData methods supported	410
IMS Universal drivers support for the Common Client Interface	411
javax.resource.cci.Connection methods supported	411
javax.resource.cci.ConnectionFactory methods supported	411
javax.resource.cci.ConnectionMetaData methods supported	412
javax.resource.cci.Interaction methods supported	412
javax.resource.cci.LocalTransaction methods supported	412
javax.resource.cci.ResultSetInfo methods supported	413
javax.resource.cci.ResourceAdapterMetaData methods supported	413
javax.resource.cci.RecordFactory methods supported	414
Java API documentation (Javadoc)	414

### Chapter 5. Message Format Service (MFS) reference . . . . . 419

MFS application program design	419
Relationships between MFS control blocks	419
Format library member selection	426
3270 or SLU 2 screen formatting	429
Device compatibility with previous versions of MFS	433
Enhancing system performance of MFS message and device formats	439
MFS definitions for intersystem communication	444
MFS message formats	446
Input message formats	446
Output message formats	448
MFS message formatting functions	482

### Chapter 6. OTMA Callable Interface API reference . . . . . 553

OTMA Callable Interface API calls	553
OTMA C/I hints and tips	553

otma_create API . . . . .	555		SQL statements . . . . .	635
otma_open API . . . . .	556		How SQL statements are invoked . . . . .	636
otma_openx API . . . . .	558		CLOSE . . . . .	638
otma_alloc API . . . . .	559		DECLARE CURSOR . . . . .	639
otma_send_receive API . . . . .	560		DECLARE STATEMENT . . . . .	640
otma_send_receivex API . . . . .	563		DELETE . . . . .	641
otma_send_async API . . . . .	564		DESCRIBE OUTPUT . . . . .	642
otma_receive_async API . . . . .	566		EXECUTE . . . . .	643
otma_free API . . . . .	568		FETCH . . . . .	645
otma_close API . . . . .	568		INCLUDE . . . . .	646
OTMA C/I sample programs . . . . .	569		INSERT . . . . .	647
Warranty and distribution for OTMA C/I			OPEN . . . . .	651
sample programs . . . . .	569		PREPARE . . . . .	653
OTMA C/I sample program for synchronous			SELECT . . . . .	655
processing . . . . .	569		UPDATE . . . . .	665
OTMA C/I sample program for asynchronous			WHENEVER . . . . .	668
processing . . . . .	581		SQL communication area (SQLIMSCA) . . . . .	669
			Description of SQLIMSCA fields . . . . .	669
			The included SQLIMSCA . . . . .	671
<b>Chapter 7. WSDL-to-PL/I segmentation</b>			SQL descriptor area (SQLIMSDA) . . . . .	671
<b>APIs for web service development . . . . .</b>	<b>595</b>		Description of SQLIMSDA fields . . . . .	672
Include file DFSPWSH . . . . .	595		The included SQLIMSDA . . . . .	674
DFSQGETS . . . . .	604			
DFSQSETS . . . . .	606		<b>Chapter 9. XML support reference . . . . .</b>	<b>677</b>
DFSXGETS . . . . .	609		SQL extensions for XML storage and retrieval . . . . .	677
DFSXSETS . . . . .	611		retrieveXML UDF . . . . .	677
Return codes from the DFSPWSIO APIs . . . . .	614		storeXML UDF . . . . .	679
			XQuery support in the IMS classic JDBC driver . . . . .	680
<b>Chapter 8. SQL programming</b>			XQuery function and operation extensions for	
<b>reference . . . . .</b>	<b>617</b>		IMS . . . . .	680
SQL concepts for IMS . . . . .	617		Standard XQuery features in IMS. . . . .	685
Structured query language . . . . .	617		XQuery functions and operators supported by	
IMS data structures for SQL . . . . .	618		IMS . . . . .	686
Language elements . . . . .	619			
Characters . . . . .	619		<b>Notices . . . . .</b>	<b>691</b>
Tokens . . . . .	619		Programming interface information . . . . .	693
Identifiers . . . . .	619		Trademarks . . . . .	693
Naming conventions . . . . .	620		Terms and conditions for product documentation . . . . .	694
Data types . . . . .	620		IBM Online Privacy Statement. . . . .	694
Assignment and comparison . . . . .	624			
Constants . . . . .	626		<b>Bibliography . . . . .</b>	<b>697</b>
Field names . . . . .	628			
References to variables . . . . .	629		<b>Index . . . . .</b>	<b>699</b>
Host structures in COBOL . . . . .	630			
Predicates . . . . .	630			
Search conditions . . . . .	634			





---

## About this information

These topics provide reference information for the IMS application programming interfaces (APIs). The topics also provide reference information for SQL programming for IMS, the IMS Adapter for REXX, the DL/I test program (DFSDDL0), and the IMS Message Format Service (MFS). Guidance information for writing IMS application programs is in *IMS Version 13 Application Programming*.

This information is available in IBM® Knowledge Center.

---

## Prerequisite knowledge

This book is an API (application programming interface) reference for IMS application programming in any of the following environments:

- IMS Database Manager (IMS DB), including IMS Database Control (DBCTL)
- IMS Transaction Manager (IMS TM)
- CICS® EXEC DLI
- WebSphere® Application Server for z/OS®
- WebSphere Application Server for distributed platforms
- Java™ dependent regions (JMP and JBP)
- Any environment for stand-alone Java application development

This book provides reference information for the IMS application programming interfaces (APIs), including DL/I, EXEC DLI, the IMS Universal drivers, and the Java class libraries for IMS. It also provides reference information for the IMS Adapter for REXX, the DL/I test program (DFSDDL0), and the IMS Message Format Service (MFS). Guidance information for writing IMS application programs is in *IMS Version 13 Application Programming*.

Before using this book, you should understand the concepts of application design described in *IMS Version 13 Application Programming*, which assumes that you understand basic z/OS and IMS concepts and the IMS environments. You should also know how to use assembler language, C language, COBOL, Pascal, or PL/I. CICS programs can be written in assembler language, C language, COBOL, PL/I, and C++.

To write Java applications, you must thoroughly understand the Java language and JDBC. This book assumes that you know Java and JDBC. It does not explain any Java or JDBC concepts.

To create the Java database metadata class, which is a required step in writing Java applications for IMS using the IMS Universal drivers or the Java class libraries, you must understand IMS databases. IMS database concepts are described in *IMS Version 13 Database Administration*.

To write applications that store or retrieve XML, you must understand XML and its related technologies, such as XML schemas.

You can learn more about z/OS by visiting the “z/OS basic skills” topics in IBM Knowledge Center.

You can gain an understanding of basic IMS concepts by reading *An Introduction to IMS*, an IBM Press publication.

IBM offers a wide variety of classroom and self-study courses to help you learn IMS. For a complete list of courses available, go to the IBM Skills Gateway and search for IMS.

---

## IMS function names used in this information

In this information, the term HALDB Online Reorganization refers to the integrated HALDB Online Reorganization function that is part of IMS Version 13, unless otherwise indicated.

---

## How new and changed information is identified

New and changed information in most IMS library PDF publications is denoted by a character (revision marker) in the left margin. The first edition (-00) of *Release Planning*, as well as the *Program Directory* and *Licensed Program Specifications*, do not include revision markers.

Revision markers follow these general conventions:

- Only technical changes are marked; style and grammatical changes are not marked.
- If part of an element, such as a paragraph, syntax diagram, list item, task step, or figure is changed, the entire element is marked with revision markers, even though only part of the element might have changed.
- If a topic is changed by more than 50%, the entire topic is marked with revision markers (so it might seem to be a new topic, even though it is not).

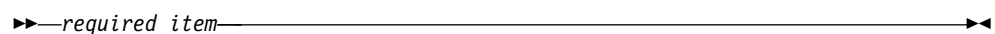
Revision markers do not necessarily indicate all the changes made to the information because deleted text and graphics cannot be marked with revision markers.

---

## How to read syntax diagrams

The following rules apply to the syntax diagrams that are used in this information:

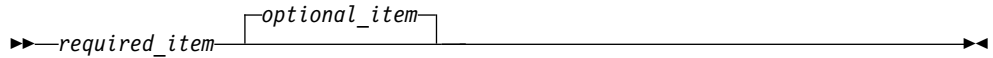
- Read the syntax diagrams from left to right, from top to bottom, following the path of the line. The following conventions are used:
  - The >>--- symbol indicates the beginning of a syntax diagram.
  - The ---> symbol indicates that the syntax diagram is continued on the next line.
  - The >--- symbol indicates that a syntax diagram is continued from the previous line.
  - The --->< symbol indicates the end of a syntax diagram.
- Required items appear on the horizontal line (the main path).



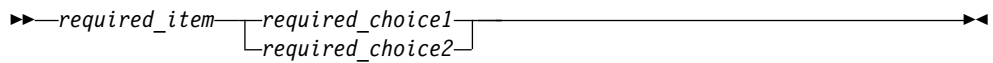
- Optional items appear below the main path.



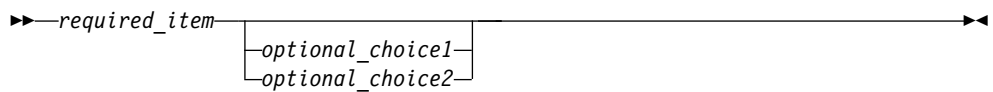
If an optional item appears above the main path, that item has no effect on the execution of the syntax element and is used only for readability.



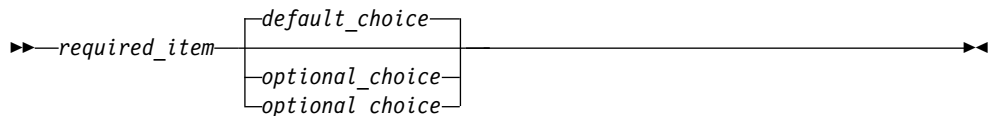
- If you can choose from two or more items, they appear vertically, in a stack. If you *must* choose one of the items, one item of the stack appears on the main path.



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



If one of the items is the default, it appears above the main path, and the remaining choices are shown below.



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

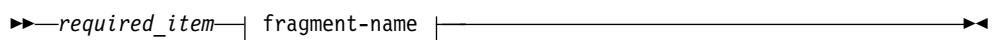


If the repeat arrow contains a comma, you must separate repeated items with a comma.

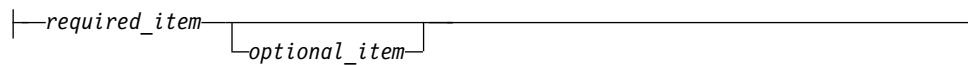


A repeat arrow above a stack indicates that you can repeat the items in the stack.

- Sometimes a diagram must be split into fragments. The syntax fragment is shown separately from the main syntax diagram, but the contents of the fragment should be read as if they are on the main path of the diagram.



**fragment-name:**



- In IMS, a b symbol indicates one blank position.
- Keywords, and their minimum abbreviations if applicable, appear in uppercase. They must be spelled exactly as shown. Variables appear in all lowercase italic letters (for example, *column-name*). They represent user-supplied names or values.
- Separate keywords and parameters by at least one space if no intervening punctuation is shown in the diagram.
- Enter punctuation marks, parentheses, arithmetic operators, and other symbols, exactly as shown in the diagram.
- Footnotes are shown by a number in parentheses, for example (1).

---

## Accessibility features for IMS Version 13

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use information technology products successfully.

### Accessibility features

The following list includes the major accessibility features in z/OS products, including IMS Version 13. These features support:

- Keyboard-only operation.
- Interfaces that are commonly used by screen readers and screen magnifiers.
- Customization of display attributes such as color, contrast, and font size.

### Keyboard navigation

You can access IMS Version 13 ISPF panel functions by using a keyboard or keyboard shortcut keys.

For information about navigating the IMS Version 13 ISPF panels using TSO/E or ISPF, refer to the *z/OS TSO/E Primer*, the *z/OS TSO/E User's Guide*, and the *z/OS ISPF User's Guide Volume 1*. These guides describe how to navigate each interface, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

### Related accessibility information

Online documentation for IMS Version 13 is available in IBM Knowledge Center.

### IBM and accessibility

See the *IBM Human Ability and Accessibility Center* at [www.ibm.com/able](http://www.ibm.com/able) for more information about the commitment that IBM has to accessibility.

---

## How to send your comments

Your feedback is important in helping us provide the most accurate and highest quality information. If you have any comments about this or any other IMS information, you can take one of the following actions:

- Click the **Contact Us** tab at the bottom of any IBM Knowledge Center topic.

- Send an email to [imspubs@us.ibm.com](mailto:imspubs@us.ibm.com). Be sure to include the book title and the publication number.

To help us respond quickly and accurately, please include as much information as you can about the content you are commenting on, where we can find it, and what your suggestions for improvement might be.



---

## Chapter 1. DL/I calls reference

These topics contain reference information for IMS DL/I calls.

---

### Database management

Use the following DL/I calls to access and administer IMS databases.

#### DL/I calls for database management

Use these DL/I calls with IMS DB to perform database management functions in your application program.

Each call description contains:

- A syntax diagram
- Definitions for parameters that are available to the call
- Details on how to use the call in your application program
- Restrictions on call usage, where applicable

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

Database management calls must use either *db pcb* or *aib* parameters. The syntax diagrams for these calls begin with the *function* parameter. The call, call interface (xxxTDLI), and *parmcount* (if it is required) are not included in the syntax diagrams.

**Related reading:** For specific information about coding your program in assembler language, C language, COBOL, Pascal, and PL/I, see the topic "Defining Application Program Elements" in *IMS Version 13 Application Programming*.

**Related reference:**

"DL/I calls for IMS TM system services" on page 123

"DL/I calls for transaction management" on page 81

"EXEC DLI commands" on page 162

#### Database management call summary

The following table shows the parameters that are valid for each database management call.

Optional parameters are enclosed in brackets ([ ]).

**Restriction:** 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.

**Related reading:** For more information on language-dependent application elements, see the topic "Defining Application Program Elements" in *IMS Version 13 Application Programming*.

Table 1. Summary of DB calls

Function Code	Meaning and Use	Options	Parameters	Valid for
CIMS		Initializes and terminates the ODBA interface in a z/OS application region.	aib	DB/DC, DBCTL, ODBA
CLSE	Close	Closes a GSAM database explicitly	function, gsam pcb or aib	DB/DC, DBCTL, DB batch, ODBA
DEQb	Dequeue	Releases segments reserved by Q command code	function, i/o pcb (full function only), or aib, i/o area (full function only)	DB batch, BMP, MPP, IFP, DBCTL, ODBA
DLET	Delete	Removes a segment and its dependents from the database	function, db pcb or aib, i/o area, [ssa]	DB/DC, DBCTL, DB batch, ODBA
FLDb	Field	Accesses a field within a segment	function, db pcb or aib, i/o area, rootssa	DB/DC, ODBA
GHNb	Get Hold Next	Retrieves subsequent message segments	function, db pcb or aib, i/o area, [ssa]	DB/DC, DBCTL, DB batch, ODBA
GHNP	Get Hold Next in Parent	Retrieves dependents sequentially	function, db pcb or aib, i/o area, [ssa]	DB/DC, DBCTL, DB batch, ODBA
GHUb	Get Hold Unique	Retrieves segments and establishes a starting position in the database	function, db pcb or aib, i/o area, [ssa]	DB/DC, DBCTL, DB batch, ODBA
GNbb	Get Next	Retrieves subsequent message segments	function, db pcb or aib, i/o area, [ssa or rsa]	DB/DC, DBCTL, DB batch, ODBA
GNPb	Get Next in Parent	Retrieves dependents sequentially	function, db pcb or aib, i/o area, [ssa]	DB/DC, DBCTL, DB batch, ODBA
GUbb	Get Unique	Retrieves segments and establishes a starting position in the database	function, db pcb or aib, i/o area, [ssa or rsa]	DB/DC, DBCTL, DB batch, ODBA
GUR	Get Unique Record	Retrieves a complete record from the IMS catalog in XML format	function, aib, i/o area, [ssa]	DB/DC, DBCTL, DB batch, ODBA
ISRT	Insert	Loads and adds one or more segments to the database	function, db pcb or aib, i/o area, [ssa or rsa]	DB/DC, DCCTL, DB batch, ODBA
OPEN	Open	Opens a GSAM database explicitly	function, gsam pcb or aib, [i/o area]	DB/DC, DBCTL, DB batch, ODBA
POSb	Position	Retrieves the location of a specific dependent or last-inserted sequential dependent segment	function, db pcb or aib, i/o area, [ssa]	DB/DC, DBCTL, DB batch, ODBA
REPL	Replace	Changes values of one or more fields in a segment	function, db pcb or aib, i/o area, [ssa]	DB/DC, DBCTL, DB batch, ODBA



Table 1. Summary of DB calls (continued)

Function Code	Meaning and Use	Options	Parameters	Valid for
RLSE	Release Locks	Releases all locks held for unmodified data	function, db pcb	DB/DC, DBCTL, DB batch, ODBA

### CIMS call

The CIMS call is used to initialize and terminate the ODBA interface in a z/OS application region.

### Format

▶▶—CIMS—*aib*—▶▶

Call Name	DB/DC	IMS DB	DCCTL	DB Batch	TM Batch
CIMS	X	X			

### Parameters

*aib*

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

These fields must be initialized in the AIB:

#### AIBID

Eye-catcher. 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

Character value.

#### AIBSFUNC

Subfunction code. This field must contain one of the 8-byte subfunction codes as follows:

#### INIT

AIBRSNM2. A 4-character ID of the ODBA startup table.

#### CONNECT

AIBRSA1. Address of the CONNECT parameter list.

The following table shows the CIMS CONNECT parameter list format.

Table 2. CIMS CONNECT parameter list format

Offset	Length	Field	Usage description
X'00'	X'04'	Input	Count of connect request table entries.
X'04'	X'04'	Input	Address of the connect request table.

#### TERM

AIBRSNM2. A 4-character ID of the ODBA startup table that represents the IMS connection that is to be terminated.

## TALL

Terminate all IMS connections.

## Usage

The CIMS call is used by an application program that is running in an application address space to establish or terminate the ODBA environment.

### INIT**bbbb**

The CIMS subfunction INIT must be issued by the application to establish the ODBA environment in the z/OS application address space.

Optionally, AIBRSNM2 can specify the 4-character ID of the ODBA Startup table member. This member is named DFSxxxx0 where xxxx is equal to the 4-character ID. If AIBRSNM2 is specified, ODBA tries to establish a connection to the IMS specified in the DFSxxxx0 member after the ODBA environment is initialized in the z/OS application address space.

### CONNECT**b**

Use the CIMS CONNECT call to establish multiple ODBA connections to IMS systems from the CSL Open Database Manager (ODBM).

A CIMS CONNECT call can be issued instead of, or in addition to, a CIMS INIT call. A CIMS CONNECT call will initialize ODBA if ODBA has not already been initialized. To complete initialization only, issue a CIMS CONNECT call with AIBRSA1 set to -1 (X'FFFFFFFF').

The connect request table contains one or more connect request entries in contiguous storage. Each entry contains the following fields:

- A 1- to 4-character alias name, left justified and padded on the right with blanks. The alias name is the value (*cccc*) taken from the startup properties table DFScccc0. This parameter is required.
- A 4-byte address of the connection properties table (DFSPRP) or 0.  
A value of 0 indicates that ODBA must load DFScccc0 to obtain the IMS connection properties. This member is constructed by specifying the DFSPRP macro in DFScccc0, and then assembling and linking the member. This member must be in the STEPLIB or JOBLIB of the ODBA application job.  
A nonzero value indicates that the caller is passing the address of the connection properties parameter table. The connection properties parameters are mapped by the DFSPRP macro.
- A 4-byte field to contain the connection request return code. The return code is one of the AIBRETRN codes.
- A 4-byte field to contain the connection request reason code. The reason code is one of the AIBREASN codes.
- A 4-byte field to contain the connection request error extension information code. The error extension contains additional diagnostic information specific to the return and reason codes.

The following table summarizes the CIMS CONNECT table entry format.

Table 3. CIMS CONNECT table entry format

Offset	Length	Field	Usage description
X'00'	X'04'	Input	1- to 4-character IMS alias name ( <i>cccc</i> ) from the startup properties table DFScccc0, where <i>cccc</i> is the alias name.

Table 3. CIMS CONNECT table entry format (continued)

Offset	Length	Field	Usage description
X'04'	X'04'	Input	0 or the address of an ODBA startup properties table.  A value of 0 indicates that ODBA must load a startup properties table named DFScccc0, where cccc is the supplied alias name.  An address indicates that the caller is supplying the startup properties table. The table is mapped by the DFSPRP macro.
X'08'	X'04'	Output	Connect request return code for this entry.
X'0C'	X'04'	Output	Connect request reason code for this entry.
X'10'	X'04'	Output	Connect request error extension code for this entry.

#### TERMbbbb

The CIMS subfunction TERM can be issued to terminate one IMS connection. AIBRSNM2 specifies the 4-character ID of the startup table member that represents the IMS connection to be terminated. On completion of the TERM subfunction, the ODBA environment remains intact in the z/OS application address space.

**Note:** If the application that issued CIMS INIT chooses to return to the operating system following completion of the CIMS TERM, the address space will terminate with a system abend A03. This can be avoided by issuing the CIMS TALL prior to returning to the operating system

#### TALLbbbb

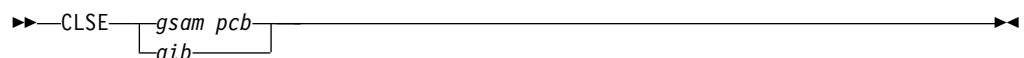
The CIMS subfunction TALL must be issued to terminate all IMS connections and terminate the ODBA environment in the application address space.

#### CLSE call

The close (CLSE) call is used to explicitly close a GSAM database.

For more information on GSAM, see the topic "Processing GSAM Databases" in *IMS Version 13 Application Programming*.

#### Format



	Call Name	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
For GSAM:	CLSE	X	X	X	X	X

## Parameters

*gsam pcb*

Specifies the GSAM PCB for the call. This parameter is an input and output parameter.

*aib*

Specifies the AIB for the call. This parameter is an input and output parameter. These fields must be initialized in the AIB:

**AIBID**

Eye catcher. This 8-byte field must contain DFSAIBbb.

**AIBLEN**

AIB length. 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 GSAM PCB.

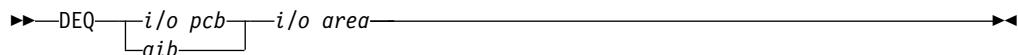
## Usage

For information on using CLSE, see the topic "Explicit Open and Close Calls to GSAM" in *IMS Version 13 Application Programming*.

## DEQ call

The Dequeue (DEQ) call is used to release a segment that is retrieved using the Q command code.

### Format (full function)



### Format (Fast Path DEDB)



	Call Name	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
For Full-Function and DEDB:	DEQ	X	X		X	

## Parameters

**DEDB pcb (Fast Path only)**

Specifies any DEDB PCB for the call.

**i/o pcb (full function only)**

Specifies the I/O PCB for the DEQ call. This is an input and output parameter.

*aib*

Specifies the AIB for the call. This is an input and output parameter. These fields must be initialized in the AIB:

**AIBID**

Eye catcher. 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 specified in the call list.

*i/o area (full function only)*

Specifies the 1-byte area containing a letter (A-J), which represents the lock class of the locks to be released. This is a mandatory input parameter.

## Usage

The DEQ call releases all segments that are retrieved using the Q command code, except:

- Segments modified by your program, until your program reaches a commit point
- Segments required to keep your position in the hierarchy, until your program moves to another database record
- A class of segments that has been locked using a different lock class

If your program only reads segments, it can release them by issuing a DEQ call. If your program does not issue a DEQ call, IMS releases the reserved segments when your program reaches a commit point. By releasing the segments with a DEQ call before your program reaches a commit point, you make them available to other programs more quickly.

For more information on the relationship between the DEQ call and the Q command code, see the topic "Reserving Segments for the Exclusive Use of Your Program" in *IMS Version 13 Application Programming*.

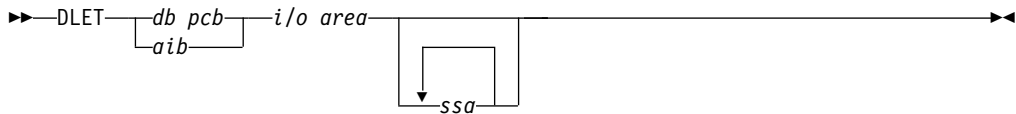
## Restrictions

In a CICS DL/I environment, calls made from one CICS (DBCTL) system are supported in a remote CICS DL/I environment, if the remote environment is also CICS (DBCTL).

## DLET call

The Delete (DLET) call is used to remove a segment and its dependents from the database.

## Format



	Call Name	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
For Full-Function:	DLET	X	X		X	
For DEDB:	DLET	X	X			
For MSDB:	DLET	X				

## Parameters

### *db pcb*

Specifies the DB PCB for the call. This parameter is an input and output parameter.

### *aib*

Specifies the AIB for the call. This parameter is an input and output parameter. These fields must be initialized in the AIB:

#### **AIBID**

Eye catcher. 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 DB PCB.

#### **AIBOALEN**

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

### *i/o area*

Specifies the I/O area in your program that communicates with IMS. This parameter is an input parameter. Before deleting a segment, you must first issue a Get Hold call to place the segment in the I/O area. You can then issue the DLET call to delete the segment and its dependents in the database.

### *ssa*

Specifies the SSA, if any, to be used in the call. This parameter is an input parameter. The SSA that you supply in the call point to data areas in your program where the SSAs have been defined for the call. You can use only one SSA in the parameter. This parameter is optional for the DLET call.

## Usage

The DLET call must be preceded by one of the three Get Hold calls. When you issue the DLET call, IMS deletes the held segment, along with all its physical dependents from the database, *regardless of whether your program is sensitive to all of these segments*. IMS rejects the DLET call if the preceding call for the PCB was not a Get

Hold, REPL, or DLET call. If the DLET call is successful, the previously retrieved segment and all of its dependents are removed from the database and cannot be retrieved again.

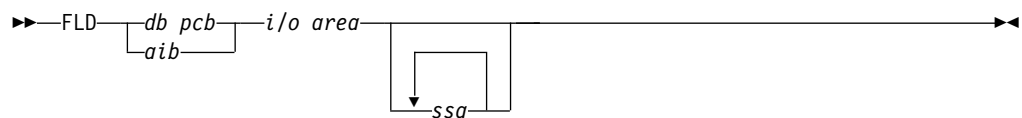
If the Get Hold call that precedes the DLET call is a path call, and you do not want to delete all the retrieved segments, you must indicate to IMS which of the retrieved segments (and its dependents, if any) you want deleted; to do this, specify an unqualified SSA for that segment. Deleting a segment this way automatically deletes all dependents of the segment. Only one SSA is allowed in the DLET call, and this is the only time a SSA is applicable in a DLET call.

No command codes apply to the DLET call. If you use a command code in a DLET call, IMS disregards the command code.

### FLD call

The Field (FLD) call is used to access a field within a segment for MSDBs or DEDBs.

#### Format



	Call Name	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
For MSDB:	FLD	X				
For DEDB:	FLD	X	X			

### Parameters

#### *db pcb*

Specifies the DB PCB for the call. This parameter is an input and output parameter.

#### *aib*

Specifies the AIB for the call. This parameter is an input and output parameter. These fields must be initialized in the AIB:

#### **AIBID**

Eye catcher. 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 DB PCB.

#### **AIBOALEN**

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

#### *i/o area*

Specifies your program's I/O area, which contains the field search argument (FSA) for this call. This parameter is an input parameter.

*ssa*

Specifies the SSA, if any, that you want to use in this call. You can use up to 15 SSAs in this input parameter. The SSA that you supply will point to those data areas that you have defined for the call. This parameter is optional for the FLD call.

## Usage

Use the FLD call to access and change the contents of a field within a segment.

The FLD call does two things for you: it compares the value of a field to the value you supply (FLD/VERIFY), and it changes the value of the field in the way that you specify (FLD/CHANGE).

All DL/I command codes are available to DEDBs, using the FLD call. The FLD call formats for DEDBs are the same as for other DL/I calls. So, if your MSDBs have been converted to DEDBs, you do not need to change application programs that use the FLD call.

You can also use the FLD call in application programs for DEDBs, instead of the combination of GHU, REPL, and DL/I calls.

## FSAs

The field search argument (FSA) is equivalent to the I/O area that is used by other DL/I database calls. For a FLD call, data is not moved into the I/O area; rather, the FSAs are moved into the I/O area.

Multiple FSAs are allowed on one FLD call. This is specified in the FSA's connector field. Each FSA can operate on either the same or different fields within the target segment.

The FSA that you reference in a FLD call contains five fields. The rules for coding these fields are as follows:

### Field name

This field must be 8 bytes long. If the field name you are using is less than 8 bytes, the name must be left-justified and padded on the right with blanks.

### FSA status code

This field is 1 byte. After a FLD call, IMS returns one of these status codes to this area:

- b** Successful
- A** Invalid operation
- B** Operand length invalid
- C** Invalid call—program tried to change key field
- D** Verify check was unsuccessful
- E** Packed decimal or hexadecimal field is invalid
- F** Program tried to change an unowned segment
- G** Arithmetic overflow
- H** Field not found in segment



### Op code

This 1-byte field contains one of these operators for a change operation:

- + To add the operand to the field value
- To subtract the operand from the field value
- = To set the field value to the value of the operand

For a verify operation, this field must contain one of the following:

- E Verify that the field value and the operand are equal.
- G Verify that the field value is greater than the operand.
- H Verify that the field value is greater than or equal to the operand.
- L Verify that the field value is less than the operand.
- M Verify that the field value is less than or equal to the operand.
- N Verify that the field value is not equal to the operand.

### Operand

This variable length field contains the value that you want to test the field value against. The data in this field must be the same type as the data in the segment field. (You define this in the DBD.) If the data is hexadecimal, the value in the operand is twice as long as the field in the database. If the data is packed decimal, the operand does not contain leading zeros, so the operand length might be shorter than the actual field. For other types of data, the lengths must be equal.

### Connector

This 1-byte field must contain a blank if this is the last or only FSA, or an asterisk (\*) if another FSA follows this one.

The format of SSA in FLD calls is the same as the format of SSA in DL/I calls. If no SSA exists, the first segment in the MSDB or DEDB is retrieved.

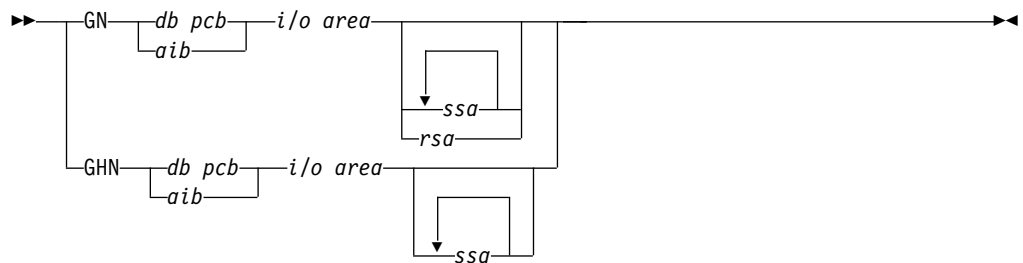
### Related concepts:

- ➡ Commit-point processing in MSDBs and DEDBs (Application Programming)
- ➡ Updating segments: REPL, DLET, ISRT, and FLD (Application Programming)

### GN/GHN call

The Get Next (GN) call is used to retrieve segments sequentially from the database. The Get Hold Next (GHN) is the hold form for a GN call.

### Format



	Call Name	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
For Full-Function:	GN/GHN	X	X		X	
For GSAM:	GN	X	X	X	X	X
For DEDB:	GN	X	X	X		
For MSDB:	GN	X				

## Parameters

### *db pcb*

Specifies the DB PCB for the call. This parameter is an input and output parameter.

### *aib*

Specifies the AIB for the call. This parameter is an input and output parameter. These fields must be initialized in the AIB:

#### **AIBID**

Eye catcher. 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 DB PCB.

#### **AIBOALEN**

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

### *i/o area*

Specifies the I/O area. This parameter is an output parameter. When you issue one of the Get calls successfully, IMS returns the requested segment to this area. If your program issues any path calls, the I/O area must be long enough to hold the longest path of concatenated segments following a path call. This area always contains left-justified segment data. The I/O area points to the first byte of this area.

When you use the GN call with GSAM, the area named by the *i/o area* parameter contains the record you are retrieving.

### *ssa*

Specifies the SSA, if any, to be used in the call. This parameter is an input parameter. The SSA that you supply in the call point to data areas in your program where the SSA have been defined for the call. You can use up to 15 SSAs in the parameter. This parameter is optional for the GN call.

### *rsa*

Specifies the area in your program where the RSA for the record should be returned. This output parameter is used for GSAM only and is optional. See the topic "GSAM Data Areas" in *IMS Version 13 Application Programming* for more information on RSAs.

## Usage: Get Next (GN)

A Get Next (GN) call is a request for a segment, as described by the SSA you supply, that is linked to the call that was issued prior to the GN call. IMS starts its search at the current position.

When you use the GN call:

- Processing moves forward from current position (unless the call includes the F command code).
- IMS uses the current position (that was set by the previous call) as the search starting point.
- The segment retrieved is determined by a combination of the next sequential position in the hierarchy and the SSA included in the call.
- Be careful when you use GN, because it is possible to use SSAs that force IMS to search to the end of the database without retrieving a segment. This is particularly true with the “not equal” or “greater than” relational operators.

A GN call retrieves the next segment in the hierarchy that satisfies the SSA that you supplied. Because the segment retrieved by a GN call depends on the current position in the hierarchy, GN is often issued after a GU call. If no position has been established in the hierarchy, GN retrieves the first segment in the database. A GN call retrieves a segment or path of segments by moving forward from the current position in the database. As processing continues, IMS looks for segments at each level to satisfy the call.

For example, sequential retrieval in a hierarchy is always top to bottom and left to right. For example, if you repeatedly issue unqualified GN calls against the hierarchy in the following figure, IMS returns the segment occurrences in the database record in this order:

1. A1 (the root segment)
2. B1 and its dependents (C1,D1,F1,D2,D3,E1,E2, and G1)
3. H1 and its dependents (I1,I2,J1, and K1).

If you issue an unqualified GN again after IMS has returned K1, IMS returns the root segment occurrence whose key follows segment A1 in the database.

A GN call that is qualified with the segment type can retrieve all the occurrences of a particular segment type in the database.

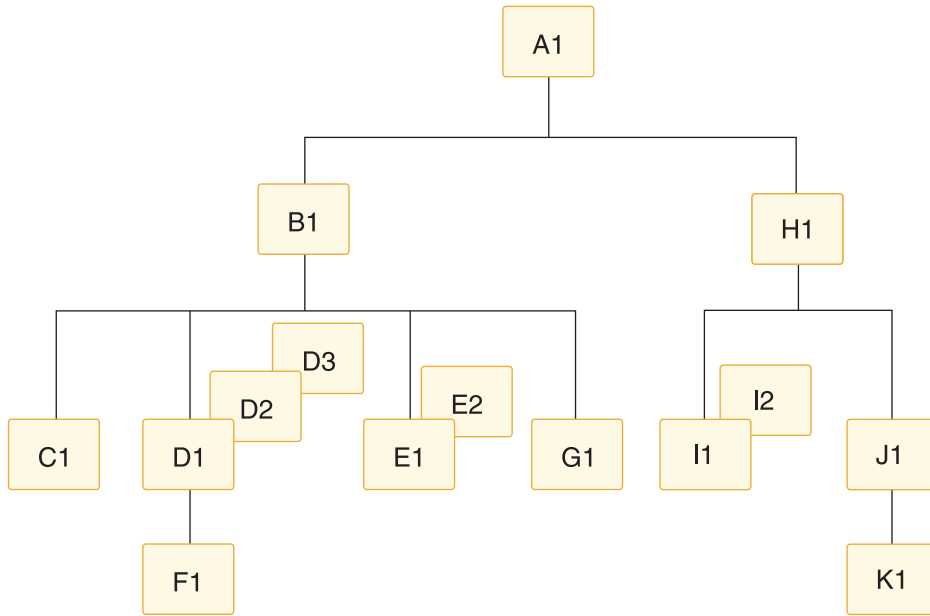


Figure 1. Hierarchic sequence

For example, if you issue a GN call with qualified SSAs for segments A1 and B1, and an unqualified SSA for segment type D, IMS returns segment D1 the first time you issue the call, segment D2 the second time you issue the call, and segment D3 the third time you issue the call. If you issue the call a fourth time, IMS returns a status code of GE, which means that IMS could not find the segment you requested.

You can use unqualified GN calls to retrieve all of the occurrences of a segment in a hierarchy, in their hierarchic sequence, starting at the current position. Each unqualified GN call retrieves the next sequential segment forward from the current position. For example, to answer the processing request:

Print out the entire medical database.

You would issue an unqualified GN call repeatedly until IMS returned a GB status code, indicating that it had reached the end of the database without being able to satisfy your call. If you issued the GN again after the GB status code, IMS would return the first segment occurrence in the database.

Like GU, a GN call can have as many SSAs as the hierarchy has levels. Using fully qualified SSAs with GN calls clearly identifies the hierarchic path and the segment you want, thus making it useful in documenting the call.

A GN call with an unqualified SSA retrieves the next occurrence of that segment type by going forward from the current position.

GN with a qualified SSA retrieves the next occurrence of the specified segment type that satisfies the SSAs.

When you specify a GN that has multiple SSAs, the presence or absence of unqualified SSAs in the call has no effect on the operation unless you use command codes on the unqualified SSA. IMS uses only qualified SSAs plus the last SSA to determine the path and retrieve the segment. Unspecified or unqualified

SSAs for higher-level segments in the hierarchy mean that any high-level segment that is the parent of the correct lower-level, specified or qualified segment will satisfy the call.

A GN call with a SSA that is qualified on the key of the root can produce different results from a GU with the same SSA, depending on the position in the database and the sequence of keys in the database. If the current position in the database is beyond a segment that would satisfy the SSA, the segment is not retrieved by the GN. GN returns the GE status code if both of these conditions are met:

- The value of the key in the SSA has an upper limit that is set, for example, to less-than-or-equal-to the value.
- A segment with a key greater than the value in the SSA is found in a sequential search before the specified segment is found.

GN returns the GE status code, even though the specified segment exists and would be retrieved by a GU call.

### **Usage: Get Hold Next (GHN)**

Before your program can delete or replace a segment, it must retrieve the segment and indicate to IMS that it is going to change the segment in some way. The program does this by issuing a Get call with a “hold” before deleting or replacing the segment. When the program has successfully retrieved the segment with a Get Hold call, it can delete the segment or change one or more fields (except the key field) in the segment.

The only difference between Get calls with a hold and Get calls without a hold is that the hold calls can be followed by REPL or DLET.

The hold status on the retrieved segment is canceled and must be reestablished before you reissue the DLET or REPL call. After issuing a Get Hold call, you can issue more than one REPL or DLET call to the segment if you do not issue intervening calls to the same PCB.

If you find out that you do not need to update it after issuing a Get Hold call, you can continue with other processing without releasing the segment. The segment is freed as soon as the current position changes—when you issue another call to the same PCB that you used for the Get Hold call. In other words, a Get Hold call must precede a REPL or DLET call. However, issuing a Get Hold call does not require you to replace or delete the segment.

### **Usage: HDAM, PHDAM, or DEDB database with GN**

For database organizations other than HDAM, PHDAM, and DEDB, processing the database sequentially using GN calls returns the root segments in ascending key sequence. However, the order of the root segments for a HDAM, PHDAM, or DEDB database depends on the randomizing routine that is specified for that database. Unless a sequential randomizing routine was specified, the order of the root segments in the database is not in ascending key sequence.

For a hierarchic direct access method (HDAM, PHDAM) or a DEDB database, a series of unqualified GN calls or GN calls that are qualified only on the root segment:

1. Returns all the roots from one anchor point
2. Moves to the next anchor point
3. Returns the roots from the anchor point

Unless a sequential randomizing routine was specified, the roots on successive anchor points are not in ascending key sequence. One situation to consider for HDAM, PHDAM, and DEDB organizations is when a GN call is qualified on the key field of the root segment with an equal-to operator or an equal-to-or-greater-than operator. If IMS has an existing position in the database, it checks to ensure that the requested key is equal to or greater than the key of the current root. If it is not, a GE status code is returned. If it is equal to or greater than the current key and is not satisfied using the current position, IMS calls the randomizing routine to determine the anchor point for that key. IMS tries to satisfy the call starting with the first root of the selected anchor.

## Restrictions

You can use GN to retrieve the next record of a GSAM database, but GHN is not valid for GSAM.

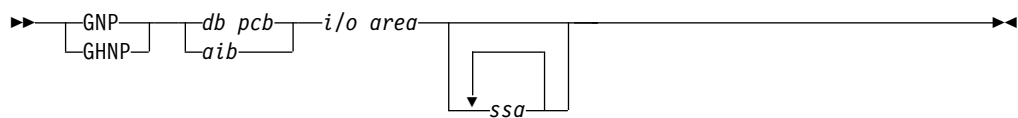
### Related reference:

“GNP/GHNP call”

## GNP/GHNP call

The Get Next in Parent (GNP) call is used to retrieve dependents sequentially. The Get Hold Next in Parent (GHNP) call is the hold form for the GNP call.

## Format



	Call Name	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
For Full-Function:	GNP/GHNP	X	X		X	
For DEDB:	GNP/GHNP	X	X	X		
For MSDB:	GNP/GHNP	X				

## Parameters

### *db pcb*

Specifies the DB PCB for the call. This parameter is an input and output parameter.

### *aib*

Specifies the AIB for the call. This parameter is an input and output parameter. These fields must be initialized in the AIB:

#### **AIBID**

Eye catcher. 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 DB PCB.

## **AIBOALEN**

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

### *i/o area*

Specifies the I/O area. This parameter is an output parameter. When you issue the Get call successfully, IMS returns the requested segment to this area. If your program issues any path calls, the I/O area must be long enough to hold the longest path of concatenated segments following a path call. The segment data that this area contains is always left-justified. The I/O area points to the first byte of this area.

### *ssa*

Specifies the SSA, if any, to be used in the call. This parameter is an input parameter. The SSA you supply in the call point to data areas in your program in which you have defined the SSAs for the call. You can use up to 15 SSAs for this parameter. This parameter is optional for the GNP call.

## **Usage: Get Next in Parent (GNP)**

A GNP call retrieves segments sequentially. The difference between a GN and a GNP is that GNP limits the segments that can satisfy the call to the dependent segments of the established parent.

An unqualified GNP retrieves the first dependent segment occurrence under the current parent. If your current position is already on a dependent of the current parent, an unqualified GNP retrieves the next segment occurrence.

If you are moving forward in the database, even if you are not retrieving every segment in the database, you can use GNP to restrict the returned segments to only those children of a specific segment.

### *Linking with previous DL/I calls*

A GNP call is linked to the previous DL/I calls that were issued by your program in two ways:

- Current position: The search for the requested segment starts at the current position established by the preceding GU, GN, or GNP call.
- Parentage: The search for the requested segment is limited to the dependents of the lowest-level segment most recently accessed by a GU or GN call. Parentage determines the end of the search and is in effect only following a successful GU or GN call.

### *Processing with parentage*

You can set parentage in two ways:

- By issuing a successful GU or GN call. When you issue a successful GU or GN call, IMS sets parentage at the lowest-level segment returned by the call. Issuing another GU or GN call (but against a different PCB) does not affect the parentage that you set using the first PCB in the previous call. An unsuccessful GU or GN call cancels parentage.
- By using the P command code with a GU, GN, or GNP call, you can set parentage at any level.

### *How DL/I calls affect parentage*

A GNP call does not affect parentage unless it includes the P command code.

Unless you are using a secondary index, REPL does not affect parentage. If you are using a secondary index, and you replace the indexed segment, parentage is lost.

A DLET call does not affect parentage unless you delete the established parent. If you do delete the established parent, you must reset parentage before issuing a GNP call.

ISRT affects parentage only when you insert a segment that is not a dependent of the established parent. In this case, ISRT cancels parentage. If the segment you are inserting is a dependent at some level of the established parent, parentage is unaffected. For example, in the topic "Position after ISRT" in *IMS Version 13 Application Programming*, assume segment B11 is the established parent. Neither of these two ISRT calls would affect parentage:

```
ISRT  Abbbbbbb(AKEYbbb=A1)
      Bbbbbbbb(BKEYbbb=bB11)
      Cbbbbbbb
```

```
ISRT  Abbbbbbbb(AKEYbbb=bA1)
      Bbbbbbbbb(BKEYbbb=bB11)
      Cbbbbbbb(CKEYbbb=bC111)
      Dbbbbbbb
```

The following ISRT call would cancel parentage, because the F segment is not a direct dependent of B, the established parent:

```
ISRT  Abbbbbbbb(AKEYbbb=bA1)
      Fbbbbbbb
```

You can include one or more SSAs in a GNP call. The SSA can be qualified or unqualified. Without SSAs, a GNP call retrieves the next sequential dependent of the established parent. The advantage of using SSAs with GNP is that they allow you to point IMS to a specific dependent or dependent type of the established parent.

A GNP with an unqualified SSA sequentially retrieves the dependent segment occurrences of the segment type you have specified under the established parent.

A GNP with a qualified SSA describes to IMS the segment you want retrieved or the segment that is to become part of the hierarchic path to the segment you want retrieved. A qualified GNP describes a unique segment only if it is qualified on a unique key field and not a data field or a non-unique key field.

A GNP with multiple SSAs defines the hierarchic path to the segment you want. If you specify SSAs for segments at levels above the established parent level, those SSAs must be satisfied by the current position at that level. If they cannot be satisfied using the current position, a GE status code is returned and the existing position remains unchanged. The last SSA must be for a segment that is below the established parent level. If it is not, a GP status code is returned. Multiple unqualified SSAs establish the first occurrence of the specified segment type as part of the path you want. If some SSAs between the parent and the requested segment in a GNP call are missing, they are generated internally as unqualified SSAs. This means that IMS includes the first occurrence of the segment from the missing SSAs as part of the hierarchic path to the segment you have requested.



## Usage: Get Hold Next in Parent (GHNP)

Retrieval for the GHNP call is the same as for the GHN call.

### Related concepts:

➡ How secondary indexing affects your program (Application Programming)

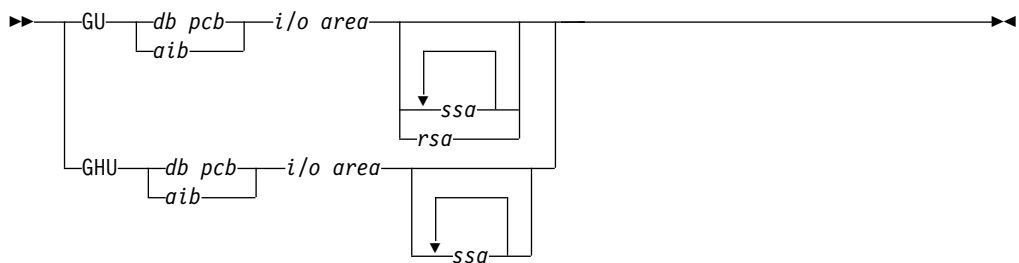
### Related reference:

“GN/GHN call” on page 11

## GU/GHU call

The Get Unique (GU) call is used to directly retrieve segments and to establish a starting position in the database for sequential processing. The Get Hold Unique (GHU) is the hold form for a GU call.

### Format



	Call Name	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
For Full-Function:	GU/GHU	X	X		X	
For GSAM:	GU	X	X	X	X	X
For DEDB:	GU	X	X	X		
For MSDB:	GU	X				

## Parameters

### *db pcb*

Specifies the DB PCB for the call. This parameter is an input and output parameter.

### *aib*

Specifies the AIB for the call. This parameter is an input and output parameter. These fields must be initialized in the AIB:

#### **AIBID**

Eye catcher. 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 DB PCB.

#### **AIBOALEN**

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

### *i/o area*

Specifies the I/O area. This parameter is an output parameter. When you issue one of the Get calls successfully, IMS returns the requested segment to this area. If your program issues any path calls, the I/O area must be long enough to hold the longest path of concatenated segments following a path call. The segment data that this area contains is always left-justified. The I/O area points to the first byte of this area.

When you use the GU call with GSAM, the area named by the *i/o area* parameter contains the record you are retrieving.

### *ssa*

Specifies the SSA, if any, to be used in the call. This parameter is an input parameter. The SSA you supply in the call point to data areas in your program where you have defined the SSAs for the call. You can use up to 15 SSAs for the parameter. This parameter is optional for the GU call.

### *rsa*

Specifies the area in your program that contains the record search argument. This required input parameter is only used for GSAM. See the topic "GSAM Data Areas" in *IMS Version 13 Application Programming* for more information on RSAs.

## **Usage: Get Unique (GU)**

GU is a request for a segment, as described by the SSAs you supply. You use it when you want a specific segment. You can also use it to establish your position in the database.

The GU call is the only call that can establish position backward in the database. (The GN and GNP calls, when used with the F command code, can back up in the database, but with limitations. Unlike GN and GNP, a GU call does not move forward in the database automatically.)

If you issue the same GU call repeatedly, IMS retrieves the same segment each time you issue the call. If you want to retrieve only particular segments, use fully qualified GUs for these segments instead of GNs. If you want to retrieve a specific segment occurrence or obtain a specific position within the database, use GU.

If you want to retrieve a specific segment or to set your position in the database to a specific place, you generally use qualified GU calls. A GU call can have the same number of SSAs as the hierarchy has levels, as defined by the DB PCB. If the segment you want is on the fourth level of the hierarchy, you can use four SSAs to retrieve the segment. (No reason would ever exist to use more SSAs than levels in the hierarchy. If your hierarchy has only three levels, you would never need to locate a segment lower than the third level.) The following is additional information for using the GU call with SSAs:

- A GU call with an unqualified SSA at the root level attempts to satisfy the call by starting at the beginning of the database. If the SSA at the root level is the only SSA, IMS retrieves the first segment in the database.
- A GU call with a qualified SSA can retrieve the segment described in the SSA, regardless of that segment's location relative to current position.
- When you issue a GU that mixes qualified and unqualified SSAs at each level, IMS retrieves the first occurrence of the segment type that satisfies the call.

- If you leave out an SSA for one of the levels in a GU call that has multiple SSAs, IMS assumes an SSA for that level. The SSA that IMS assumes depends on current position:
  - If IMS has a position established at the missing level, the SSA that IMS uses is derived from that position, as reflected in the DB PCB.
  - If IMS does not have a position established at the missing level, IMS assumes an unqualified SSA for that level.
  - If IMS moves forward from a position established at a higher level, IMS assumes an unqualified SSA for that level.
  - If the SSA for the root level is missing, and IMS has position established on a root, IMS does not move from that root when trying to satisfy the call.

### Usage: Get Hold Unique (GHU)

Before your program can delete or replace a segment, it must retrieve the segment and indicate to IMS that it is going to change the segment in some way. The program does this by issuing a Get call with a “hold” before deleting or replacing the segment. Once the program has successfully retrieved the segment with a Get Hold call, it can delete the segment or change one or more fields (except the key field) in the segment.

The only difference between Get calls with a hold and without a hold is that the hold calls can be followed by a REPL or DLET call.

The hold status on the retrieved segment is canceled and must be reestablished before you reissue the DLET or REPL call. After issuing a Get Hold call, you can issue more than one REPL or DLET call to the segment if you do not issue intervening calls to the same PCB.

If you find out that you do not need to update it after issuing a Get Hold call, you can continue with other processing without releasing the segment. The segment is freed as soon as the current position changes—when you issue another call to the same PCB you used for the Get Hold call. In other words, a Get Hold call must precede a REPL or DLET call. However, issuing a Get Hold call does not require you to replace or delete the segment.

### Restrictions

You can use GU to retrieve the record with the RSA you provide with a GSAM database, but GHU is not valid for GSAM.

#### Related concepts:

“F command code” on page 217

### GUR call

The Get Unique Record (GUR) call is used to retrieve entire records from the IMS catalog database. The records are returned as XML instance documents.

### Format

►► GUR—*aib*—*i/o area*—*header ssa* —————►  
   └resource ssa┘

## Parameters

*aib*

Specifies the AIB for the call. This parameter is an input and output parameter. These fields must be initialized in the AIB:

### **AIBID**

Eye catcher. 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-aligned field must contain the name of a DB PCB.

### **AIBOALEN**

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

### **AIBRTKN**

AIB return token. This 8-byte field contains a token value when a GUR call returns more data than can fit in the I/O area. You can retrieve the rest of the data by setting this field to the returned value when you issue a subsequent GUR call. IMS returns the next block of data, and you can continue to issue sequential calls by continuing to set the AIBRTKN field until all of the data is retrieved.

When the AIBRTKN field is non-zero, all SSAs are ignored for the call.

If an invalid or unrecognized token value is specified, the call fails.

### **AIBOAUSE**

Specifies the total length of the XML instance document returned by the GUR call. This value is set by IMS after a successful GUR call. The value is given in bytes.

When the value of the AIBOAUSE field is less than the value of the AIBOALEN field, the application program can retrieve the entire XML document from the I/O area.

When the value of the AIBOAUSE field is greater than the value of the AIBOALEN field, the application program must make additional GUR calls with the AIBRTKN value set to the returned token value of the first call to retrieve the entire XML instance document.

The size of the last GUR call in a linked series might not match the size of the remaining data. For example, a GUR call that returns 9000 bytes of data for a request with AIBOALEN=4096 requires three linked GUR calls to retrieve all of the data. The third call returns only 808 bytes of data in the I/O area.

The AIBOAUSE value is returned for all GUR calls in a linked series, and always reflects the total size of the XML instance document.

### **AIBRETRN**

Return code.

### **AIBREASN**

Reason code.

*i/o area*

Specifies the I/O area where IMS places the XML instance document returned

by the call. This parameter is an output parameter. When you issue the calls successfully, IMS returns the requested record to this area. The XML instance document that this area contains is always left-aligned. The I/O area parameter points to the first byte of this area.

*header ssa*

Specifies the name of the HEADER segment to search for. This parameter is required.

*resource ssa*

Specifies the name of the DBD or PSB segment to search for. This parameter is optional and is only valid if a HEADER SSA is specified.

IMS uses the timestamp for the active resource, either a DBD or PSB, in the ACBLIB to find the corresponding resource in the catalog.

## Usage

The Get Unique Record (GUR) call is a request for a complete record from the IMS catalog.

Catalog records are returned as XML instance documents, and can be larger than the available I/O area. IMS stores a complete XML instance document for a successful GUR call in an internal retrieval cache and can return it to an application program in pieces that are each the size of the available I/O area. Each subsequent GUR call to retrieve another piece of the XML instance document must use the token value set by IMS in the AIBRTKN field after the original call.

The XML schemas for the documents returned as responses to this call are included in the IMS.ADFSSMPL data set:

- DFS3XDBD.xsd (for DBD records)
- DFS3XPSB.xsd (for PSB records)

You can use z/OS XML System Services to parse the response document. The z/OS XML parser is started as a callable service. The services stubs are shipped in CSSLIB.

A GUR call SSA must start with the HEADER segment.

A GUR call that is issued with an unqualified SSA attempts to satisfy the request by starting at the beginning of the target database. If the SSA at the root level is the only SSA, IMS retrieves the first segment in the database. A GUR call with a qualified SSA can retrieve the segment described in the SSA, regardless of the location of the segment relative to the current position of the cursor. The two levels of SSA qualification that can be used with a GUR call correspond to the levels of the DBD or PSB stored in the catalog.

The IMS catalog has a structure that uses a header segment as the root for each record. Each header segment instance has either a PSB or DBD child segment instance. This structure is important to understand because an unqualified GUR call (such as the following example) might not return the expected record.

```
GUR HEADER
  PSB
```

This call locates the first record, which is always a DBD record because DBD precedes PSB in alphanumeric order. Because the first record does not contain a

PSB segment instance, the call does not return the first PSB record as expected. You must qualify the wanted record type at the level of the segment header:

```
GUR HEADER (TYPE = PSB      )
           PSB
```

A GUR call that is issued without a qualification at the PSB or DBD level retrieves the record for the member that is currently active in the ACB library. If no catalog record is found that corresponds to the active member, the call fails with return code X'108' and reason code X'338'. This error occurs even if there are one or more catalog records for inactive members of the ACB library or records for members that do not currently exist in the ACB library. To retrieve those catalog records, you must determine the ACB generation timestamp for the member corresponding to the wanted catalog record and include it as a PSB or DBD-level qualification.

For example, the following GUR call fails if there is no active ACB library member for BMP255:

```
GUR  HEADER (RHDRSEQ ==PSB      BMP255  )
```

To retrieve the record for an inactive or removed ACB library member, add an SSA qualification for the correct ACB generation timestamp:

```
GUR  HEADER (RHDRSEQ ==PSB      BMP255  )
           PSB      (TSVERS  ==XXXXXXXXXXXX)
```

**Note:** A GUR call that is not qualified with a timestamp always fails in environments without an active ACB library, such as batch regions.

### Special AIB return and reason codes

The following combinations of AIB return and reason codes have specific meanings for the GUR call:

**AIBRETRN = X'000' (CALLCOMP)**

**AIBREASN = X'000' (CALLOK)**

The GUR call completed successfully.

**AIBRETRN = X'100' (CALLOKWE)**

**AIBREASN = X'00C' (PARTDATA)**

The XML response document did not fit in the I/O area. A GUR continuation token is set in the AIBRTKN field.

**AIBRETRN = X'004' (CALLOKWI)**

**AIBREASN = X'004' (LASTSEG)**

This GUR call contains the last portion of response data for a continued GUR call that was previously issued. The GUR continuation token for the call is now invalid.

**AIBRETRN = X'104' (APPLERR)**

**AIBREASN = X'224' (INVAOITK)**

The GUR continuation token passed with the call is invalid.

**AIBRETRN = X'104' (APPLERR)**

**AIBREASN = X'248' (INVPCBN)**

The correct PCB name specified to access the IMS™ catalog was not found.

**AIBRETRN = X'104' (APPLERR)**

**AIBREASN = X'404' (INVFUNC)**

The function code specified on the DL/I call was unknown or invalid.

**AIBRETRN = X'108' (SYSERROR)**

**AIBREASN = X'338' (NOCATMBR)**

The requested catalog member is not in the catalog. IMS searched for a member with the timestamp of the active ACBLIB member, but no member with a matching timestamp was found.

**AIBRETRN = X'108' (SYSERROR)**

**AIBREASN = X'340' (NOGURDLI)**

The GUR call did not find the specified IMS catalog resource in the batch region.

**AIBRETRN = X'108' (SYSERROR)**

**AIBREASN = X'342' (NOGURXML)**

The GUR call was unable to build a valid XML response document.

**AIBRETRN = X'108' (SYSERROR)**

**AIBREASN = X'344' (NOGURNFD)**

The requested catalog member is not in the catalog.

## Example

The following GUR example retrieves the catalog record for the DBD named DX41SK01

```
GUR HEADER (RHDRSEQ=DBD DX41SK01)
```

## Restrictions

The GUR call is valid only for retrieving records from the IMS catalog database. The IMS catalog database must be available.

The GUR call is only supported by the AIB interface.

SSA command codes are not allowed.

### Related concepts:

[➤ Application programming with the IMS catalog \(Application Programming\)](#)

[➤ Overview of the IMS catalog \(Database Administration\)](#)

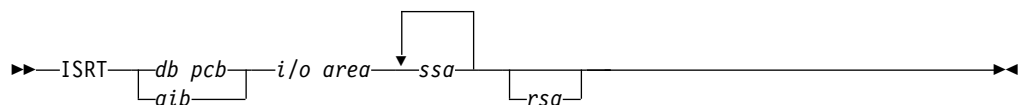
### Related reference:

[➤ AIB return and reason codes \(Messages and Codes\)](#)

## ISRT call

The Insert (ISRT) call is used to load a database and to add one or more segments to the database. You can use ISRT to add a record to the end of a GSAM database or for an alternate PCB that is set up for IAFP processing.

## Format



	Call Name	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
For Full-Function:	ISRT	X	X		X	

	Call Name	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
For GSAM:	ISRT	X	X	X	X	X
For DEDB:	ISRT	X	X	X		
For MSDB:	ISRT	X				

## Parameters

### *db pcb*

Specifies the DB PCB for the call. This parameter is an input and output parameter.

### *aib*

Specifies the AIB for the call. This parameter is an input and output parameter. These fields must be initialized in the AIB:

#### **AIBID**

Eye catcher. 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 DB PCB.

#### **AIBOALEN**

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

### *i/o area*

Specifies the I/O area. This parameter is an input parameter. When you want to add a new segment to the database, you place the new segment in this area before issuing the ISRT call. This area must be long enough to hold the longest segment that IMS returns to this area. For example, if none of the segments your program retrieves or updates is longer than 48 bytes, your I/O area should be 48 bytes.

If your program issues any path calls, the I/O area must be long enough to hold the longest concatenated segment following a path call. The segment data that this area contains is always left-justified. The I/O area points to the first byte of this area.

When you use the ISRT call with GSAM, the area named by the *i/o area* parameter contains the record you want to add. The area must be long enough to hold these records.

### *ssa*

Specifies the SSA, if any, to be used in the call. This parameter is an input parameter. The SSA you supply in the call point to data areas in your program where you have defined the SSAs for the call. You can use up to 15 SSAs on the call. This parameter is required.

### *rsa*

Specifies the area in your program where the RSA should be returned by DL/I. This output parameter is used for GSAM only and is optional. See the topic "GSAM Data Areas" in *IMS Version 13 Application Programming* for more information on RSAs.



## Usage

Your program uses the ISRT call to initially load a database and to add information to an existing one. The call looks the same in either case. However, the way it is used is determined by the processing option in the PCB.

ISRT can add new occurrences of an existing segment type to a HIDAM, PHIDAM, HISAM, HDAM, PHDAM, DEDB, or MSDB database.

**Restriction:** New segments cannot be added to a HSAM database unless you reprocess the whole database or add the new segments to the end of the database.

Before you issue the ISRT call, build the new segment in the I/O area. The new segment fields must be in the same order and of the same length as defined for the segment. (If field sensitivity is used, they must be in the order defined for the application program's view of the segment.) The DBD defines the fields that a segment contains and the order in which they appear in the segment.

### *Root segment occurrence*

If you are adding a root segment occurrence, IMS places it in the correct sequence in the database by using the key you supply in the I/O area. If the segment you are inserting is not a root, but you have just inserted its parent, you can insert the child segment by issuing an ISRT call with an unqualified SSA. You must build the new segment in your I/O area before you issue the ISRT call. Also, you use an unqualified SSA when you insert a root. When you are adding new segment occurrences to an existing database, the segment type must have been defined in the DBD. You can add new segment occurrences directly or sequentially after you have built them in the program's I/O area. At least one SSA is required in an ISRT call; the last (or only) SSA specifies the segment being inserted. To insert a path of segments, you can set the D command code for the highest-level segment in the path.

### *Insert rules*

If the segment type you are inserting has a unique key field, the place where IMS adds the new segment occurrence depends on the value of its key field. If the segment does not have a key field, or if the key is not unique, you can control where the new segment occurrence is added by specifying either the FIRST, LAST, or HERE insert rule. Specify the rules on the RULES parameter of the SEGM statement of DBDGEN for this database.

The rules on the RULES parameter are as follows:

- FIRST** IMS inserts the new segment occurrence before the first existing occurrence of this segment type. If this segment has a nonunique key, IMS inserts the new occurrence before all existing occurrences of that segment that have the same key field.
- LAST** IMS inserts the new occurrence after the last existing occurrence of the segment type. If the segment occurrence has a nonunique key, IMS inserts the new occurrence after all existing occurrences of that segment type that have the same key.
- HERE** IMS assumes you have a position on the segment type from a previous IMS call. IMS places the new occurrence before the segment occurrence that was retrieved or deleted by the last call, which is immediately before

current position. If current position is not within the occurrences of the segment type being inserted, IMS adds the new occurrence before all existing occurrences of that segment type. If the segment has a nonunique key and the current position is not within the occurrences of the segment type with equal key value, IMS adds the new occurrence before all existing occurrences that have equal key fields.

You can override the insert rule of FIRST with the L command code. You can override the insert rule of HERE with either the F or L command code. This is true for HDAM and PHDAM root segments and for dependent segments in any type of database that have either nonunique keys or no keys at all.

An ISRT call must have at least one unqualified SSA for each segment that is added to the database. Unless the ISRT is a path call, the lowest-level SSA specifies the segment being inserted. This SSA must be unqualified. If you use the D command code, all the SSAs below and including the SSA containing the D command code must be unqualified.

Provide qualified SSAs for higher levels to establish the position of the segment being inserted. Qualified and unqualified SSAs can be used to specify the path to the segment, but the last SSA must be unqualified. This final SSA names the segment type to be inserted.

If you supply only one unqualified SSA for the new segment occurrence, you must be sure that current position is at the correct place in the database to insert that segment.

#### *Mix qualified and unqualified SSA*

You can mix qualified and unqualified SSAs, but the last SSA must be unqualified. If the SSAs are unqualified, IMS satisfies each unqualified SSA with the first occurrence of the segment type, assuming that the path is correct. If you leave out a SSA for one of the levels in an ISRT with multiple SSAs, IMS assumes an SSA for that level. The SSA that IMS assumes depends on current position:

- If IMS has a position established at the missing level, the SSA that IMS uses is derived from that position, as reflected in the DB PCB.
- If IMS does not have a position established at the missing level, IMS assumes an unqualified SSA for that level.
- If IMS moves forward from a position established at a higher level, IMS assumes an unqualified SSA for that level.
- If the SSA for the root level is missing, and IMS has position established on a root, IMS does not move from that root when trying to satisfy the call.

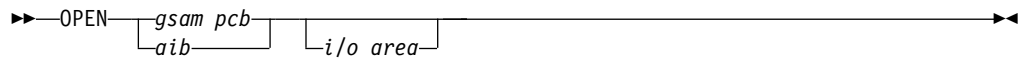
#### *Using SSA with the ISRT call*

Using SSA with ISRT is a good way to check for the parent segments of the segment you want to insert. You cannot add a segment unless its parent segments exist in the database. Instead of issuing Get calls for the parents, you can define a fully qualified set of SSAs for all the parents and issue the ISRT call for the new segment. If IMS returns a GE status code, at least one of the parents does not exist. You can then check the segment level number in the DB PCB to find out which parent is missing. If the level number in the DB PCB is 00, IMS did not find any of the segments you specified. A 01 means that IMS found only the root segment; a 02 means that the lowest-level segment that IMS found was at the second level; and so on.

## OPEN call

The OPEN call is used to explicitly open a GSAM database.

### Format



	Call Name	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
For GSAM:	OPEN	X	X	X	X	X

### Parameters

#### *gsam pcb*

Specifies the GSAM PCB for the call. This parameter is an input and output parameter.

#### *aib*

Specifies the AIB for the call. This parameter is an input and output parameter. These fields must be initialized in the AIB:

#### **AIBID**

Eye catcher. 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 of a GSAM PCB.

#### **AIBOALEN**

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

#### *i/o area*

Specifies the kind of data set you are opening. This parameter is an input parameter.

### Usage

For more information, see the topic "Explicit Open and Close Calls to GSAM" in *IMS Version 13 Application Programming*.

## POS call

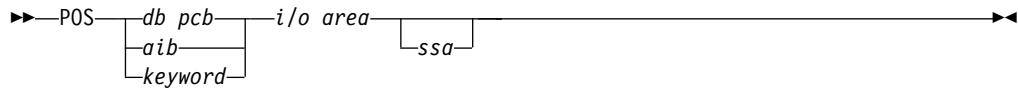
A qualified Position (POS) call is used to retrieve the location of a specific sequential dependent segment. In addition to location, a qualified POS call using an SSA for a committed segment will return the sequential dependent segment (SDEP) time stamp and the ID of the IMS owner that inserted it.

For more information about the qualified POS call, refer to the topic "Processing Fast Path Databases" in *IMS Version 13 Application Programming*.

An unqualified POS points to the logical end of the sequential dependent segment (SDEP) data. By default, an unqualified POS call returns the DMACNXTS value,

which is the next SDEP CI to be allocated. Because this CI has not been allocated, its specification without the EXCLUDE keyword will often result in a DFS2664A message from the SDEP utilities.

### Format



	Call Name	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
For DEDB:	POS	X	X			

### Parameters

#### *db pcb*

Specifies the DB PCB for the DEDB that you are using for this call. This parameter is an input and output parameter.

#### *aib*

Specifies the AIB for the DEDB that you are using for this call. This parameter is an input and output parameter. These fields must be initialized in the AIB:

#### **AIBID**

Eye catcher. 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 DB PCB.

#### **AIBOALEN**

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

#### *keyword*

Specifies the Keyword for the DEDB that you are using for this call. Returns six words containing field codes to I/O area. The following table lists the five keywords and the corresponding output.

#### *i/o area*

Specifies the I/O area in your program that you want to contain the positioning information that is returned by a successful POS call. This parameter is both an input and an output parameter. The I/O area must be long enough to contain all the returned entries. IMS returns an entry for each area in the DEDB.

The I/O area returned on POS call contained six words with nine potential fields of data for each return output. Each field is four or eight bytes. When the successful POS is an unqualified call, the I/O area consists of a 2 byte field that contains the length of the data area (LL), followed by 24 bytes of positioning information. The I/O data area will have 24 bytes of positioning information for every area in the DEDB. By selecting one of the five keywords in position zero of the input I/O area, the user specifies the kind of data in the

return I/O area. The following table lists the five keywords and the data that an unqualified POS call returns based on the keyword you choose for position zero of the input I/O area.

Table 4. Unqualified POS call: keywords and map of the I/O area return output

Keyword	byte 2	word 0	word 1	word 2	word 3	word 4	word 5
<null>	LL	Field 1		Field 2		Field 4	Field 5
V5SEGRBA	LL	Field 1		Field 3		<null>	
PCSEGRTS	LL	Field 1		Field 3		Field 6	
PCSEGHWM	LL	Field 1		Field 3		Field 7	
PCHSEGTS	LL	Field 1		Field 8		Field 6	
PCLBSGTS	LL	Field 1		Field 9		Field 6	

### Field 1

#### Area name

This 8-byte field contains the ddname from the AREA statement.

### Field 2

#### Sequential dependent next to allocate CI

This field is the default if no keyword is specified in position zero of the I/O area. The data returned is the 8-byte cycle count and RBA (CC+RBA) acquired from the global DMACNXTS field. This data represents the next pre-allocated CI as a CI boundary.

### Field 3

#### Local sequential dependent next segment

The data returned is the 8-byte CC+RBA segment boundary of the most recently committed SDEP segment. This data is specific to only the IMS that executes the POS call. Its scope is for local IMS use only.

### Field 4

#### Unused CIs in sequential dependent part

This 4-byte field contains the number of unused control intervals in the sequential dependent part.

### Field 5

#### Unused CIs in independent overflow part

This 4-byte field contains the number of unused control intervals in the independent overflow part.

### Field 6

#### Sequential dependent segment time stamp

The data returned is the 8-byte time stamp of the most recently committed SDEP segment across all IMS partners, or for a local SDEP, the time stamp of the first pre-allocated SDEP CI dummy segment of the local IMS. If the area (either local or shared) has not been opened, or a /DBR was performed without any subsequent SDEP segment inserts, the current time is returned.

### Field 7

#### Sequential dependent High Water Mark (HWM)

This 8-byte field contains the cycle count plus RBA (CC+RBA) of the last pre-allocated SDEP CI which is the High Water Mark (HWM) CI.

### Field 8

#### Highest committed SDEP segment

The data returned is the 8-byte cycle count plus RBA (CC+RBA) for the most recently committed SDEP segment across all IMS partners, or for a local SDEP, the CC+RBA of the most recently committed SDEP segment of the local IMS. If the area (either local or shared) has not been opened, or a /DBR was performed without any subsequent SDEP segment inserts, the HWM CI is returned.

### Field 9

#### Logical begin time stamp

This 8-byte field contains the logical begin time stamp from the DMACSDEP\_LOGICALBEGIN\_TS field.

*ssa*

Specifies the SSA that you want to use in this call. This parameter is an input parameter. The format of SSA in POS calls is the same as the format of SSA in DL/I calls. You can use only one SSA in this parameter. This parameter is optional for the POS call.

### Usage

The POS call:

- Retrieves the location of a specific sequential dependent segment.
- Retrieves the location of last-inserted sequential dependent segment, its time stamp, and the IMS ID.
- Retrieves the time stamp of a sequential dependent segment or Logical Begin.
- Tells you the amount of unused space within each DEDB area. For example, you can use the information that IMS returns for a POS call to scan or delete the sequential dependent segments for a particular time period.

If the area which the POS call specifies is unavailable, the I/O area is unchanged, and the status code FH is returned.

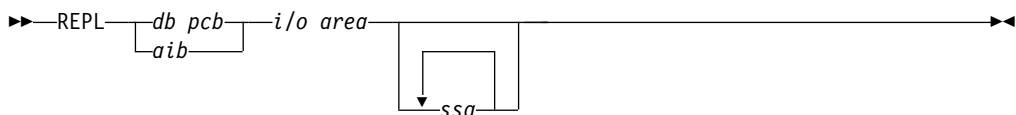
### Restrictions

You can only use the POS call with a DEDB.

### REPL call

The Replace (REPL) call is used to change the values of one or more fields in a segment.

### Format



	Call Name	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
For Full-Function:	REPL	X	X		X	
For DEDB:	REPL	X	X			
For MSDB:	REPL	X				

## Parameters

### *db pcb*

Specifies the DB PCB for the call. This parameter is an input and output parameter.

### *aib*

Specifies the AIB for the call. This parameter is an input and output parameter. These fields must be initialized in the AIB:

#### **AIBID**

Eye catcher. 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 DB PCB.

#### **AIBOALEN**

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

### *i/o area*

Specifies the area in your program that communicates with IMS. This parameter is an input parameter. When you want to replace an existing segment in the database with a new segment, you first issue a Get Hold call to place the new segment in the I/O area. You can modify the data in the I/O area, and then issue the REPL call to replace the segment in the database.

### *ssa*

Specifies the SSA, if any, to be used in the call. This parameter is an input parameter. The SSA you supply in the call point to data areas in your program in which you have defined the SSA for the call. You can use up to 15 SSAs in this parameter. This parameter is optional for the REPL call.

## Usage

A REPL call must be preceded by one of the three Get Hold calls. After you retrieve the segment, you modify it in the I/O area, and then issue a REPL call to replace it in the database. IMS replaces the segment in the database with the segment you modify in the I/O area. You cannot change the field lengths of the segments in the I/O area before you issue the REPL call.

For example, if you do not change one or more segments that are returned on a Get Hold call, or if you change the segment in the I/O area but do not want the change reflected in the database, you can inform IMS not to replace the segment. Specify an unqualified SSA with an N command code for that segment, which tells IMS not to replace the segment.

The N command enables you to tell IMS not to replace one or more of the multiple segments that were returned using the D command code. However, you can specify an N command code even if there were no D command codes on the preceding Get Hold call.

You should not include a qualified SSA on a REPL call. If you do, you receive an AJ status code.

For your program to successfully replace a segment, the segment must have been previously defined as replace-sensitive by PROCOPT=A or PROCOPT=R on the SENSEG statement in the PCB.

If no fields in the segment were changed by the REPL call, the lock is released when the application moves to another database record. Use the Q command code if you need to preserve the segment for the exclusive use of your program.

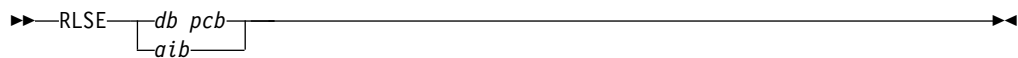
**Related reading:** For more information on the PROCOPT option, see *IMS Version 13 System Utilities*.

If your program attempts to do a path replace of a segment where it does not have replace sensitivity, and command code N is not specified, the data for the segment in the I/O area for the REPL call must be the same as the segment returned on the preceding Get Hold call. If the data changes in this situation, your program receives the status code, AM, and data does not change as a result of the REPL call.

## RLSE call

The Release Locks (RLSE) call is used to release all locks held for unmodified data.

### Format



	Call Name	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
For Full-Function:	RLSE	X	X		X	
For DEDB:	RLSE	X	X		X	

### Parameters

#### *db pcb*

Specifies the DB PCB for the call. This parameter is an input and output parameter.

#### *aib*

Specifies the AIB for the call. This parameter is an input and output parameter. These fields must be initialized in the AIB:

#### **AIBID**

Eye catcher. 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 DB PCB.

### Usage

For Fast Path databases, use the RLSE call to release all locks held for unmodified data that are owned by an application. For full-function databases, use the RLSE call to release the locks held by the DB PCB that is referenced in the call. If the lock is protecting a resource that has been updated, the lock will not be released.



After the RLSE call, all database position information is lost.

### Restrictions

The RLSE call has to be issued using a DB PCB. The PCB cannot be an I/O PCB or an MSDB PCB.

## DL/I calls for IMS DB system services

Use these DL/I calls to obtain IMS DB system services.

Each call description contains:

- A syntax diagram
- Definitions for parameters that are available to the call
- Details on how to use the call in your application program
- Restrictions on call usage, where applicable

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

Syntax diagrams for these calls begin with the *function* parameter. The call interface (xxxTDLI) and *parmcount* (if it is required) are not included in the syntax diagrams.

**Related reading:** For specific information about coding your program in assembler language, C language, COBOL, Pascal, and PL/I, see the topic "Defining Application Program Elements" in *IMS Version 13 Application Programming* for the complete structure.

**Related reference:**

"DL/I calls for IMS TM system services" on page 123

"DL/I calls for transaction management" on page 81

"EXEC DLI commands" on page 162

"IMSCALL command (X'C803')" on page 301

### System service call summary

The following table summarizes which system service calls you can use in each type of IMS DB application program and the parameters for each call. Optional parameters are enclosed in brackets ([ ]).

**Exception:** Language-dependent parameters are not shown here.

For more information on language-dependent application elements, see the topic "Defining Application Program Elements" in *IMS Version 13 Application Programming*.

Table 5. Summary of system service calls.

Function Code	Meaning	Use/Options	Parameters	Valid for
APSB	Allocate PSB	Allocates a PSB for an ODBA application	aib	DB/DC, IMS DB
DPSB	Deallocate PSB	Deallocates a PSB for an ODBA application	aib	DB/DC, IMS DB

Table 5. Summary of system service calls (continued).

Function Code	Meaning	Use/Options	Parameters	Valid for
CHKP (Basic)	Basic checkpoint	Prepares for recovery	function, i/o pcb or aib, i/o area	DB batch, TM batch, BMP, MPP, IFP
CHKP (Symbolic)	Symbolic checkpoint	Prepares for recovery. Specifies up to seven program areas to be saved	function, i/o pcb or aib, i/o area len, i/o area[, area len, area]	DB batch, TM batch, BMP
GMSG	Get Message	Retrieves a message from the AO exit routine. Waits 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), ODBA
GSCD <sup>1</sup>	Get System Contents Directory	Gets address of system contents directory	function, db pcb, i/o pcb or aib, i/o area	DB Batch, TM Batch
ICMD	Issue Command	Issues an IMS command and retrieves the first command response segment	function, aib, i/o area	DB/DC and DCCTL (BMP, MPP, IFP), DB/DC and DBCTL (DRA thread), DBCTL (BMP non-message driven), ODBA
INIT	Initialize application	Receives data availability and deadlock occurrence status codes and checks each PCB database for data availability	function, i/o pcb or aib, i/o area	DB batch, TM batch, BMP, MPP, IFP, DBCTL, ODBA
INQY	Inquiry	Returns information and status codes about I/O or alternate PCB destination type, location, and session status	function, aib, i/o area, AIBFUNC=FINDD  DBQUERY  ENVIRON	DB batch, TM batch, BMP, MPP, IFP, ODBA
LOGb	Log	Writes a message to the system log	function, i/o pcb or aib, i/o area	DB batch, TM batch, BMP, MPP, IFP, DBCTL, ODBA
PCBb	Program Communication Block	Specifies and schedules another PSB	function, psb name, uibptr, [,sysserve]	CICS (DBCTL or DB/DC)
RCMD	Retrieve Command	Retrieves the second and subsequent command response segments resulting from an ICMD call	function, aib, i/o area	DB/DC and DCCTL (BMP, MPP, IFP), DB/DC and DBCTL (DRA thread), DBCTL (BMP non-message driven), ODBA
ROLB	Roll back	Eliminates database updates and returns last message to i/o area	function, i/o pcb or aib, i/o area	DB batch, TM batch, BMP, MPP, IFP
ROLL	Roll	Eliminates database updates	function	DB batch, TM batch, BMP, MPP, IFP

Table 5. Summary of system service calls (continued).

Function Code	Meaning	Use/Options	Parameters	Valid for
ROLS	Roll back to SETS	Issues call using name of DB PCB or i/o PCB and backs out database changes to SETS points	function, db pcb, i/o pcb or aib, i/o area, token	DB batch, TM batch, BMP, MPP, IFP, DBCTL, ODBA
SETS/SETU	Set a backout point	Cancels all existing backout points and establishes as many as nine intermediate backout points	function, i/o pcb or aib, i/o area, token	DB batch, TM batch, BMP, MPP, IFP, DBCTL, ODBA
SNAP <sup>2</sup>		Collects diagnostic information; choose SNAP options	function, db pcb or aib, i/o area	DB batch, BMP, MPP, IFP, CICS (DBCTL or DB/DC), ODBA
STAT <sup>3</sup>	Statistics	Retrieves IMS system statistics; choose type and format	function, db pcb or aib, i/o area, stat function	DB batch, BMP, MPP, IFP, DBCTL, ODBA
SYNC	Synchronization	Releases locked resources and requests commit-point processing	function, i/o pcb or aib	BMP
TERM	Terminate	Releases a PSB so another can be scheduled to commit database changes	function	CICS (DBCTL or DB/DC)
XRST	Extended restart	Specifies up to seven areas to be saved. Works with symbolic checkpoint to restart application program	function, i/o pcb or aib, i/o area len, i/o area[, area len, area]	DB batch, TM batch, BMP

**Note:**

1. GSCD is a Product-sensitive Programming Interface.
2. SNAP is a Product-sensitive Programming Interface.
3. STAT is a Product-sensitive Programming Interface.

**APSB call**

The Allocate PSB (APSB) calls are used to allocate a PSB for an ODBA application.

**Format**

▶▶—APSB—*aib*—▶▶

Call Name	DB/DC	IMS DB	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.

These fields must be initialized in the AIB:

### AIBID

Eye catcher. 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.

### AIBRSNM2

This is the 4-character ID of ODBA startup table representing the target IMS of the APSB.

## Usage

The ODBA application must load or be link edited with the ODBA application interface AERTDLI.

The APSB call must be issued prior to any DLI calls.

The APSB call uses the AIB to allocate a PSB for ODBA application programs.

z/OS Resource Recovery Services (RRS) must be active at the time of the APSB call. If RRS is not active, the APSB call will fail and the application will receive:

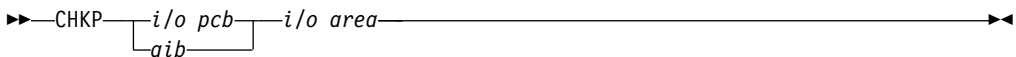
AIBRETRN = X'00000108'  
AIBREASN = X'00000548'

## CHKP (basic) call

A basic Checkpoint (CHKP) call is used for recovery purposes.

The ODBA interface does not support this call.

## 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 for the call. A basic CHKP call must refer to the I/O PCB. This parameter is an input and output parameter.

*aib*

Specifies the AIB for the call. This parameter is an input and output parameter. These fields must be initialized in the AIB:

**AIBID**

Eye catcher. 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 specified in the call list.

*i/o area*

Specifies your program's I/O area that contains the 8-byte checkpoint ID. This parameter is an input parameter. If the program is an MPP or a message-driven BMP, the CHKP call implicitly returns the next input message to this I/O area. Therefore, the area must be large enough to hold the longest returned message.

**Usage**

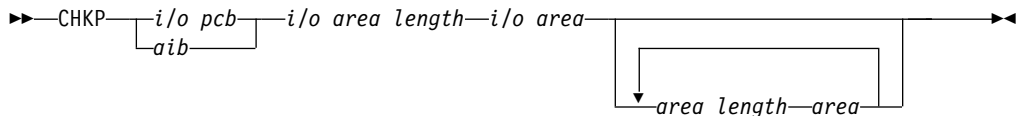
Basic CHKP commits the changes your program has made to the database and establishes places in your program from which you can restart your program, if it terminates abnormally.

**CHKP (symbolic) call**

A symbolic Checkpoint (CHKP) call is used for recovery purposes. If you use the symbolic Checkpoint call in your program, you also must use the XRST call.

The ODBA interface does not support this call.

**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 for the call. This parameter is an input and output parameter. A symbolic CHKP call must refer to the I/O PCB.

*aib*

Specifies the AIB for the call. This parameter is an input and output parameter. These fields must be initialized in the AIB:

**AIBID**

Eye catcher. 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 specified in the call list.

*i/o area length*

This parameter is no longer used by IMS. For compatibility reasons, this parameter must 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 in your program that contains the 8-byte ID for this checkpoint. This parameter is an input parameter. 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 large enough to hold the longest returned message.

*area length*

Specifies a 4-byte field in your program that contains the length (in binary) of the area to checkpoint. This parameter is an input parameter. You can specify up to seven area lengths. For each area length, you must also specify the *area* parameter. All seven *area* parameters (and corresponding length parameters) are optional. When you restart the program, IMS restores only the areas you specified in the CHKP call.

*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. Each area specified must be preceded by an *area length* parameter.

**Usage**

The symbolic CHKP call commits the changes your program has made to the database and establishes places in your program from which you can restart your program, if it terminates abnormally. In addition, the CHKP call:

- Works with the Extended Restart (XRST) call to restart your program if it terminates abnormally
- Enables you to save as many as seven data areas in your program, which are restored when your program is restarted

An XRST call is required before a CHKP call to indicate to IMS that symbolic check points are being taken.

**Restrictions**

The Symbolic CHKP call is allowed only from batch and BMP applications.

**DPSB call**

The DPSB call is used to deallocate IMS DB resources.

## Format

▶—DPSB—*aib*—▶

Call Name	DB/DC	IMS DB	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.

These fields must be initialized in the AIB:

### AIBID

Eye catcher. 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.

### AIBSFUNC

Subfunction code. This field must contain one of the 8-byte subfunction codes as follows:

bbbbbbbb (Null)  
PREPbbbb

## Usage

The DPSB call is used by an application running in a z/OS application region to deallocate a PSB. If the PREP subfunction is not used, the application must activate sync-point processing prior to issuing the DPSB. Use the z/OS Resource Recovery Services (RRS) SRRCMIT/ATRCMIT calls to activate the sync-point process. Refer to *z/OS MVS Programming: Resource Recovery* for more information on these calls.

If the DPSB is issued before changes are committed, and, or locks released, the application will receive:

AIBRETRN = X'00000104'  
AIBREASN = X'00000490'

The thread will not be terminated. The application should issue a SRRCMIT or SRRBACK call, and retry the DPSB.

The PREP sub-function allows the application to issue the DPSB prior to activating the sync-point process. The sync-point activation can occur at a later time, but still must be issued.

## GMSG call

A Get Message (GMSG) call is used in an automated operator (AO) application program to retrieve a message from the AO exit routine DFSAOE00.

## Format

▶—MSG—*aib*—*i/o area*—▶

## 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**

Eye catcher. This 8-byte field must contain DFSAIBbb.

#### **AIBLEN**

AIB lengths. This field must contain the length of the AIB the application actually obtained.

#### **AIBSFUNC**

Subfunction code. This field must contain one of these 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 program.

##### **WAITAOI**

When coded with an AOI token in the AIBRSNM1 field, WAITAOI indicates IMS is to wait for an AOI message when none is currently available for the application program. 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 on the right 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 that is padded on the right with blanks.

#### **AIBOALEN**

I/O area length. This field must contain the length of the I/O area 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



that is passed from IMS to the AO application program. If the I/O area is not large enough to contain all the data, IMS returns partial data.

## Usage

GMSG is used in an AO application program to retrieve a message associated with an AOI token. The AO application program must pass an 8-byte AOI token to IMS in order to retrieve the first segment of the message. IMS uses the AOI token to associate messages from an AO exit routine DFSAOE00 with the GMSG call from an AO application program. IMS returns to the application program only those messages associated with the AOI token. By using different AOI tokens, DFSAOE00 can direct messages to different AO application programs. Note that your installation defines the AOI token.

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 that specifies an AOI token is issued.

Your AO application program can specify a wait on the GMSG call. If no messages are currently available for the associated AOI token, your AO application program waits until a message is available. The decision to wait is specified by the AO application program, 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 application program some GMSG calls can specify waits, while others do not. The following table shows, by IMS environment, the types of AO application programs that can issue GMSG. GMSG is also supported from a CPI-C driven 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 allocate PSB (APSB) call before issuing GMSG.

### GSCD call

A Get System Contents Directory (GSCD) call retrieves the address of the IMS system contents directory for batch programs.

**This topic contains Product-sensitive Programming Interface information.**

The ODBA interface does not support this call.

## Format



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

## Parameters

### *db pcb*

Specifies the DB PCB for the call. This parameter is an input and output parameter.

### *i/o pcb*

Specifies the I/O PCB for the call. This parameter is an input and output parameter.

### *aib*

Specifies the AIB for the call. This parameter is an input and output parameter. These fields must be initialized in the AIB:

#### **AIBID**

Eye catcher. 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 a DB PCB (if a DB PCB is used).

#### **AIBOALEN**

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

### *i/o area*

Specifies the I/O area, which must be 8 bytes long. IMS places the address of the system contents directory (SCD) in the first 4 bytes and the address of the program specification table (PST) in the second 4 bytes. This parameter is an output parameter.

## 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.

## ICMD call

An Issue Command (ICMD) call enables an automated operator (AO) application program to issue an IMS command and retrieve the first command response segment.

### Format

►►—ICMD—*aib*—*i/o area*—————►►

### Parameters

#### *aib*

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

These fields must be initialized in the AIB:

#### **AIBID**

Eye catcher. 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 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 that the command is either in progress or 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 that is passed from the AO application program to IMS, or the largest command response segment that is passed from IMS to the AO application program. If the I/O area is not large enough to contain all the data, IMS returns partial data.

### 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 program's I/O area. After IMS has processed the command, it returns the first segment of the response message to your AO application program's I/O area. To retrieve subsequent segments (one segment at a time) use the RCMD call.

Some IMS commands that complete successfully result in a DFS058 message indicating that the command is complete. Some IMS commands that are processed asynchronously result in a DFS058 message indicating that the command is in progress. For a command entered on an ICMD call, neither DFS058 message is returned to the AO application program. In this case, the AIBOAUSE field is set to 0 to indicate that no segment was returned. So, your AO application program 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 Version 13 Exit Routines*.

The following table shows, by IMS environment, the types of AO application programs that can issue ICMD. ICMD is also supported from a CPI-C driven program.

Table 7. ICMD 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

See *IMS Version 13 Operations and Automation* for a list of commands that can be issued using the ICMD call.

## Restrictions

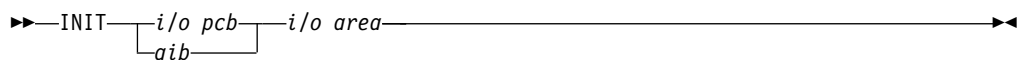
Before issuing ICMD, a CPI-C driven program must issue an allocate PSB (APSB) call.

## INIT call

The Initialize (INIT) call allows an application to receive status codes regarding deadlock occurrences and data availability (by checking each DB PCB).

For GSAM databases, you can use the Initialize (INIT) call to tell IMS that the program can accept a 12-byte record search argument (RSA) when retrieving a record for a large format data set.

## 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 for the call. INIT must refer to the I/O PCB. This parameter is an input and output parameter.

### *aib*

Specifies the AIB for the call. This parameter is an input and output parameter. These fields must be initialized in the AIB:

#### **AIBID**

Eye catcher. 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 specified in the call list.

### *i/o area*

Specifies the I/O area in your program that contains the character string or strings indicating which INIT functions are requested. This parameter is an input parameter.

The functions that you can specify include:

- DBQUERY
- RSA12
- STATUS GROUPA
- STATUS GROUPB
- VERSION

## Usage

You can use the call in any application program, including IMS batch in a sharing environment.

Specify the function in your application program with a character string in the I/O area.

For example, use the format LLZZ Character-String, where LL is the length of the character string including the length of the LLZZ portion; ZZ must be binary 0. For PL/I, you must define the LL field as a fullword; the value is the length of the character string including the length of the LLZZ portion, minus 2. If the I/O area is invalid, an AJ status code is returned. The following tables contain sample I/O areas for INIT when it is used with assembler language, COBOL, C language, Pascal, and PL/I.

### *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.

Application programs that use the language-independent AIB interface or the language-specific interfaces for the assembler, COBOL, C, or Pascal programming languages use a 2-byte LL field to specify the length of the I/O area. The following table shows an example of the INIT call I/O area with the LLZZ length field and DBQUERY specified.

*Table 8. INIT DBQUERY example for the AIB, ASMTDLI, CBLTDLI, CTDLI, and PASTDLI interfaces*

L	L	Z	Z	Character String
00	0B	00	00	DBQUERY

**Note:** The LL value of X'0B' is a hexadecimal representation of decimal 11. ZZ fields are binary.

The following table contains a sample I/O area for the INIT call with DBQUERY for PL/I. The PLITDLI interface uses a 4-byte LLLL field for the length of the I/O area.

*Table 9. INIT DBQUERY: I/O area example for PLITDLI*

L	L	L	L	Z	Z	Character String
00	00	00	0B	00	00	DBQUERY

**Note:** The LL value of X'0B' is a hexadecimal representation of decimal 11. ZZ fields are binary.

#### **LL or LLLL**

A 2-byte field that contains the length of the character string, plus two bytes for LL. For the PLITDLI interface, use the 4-byte field LLLL. When you use the AIB interface (AIBTDLI), 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.

**bb** The data that can be accessed with this PCB can be used for all functions that the PCB allows. DEDBs and MSDBs always have the bb 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. The segment name

field contains one of the following database organizations: DEDB, MSDB, GSAM, HDAM, PHDAM, HIDAM, PHIDAM, HISAM, HSAM, INDEX, SHSAM, or SHISAM.

For a DCCTL environment, the database organization is UNKNOWN.

**Important:** If you are working with a High Availability Large Database (HALDB), you need to be aware that the feedback on data availability at PSB schedule time only shows the availability of the HALDB master, not of the HALDB partitions. However, the error settings for data unavailability of a HALDB partition are the same as those of a non-HALDB database, namely status code 'BA' or pseudoabend U3303.

### *Automatic INIT DBQUERY*

When the program is initially scheduled, the status code in the database PCBs is initialized as if the INIT DBQUERY call were issued. The application program can therefore determine database availability without issuing the INIT call.

### *Performance considerations for the INIT call (IMS online only)*

For a DCCTL environment, the status code is NA.

For performance reasons, the INIT call should not be issued before the first GU call to the I/O PCB. If the INIT call is issued first, the GU call is not processed as efficiently.

### *Determining data availability status without abends*

To avoid abendu3303, first use INIT STATUS GROUPx (x=A or B). IMS will give you a status code for unavailable databases (or HALDB partitions). Then, use INIT DBQUERY, which will set a status code in each DB PCB. Before attempting any DB call, you can test all PCBs for non-blank status.

### *Enabling data availability status codes: INIT STATUS GROUPA*

The following table contains a sample I/O area for the INIT call for assembler language, COBOL, C language, and Pascal.

*Table 10. INIT I/O area examples for ASMTDLI, CBLTDLI, CTDLI, and PASTDLI*

L	L	Z	Z	Character String
00	11	00	00	STATUS GROUPA

**Note:** The LL value of X'11' is a hexadecimal representation of decimal 17. ZZ fields are binary.

The following table contains a sample I/O area for the INIT call for PL/I.

*Table 11. INIT I/O area examples for PLITDLI*

L	L	L	L	Z	Z	Character String
00	00	00	11	00	00	STATUS GROUPA

**Note:** The LL value of X'11' is a hexadecimal representation of decimal 17. ZZ fields are binary.

**LL or LLLL**

LL is a halfword-length field. For non-PLITDLI calls, LLLL is a fullword-length field for PLITDLI.

**ZZ** A 2-byte field of binary zeros.

The value for LLZZ data or LLLLZZ data is always 4 bytes (for LLZZ or LLLLZZ), plus data length.

**Recommendation:** You should be familiar with data availability.

When the INIT call is issued with the character string STATUS GROUPA in the I/O area, the application program informs IMS that it is prepared to accept status codes regarding data unavailability. IMS then returns a status code rather than a resultant pseudoabend if a subsequent call requires access to unavailable data. The status codes that are returned when IMS encounters unavailable data are BA and BB. Status codes BA and BB both indicate that the call could not be completed because it required access to data that was not available. DEDBs can receive the BA or BB status code.

In response to status code BA, the system backs out only the updates that were done for the current call before it encountered the unavailable data. If changes have been made by a previous call, the application must decide to commit or not commit to these changes. The state of the database is left as it was before the failing call was issued. If the call was a REPL or DLET call, the PCB position is unchanged. If the call is a Get type or ISRT call, the PCB position is unpredictable.

In response to status code BB, the system backs out all database updates that the program made since the last commit point and cancels all nonexpress messages that were sent since the last commit point. The PCB position for all PCBs is at the start of the database.

**Enabling deadlock occurrence status codes: INIT STATUS GROUPB**

The following table contains a sample I/O area for the INIT call for assembler language, COBOL, C language, and Pascal.

*Table 12. INIT I/O area examples for ASMTDLI, CBLTDLI, CTDLI, and PASTDLI*

L	L	Z	Z	Character String
00	11	00	00	STATUS GROUPB

**Note:** The LL value of X'11' is a hexadecimal representation of decimal 17. ZZ fields are binary.

The following table contains a sample I/O area for the INIT call for PL/I.

*Table 13. INIT I/O area examples for PLITDLI*

L	L	L	L	Z	Z	Character String
00	00	00	11	00	00	STATUS GROUPB

**Note:** The LL value of X'11' is a hexadecimal representation of decimal 17. ZZ fields are binary.

**LL or LLLL**

LL is a halfword-length field. For non-PLITDLI calls, LLLL is a fullword-length field for PLITDLI.



**ZZ** A 2-byte field of binary zeros.

The value for LLZZ data or LLLLZZ data is always four bytes (for LLZZ or LLLLZZ), plus data length.

When the INIT call is issued with the character string STATUS GROUPB in the I/O area, the application program informs IMS that it is prepared to accept status codes regarding data unavailability and deadlock occurrences. The status codes for data unavailability are BA and BB, as described under "Enabling data availability status codes: INIT STATUS GROUPA".

When a deadlock occurs in batch and the INITSTATUS GROUPB call has been issued, the following occurs:

- If no changes were made to the database, the BC status code is returned.
- If updates were made to the database, and if a datalog exists and BKO=YES is specified, the BC status code is returned.
- If changes were made to the database, and a disklog does not exist or BKO=YES is not specified, a 777 pseudoabend occurs.

When the application program encounters a deadlock occurrence, IMS:

- Backs out all database resources (with the exception of GSAM and DB2<sup>®</sup>) to the last commit point. Although GSAM PCBs can be defined for pure batch or BMP environments, GSAM changes are not backed out. Database resources are backed out for DB2 only when IMS is the sync-point coordinator.

When you use INIT STATUS GROUPB in a pure batch environment, you must specify the DISKLOG and BACKOUT options.

- Backs out all output messages to the last commit point.
- Requeues all input messages as follows:

**Environment**

**Action**

**MPP and BMP**

All input messages are returned to the message queue. The application program no longer controls its input messages.

- IFP** All input messages are returned to IMS Fast Path (IFP) balancing group queues (BALGRP), making them available to any other IFP region on the BALGRP. The IFP that is involved in the deadlock receives the next transaction or message that is available on the BALGRP.

**DBCTL**

Action is limited to resources that are managed by DBCTL, for example, database updates.

- Returns a BC status code to the program in the database PCB.

***Determining GSAM databases for large format data sets: INIT RSA12***

When you issue the INIT call with the character string "RSA12" set in the I/O area, the GSAM application program tells IMS that the program can accept a 12-byte RSA when retrieving a record for a large format data set. The following table contains a sample I/O area for the INIT call with RSA12 for assembler language, COBOL, C language, and Pascal.

Table 14. INIT RAS12: Examples for ASMTDLI, CBLTDLI, CTDLI, and PASTDLI

L	L	Z	Z	Character string
00	09	00	00	RSA12

**Note:** The LL value of X'09' is a hexadecimal representation of decimal 9. ZZ fields are binary.

The following table contains a sample I/O area for the INIT call with RSA12 for PL/I.

Table 15. INIT RSA12: Example for PLITDLI

L	L	L	L	Z	Z	Character string
00	00	00	09	00	00	RSA12

**Note:** The LL value of X'09' is a hexadecimal representation of decimal 9. ZZ fields are binary.

### LL or LLLL

A 2-byte or 4-byte field that contains the total length of the I/O area. For PL/I, the length of the LLLL field is considered 2 bytes even though it is a 4-byte field. When you use the AIBTDLI interface, the length of the record is equal to the total length of LL + ZZ + character string. For the PLITDLI interface, the length of the record is equal to the total length of LLLL + ZZ + character string, where LLLL is considered 2 bytes.

**ZZ** A 2-byte field of binary zeros.

### Specify a database version number: INIT VERSION(*dbname=version*)

When database versioning is enabled, an application program can use the "VERSION" function to request a version of a database that is different from the version number that is specified for the application program on the PCB or from the default version that is returned by IMS. A version number specified on the INIT VERSION call takes precedence over all other version specifications and defaults.

When the INIT VERSION call is not issued prior to a DL/I to access a database, the version of the database that is returned to the application program is determined by the DBVER keyword of the PCB statement. If the DBVER keyword is not specified, IMS returns either the version of the database that is active in the ACB library or version 0 of the database, as determined by the DBLEVEL keyword in either the PSBGEN statement or the database section of the DFSDFxxx PROCLIB member.

In the I/O area, the VERSION function is specified by using the following format:

►►—VERSION( , *dbname=version* )—►►

Each database name is specified by using alphabetic characters and can be specified only once. Specify only names of physical databases. The names of logical databases are not supported.

Each version is specified as a numeric value from 0 to 2147483647. The number that is specified must match a version number that is defined on a DBD for the named database and stored in the IMS catalog.

Calculate the size that is required for the I/O area by multiplying the number of databases that are specified in the input I/O area by 20.

For example, the following table contains a sample I/O area for the INIT VERSION call for assembler language, COBOL, C language, and Pascal. In the table, the LL value of X'3C' is the hexadecimal representation of decimal 60, the length in bytes that is required to hold the output in the I/O area when three database names are specified on input. The ZZ fields are binary.

*Table 16. INIT VERSION: Example format for ASMTDLI, CBLTDLI, CTDLI, and PASTDLI*

L	L	Z	Z	Character string
00	3C	00	00	VERSION (DBa=1,DBb=2,DBc=3)

The following table contains a sample I/O area for the INIT call with VERSION for PL/I. In the table, the LL value of X'3C' is the hexadecimal representation of decimal 60. The ZZ fields are binary.

*Table 17. INIT VERSION: Example format for PLITDLI*

L	L	L	L	Z	Z	Character string
00	00	00	3C	00	00	VERSION (DBa=1,DBb=2,DBc=3)

#### **LL or LLLL**

A 2-byte or 4-byte field that contains the total length of the I/O area. For PL/I, the length of the LLLL field is considered 2 bytes even though it is a 4-byte field. When you use the AIBTDLI interface, the length of the record is equal to the total length of LL + ZZ + character string. For the PLITDLI interface, the length of the record is equal to the total length of LLLL + ZZ + required length for output, where LLLL is considered 2 bytes.

**ZZ** A 2-byte field of binary zeros.

#### **Character string**




The function specification on input. The length that is specified in the LL or LLLL is the length that is required for the output: 20 bytes for each database that is specified in the input character string.

### **Restrictions**

For function shipping in the CICS environment, the local and remote CICS must both be DBCTL.

You should be familiar with deadlock occurrences as described in *IMS Version 13 System Administration*.

#### **Related concepts:**

-  Retrieving and inserting GSAM records (Application Programming)
-  Converting HDAM and HIDAM databases to HALDB (Database Administration)
-  Data availability considerations (Application Programming)

## 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 for application interfaces that use the AIB structure.

### Format

▶▶—INQY—*aib*—*i/o area*—————▶▶

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 (DFSAIB) for the call. This parameter is an input and output parameter. These fields must be initialized in the AIB:

#### **AIBID**

Eye catcher. 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
- ENVIRONb
- FINDbbbb
- LERUNOPT
- MSGINFOb
- PROGRAMb (Not supported with the ODBA interface)

#### **AIBRSNM1**

Resource name. This 8-byte, left-justified field must contain the name of any named PCB in the PSB.

#### **AIBOALEN**

I/O area length. This field must contain the length of the I/O area specified in the call list. This field is not changed by IMS.

#### *i/o area*

Specifies the data output area to use with the call. This parameter is an output parameter. An I/O area is required for INQY subfunctions ENVIRONb, MSGINFOb and PROGRAMb. It is not required for subfunctions DBQUERYb, FINDbbbb, and LERUNOPT.

### Restrictions

The INQY call is valid only when using the AIB. An INQY call issued through the PCB interface is rejected with an AD status code.

## Usage

The INQY call operates in both batch and online IMS environments. IMS application programs can use the INQY call to request information regarding the output destination, the session status, the current execution environment, the availability of databases, and the PCB address, which is based on the PCB name. You must use the AIB when issuing an INQY call. Before you can issue an INQY call, initialize the fields of the AIB.

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 program receives.

The INQY call returns information to the caller's I/O area. The length of the data that is returned from the INQY call is passed back to the application program 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 information, 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 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 INITDBQUERY call. The INQY DBQUERY call does not return information in the I/O area, but like the INIT DBQUERY call, it updates status codes in the database PCBs.

The application program is not made aware of the status of each PCB until an INQY FIND call is issued. To retrieve the status for a database, you must pass the DB PCB for that database in the INQY FIND call.

In addition to the INIT DBQUERY status codes, the INQY DBQUERY call returns these status codes in the I/O PCB:

- bb** 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 that is 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 the call has not been issued.

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 been issued. 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.

**bb** The data that can be accessed with this PCB can be used for all functions the PCB allows. DEDBs and MSDBs always have the bb.

### 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 IOPCBbbb. This includes the IMS identifier, release, region, and region type.

The INQY ENVIRON call returns character-string data. The output is left justified and padded with blanks on the right.

**Recommendations:** To account for expansion in the length of the reply data, specify an I/O area length of 512 bytes.

To reference the field that contains the recovery token or the application parameter string, code your application programs to locate the field by using the address of the field that is returned in the data output of the INQY ENVIRON call. This is the only valid programming technique to reference the recovery token field and the application parameter string field. No other programming technique should be used to reference these fields.

The recovery token or the application parameter string are optional and therefore are not always returned. If they are not returned, the value in the address field is zero.

For more information about the recovery token and application parameter fields, see note 2 after the following table.

The following table lists the output that is returned from the INQY ENVIRON call. Included with the information returned is the outputs byte length, the actual value, and an explanation.

*Table 18. INQY ENVIRON data output*

Information returned	Length in bytes	Actual value	Explanation
IMS Identifier	8		Provides the identifier from the execution parameters.
IMS Release Level	4		Provides the release level for IMS. For example, X'00000410'.

Table 18. INQY ENVIRON data output (continued)

Information returned	Length in bytes	Actual value	Explanation
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.
		JBP	Indicates that the Java batch processing region is active.
		JMP	Indicates that the Java message processing region is active.
		MPP	Indicates that the Message Processing region is active.
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.
		b	Indicates that no associated transaction exists.
User Identifier <sup>1</sup>	8		Provides the user ID.
		b	Indicates that the user ID is unavailable.
Group Name	8		Provides the group name.
		b	Indicates that the group name is unavailable.
Status Group Indicator	4	A	Indicates an INIT STATUS GROUPA call is issued.
		B	Indicates an INIT STATUS GROUPB call is issued.
		b	Indicates that a status group is not initialized.
Address of Recovery Token <sup>2</sup>	4		Provides the address of the LL field, followed by the recovery token.
		0	Indicates that the recovery token is not available.
Address of the Application Parameter String <sup>2</sup>	4		Provides the address of the LL field, followed by the application program parameter string.
		0	Indicates that the APARM= parameter is not coded in the execution parameters of the dependent region JCL.
Shared Queues Indicator	4		Indicates IMS is not using Shared Queues.
		SHRQ	Indicates IMS is using Shared Queues.
User ID of Address Space	8		User ID of dependent address space.

Table 18. INQY ENVIRON data output (continued)

Information returned	Length in bytes	Actual value	Explanation
User ID Indicator	1		<p>Contains one of the following possible values to indicate the contents of the userid field:</p> <p><b>U</b> Indicates the user's identification from the source terminal during sign-on.</p> <p><b>L</b> Indicates the LTERM name of the source terminal in sign-on is not active.</p> <p><b>P</b> Indicates the PSBNAME of the source BMP or transaction.</p> <p><b>O</b> Indicates some other name.</p>
z/OS Resource Recovery Services (RRS) Indicator	3	b	Indicates that IMS has not expressed interest in the UR with RRS. Therefore, the application should refrain from performing any work that causes RRS to become the syncpoint manager for the UR because IMS will not be involved in the commit scope. For example, the application should not issue any outbound protected conversations.
		RRS	Indicates IMS has expressed interest in the UR with RRS. Therefore, IMS will be involved in the commit scope if RRS is the syncpoint manager for the UR.
IMS catalog enablement indicator	7	b	<p>Indicates that the IMS catalog is not enabled in the DFSDFxxx PROCLIB member.</p> <p>For information about setting up and enabling an IMS catalog, see IMS catalog definition and tailoring (System Definition).</p> <p>For information about enabling the IMS catalog in the DFSDFxxx PROCLIB member, see DFSDFxxx member of the IMS PROCLIB data set (System Definition).</p>
		CATALOG	Indicates that the IMS catalog is enabled. Database and application metadata is available in IMS.

**Notes:**

- 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 that is currently scheduled into the BMP region.
  - For message-driven BMP regions that have completed a successful GU call and for any MPP region, the PSTUSID field is derived which is usually the input terminal's RACF® ID. If the terminal has not signed on to RACF, the ID is the input terminal's LTERM.
- The pointer is an address that identifies a length field (LL) which contains the length of the recovery token or application program parameter string in binary, including the two bytes required for LL. Use this pointer to set up addressability of the AIB between releases in a batch program.

**Querying the input message information: INQY MSGINFO**

To obtain information regarding the current input message, use the INQY call with the MSGINFO subfunction. The only valid PCB name that can be passed in the AIBRSNM1 field is IOPCBbbb. The output returns the version number and the output fields for the message information. The INQY MSGINFO call returns the response in the I/O area.



The following table lists the output that is returned from the INQY MSGINFO call. Included with the information returned is the byte length, the actual value, and an explanation of the output.

*Table 19. INQY MSGINFO data output*

Information returned	Length in bytes	Actual value	Explanation
Version number	4	1	Output response version 1.
Origin IMSID	8		The IMS identifier from which the input message originated.
Reserved for IMS	68		This field is reserved for future output expansion.

### Querying the PCB: 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 only valid PCB names that can be passed in AIBRSNM1 are IOPCBbbb or the name of an alternate PCB or DB PCB, as defined in the PSB. The PCB address is returned in the AIBRSA1 field of the AIB mask. When the INQY call is completed, the AIBRSA1 field contains call-specific information.

To retrieve the status for a database, you must pass the DB PCB for that database in the INQY FIND call. You must issue one call for each PCB required.

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 program to analyze the PCB status code to determine if either an NA or NU status code is set in the PCB.

The following PL/I code sample shows how to retrieve the database status values.

```

II00_INITSTAT: PROC;
  DCL DUMMY_LENGTH CHAR(4) INIT(' '); /* TO PLEASE IMS */
  AIB.PCBNAME      = 'IOPCB';
  CALL AIBTDLI($3,INIT,AIB,STATUS_CALL2);
  IF AIB.RETURN = 0 THEN
    PUT SKIP LIST('INIT ISSUED');
  ELSE
    DO;
      PUT SKIP LIST ('AIB RETURN CODE  ',AIB.RETURN);
      PUT SKIP LIST ('AIB REASON CODE  ',AIB.REASON);
      PUT SKIP LIST ('IOPCB STATUS CODE ',IO_PCB.STATUS_CODE);
      PUT SKIP LIST ('INIT UNSUCCESSFULL');
    END;
  SELECT (IO_PCB.STATUS_CODE);
  WHEN (' ')
    GROUPA_STATUS = ' ';
  WHEN ('NA')
    GROUPA_STATUS = 'NA';
  WHEN ('NU')
    GROUPA_STATUS = 'NU';
  OTHERWISE
    DO;
      PUT SKIP LIST

```

```

        ('INIT STATUS GROUPA FAILED ',IO_PCB.STATUS_CODE);
    END;
END;
PUT SKIP LIST
('INIT STATUS GROUPA = ',IO_PCB.STATUS_CODE);
END II00_INITSTAT;
JJ00_INQY: PROC;
    DCL DUMMY_LENGTH CHAR(4) INIT(' '); /* TO PLEASE IMS */
    AIB.PCBNAME      = 'IOPCB';
    AIB.SUB_FUNC     = 'DBQUERY  ';
    AIB.OUT_LEN_TOT = 2000;
    CALL AIBTDLI($3,INQY,AIB,IO_AREA);
    PUT SKIP LIST('INQY ISSUED ON IOPCB BEFORE CHECK OF AIB RETURN');
    IF AIB.RETURN = 0 THEN
        PUT SKIP LIST('INQY ISSUED - 0 RC ON AIB.RETURN');
    ELSE
    DO;
        PUT SKIP LIST ('AIB RETURN CODE ',AIB.RETURN);
        PUT SKIP LIST ('AIB REASON CODE ',AIB.REASON);
        PUT SKIP LIST ('IOPCB STATUS CODE ',IO_PCB.STATUS_CODE);
        PUT SKIP LIST ('INQY IOPCB DBQUERY UNSUCCESSFULL');
    END;
    SELECT (IO_PCB.STATUS_CODE);
    WHEN (' ')
    DO;
        PUT SKIP DATA (IO_AREA);
        PUT SKIP DATA (IO_PCB.STATUS_CODE);
    END;
    WHEN ('NA')
        PUT SKIP LIST ('NA STATUS ON IO_PCB.STATUS_CODE');
    WHEN ('NU')
        PUT SKIP LIST ('NU STATUS ON IO_PCB.STATUS_CODE');
    OTHERWISE
    DO;
        PUT SKIP LIST
        ('INQY FAILED ',IO_PCB.STATUS_CODE);
    END;
END;
PUT SKIP LIST ('START B1CSTP FIND CALL');
AIB.PCBNAME      = 'B1CSTP';
AIB.SUB_FUNC     = 'FIND  ';
AIB.OUT_LEN_TOT = 2000;
CALL AIBTDLI($3,INQY,AIB,IO_AREA);
PUT SKIP LIST('INQY B1CSTP FIND READY TO BE CALLED');
IF AIB.RETURN = 0 THEN
    PUT SKIP LIST('INQY B1CSTP FIND CALLED - 0 RC');
ELSE
DO;
    PUT SKIP LIST ('AIB RETURN CODE ',AIB.RETURN);
    PUT SKIP LIST ('AIB REASON CODE ',AIB.REASON);
    PUT SKIP LIST ('CSTP_PCB STATUS CODE ',CSTP_PCB.STATUS_CODE);
    PUT SKIP LIST ('INQY B1CSTP FIND UNSUCCESSFULL');
END;
PUT SKIP LIST ('CSTP STATUS ',CSTP_PCB.STATUS_CODE);
PUT SKIP LIST ('IO PCB ', IO_PCB.STATUS_CODE);
SELECT (CSTP_PCB.STATUS_CODE);
WHEN (' ')
DO;
    PUT SKIP DATA (CSTP_PCB.STATUS_CODE);
    PUT SKIP DATA (IO_AREA);
END;
WHEN ('NA')
    PUT SKIP LIST ('NA STATUS ON B1CSTP CSTPPCB.STATUS_CODE');
WHEN ('NU')
    PUT SKIP LIST ('NU STATUS ON B1CSTP CSTPPCB.STATUS_CODE');
OTHERWISE
DO;

```

```

        PUT SKIP LIST
        ('INQY FAILED ',IO_PCB.STATUS_CODE);
    END;
END;
PUT SKIP LIST ('START D1CSTP FIND CALL');
AIB.PCBNAME      = 'D1CSTP';
AIB.SUB_FUNC     = 'FIND      ';
AIB.OUT_LEN_TOT = 2000;
CALL AIBTDLI($3,INQY,AIB,IO_AREA);
PUT SKIP LIST('INQY D1CSTP FIND READY TO BE CALLED');
IF AIB.RETURN = 0 THEN
    PUT SKIP LIST('INQY D1CSTP FIND CALLED - 0 RC');
ELSE
    DO;
        PUT SKIP LIST ('AIB RETURN CODE ',AIB.RETURN);
        PUT SKIP LIST ('AIB REASON CODE ',AIB.REASON);
        PUT SKIP LIST ('CSTP_PCB STATUS CODE ',CSTP_PCB.STATUS_CODE);
        PUT SKIP LIST ('INQY D1CSTP FIND UNSUCCESSFULL');
    END;
    PUT SKIP LIST ('CSTP STATUS ',CSTP_PCB.STATUS_CODE);
    PUT SKIP LIST ('IO PCB ', IO_PCB.STATUS_CODE);
    SELECT (CSTP_PCB.STATUS_CODE);
    WHEN (' ')
        DO;
            PUT SKIP DATA (CSTP_PCB.STATUS_CODE);
            PUT SKIP DATA (IO_AREA);
        END;
    WHEN ('NA')
        PUT SKIP LIST ('NA STATUS ON D1CSTP CSTPPCB.STATUS_CODE');
    WHEN ('NU')
        PUT SKIP LIST ('NU STATUS ON D1CSTP CSTPPCB.STATUS_CODE');
    OTHERWISE
        DO;
            PUT SKIP LIST
            ('INQY FAILED ',IO_PCB.STATUS_CODE);
        END;
END;
PUT SKIP LIST ('START S1CSTP FIND CALL');
AIB.PCBNAME      = 'XXCSTP';
AIB.SUB_FUNC     = 'FIND      ';
AIB.OUT_LEN_TOT = 2000;
CALL AIBTDLI($3,INQY,AIB,IO_AREA);
PUT SKIP LIST('INQY S1CSTP FIND READY TO BE CALLED');
IF AIB.RETURN = 0 THEN
    PUT SKIP LIST('INQY S1CSTP FIND CALLED - 0 RC');
ELSE
    DO;
        PUT SKIP LIST ('AIB RETURN CODE ',AIB.RETURN);
        PUT SKIP LIST ('AIB REASON CODE ',AIB.REASON);
        PUT SKIP LIST ('CSTP_PCB STATUS CODE ',CSTP_PCB.STATUS_CODE);
        PUT SKIP LIST ('INQY S1CSTP FIND UNSUCCESSFULL');
    END;
    PUT SKIP LIST ('CSTP STATUS ',CSTP_PCB.STATUS_CODE);
    PUT SKIP LIST ('IO PCB ', IO_PCB.STATUS_CODE);
    SELECT (CSTP_PCB.STATUS_CODE);
    WHEN (' ')
        DO;
            PUT SKIP DATA (CSTP_PCB.STATUS_CODE);
            PUT SKIP DATA (IO_AREA);
        END;
    WHEN ('NA')
        PUT SKIP LIST ('NA STATUS ON S1CSTP CSTPPCB.STATUS_CODE');
    WHEN ('NU')
        PUT SKIP LIST ('NU STATUS ON S1CSTP CSTPPCB.STATUS_CODE');
    OTHERWISE
        DO;

```

```

        PUT SKIP LIST
        ('INQY FAILED ',IO_PCB.STATUS_CODE);
    END;
END;

```

## Querying for LE overrides: INQY LERUNOPT

When the LERUNOPT call is issued with the LERUNOPT subfunction, IMS determines if LE overrides are allowed based on the LEOPT system parameter. The LE override parameters are defined to IMS through the UPDATE LE command. IMS checks to see if there are any overrides applicable to the caller based on the specific combinations of transaction name, lterm name, userid, or program name in the callers environment. IMS will return the address of the string to the caller if an override parameter is found. The LE overrides are used by the IMS supplied CEEBXITA exit, DFSBXITA, to allow dynamic overrides for LE runtime parameters.

The call string must contain the function code and the AIB address. The I/O area is not a required parameter and will be ignored if specified. The only valid PCB name that can be passed in AIBRSNM1 is IOPCB. The AIBOALEN and AIBOAUSE fields are not used.

The rules for matching an entry that results in it being returned on a DL/I INQY LERUNOPT call are:

- An MPP or JMP region uses transaction name, lterm, userid, and program to match with each entry.
- An IFB, JBP, or non-message driven BMP uses program name to match with each entry. If an entry has a defined filter for transaction name, lterm, or userid, it does not match. Message driven BMPs also use transaction name.
- The entries are scanned to find the entry with the most filter matches. The first entry in the list with the most exact filter matches is selected. The scan stops with an entry found with all of the filters matching the entry.

**Note:** Searching table entries may cause user confusion because of the way entries are built and searched. For example, assume there are two entries in the table that match on the filters specified on the DL/I INQY call. The first transaction matches on transaction name and lterm name. The second entry matches on transaction name and program name. IMS chooses the first entry because it was the first entry encountered with highest number of filter matches. If the second entry is now updated with a longer parameter string, which causes a new entry to be built, it will be added to the head of the queue. The next search would result in the entry with transaction name and program name being selected. This could result in a set of runtime options being selected that were not expected by the user.

## 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. 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

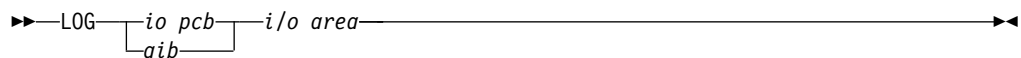
Table 20. Subfunction, PCB, and I/O area combinations for the INQY call

Subfunction	I/O PCB	Alternate PCB	DB PCB	I/O Area Required
FIND	OK	OK	OK	NO
ENVIRON	OK	NO	NO	YES
DBQUERY	OK	NO	NO	NO
LERUNOPT	OK	NO	NO	NO
PROGRAM	OK	NO	NO	YES
MSGINFO	OK	NO	NO	YES

## 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 I/O PCB for the call. This parameter is an input and output parameter.

### *aib*

Specifies the AIB for the call. This parameter is an input and output parameter. These fields must be initialized in the AIB:

#### **AIBID**

Eye catcher. 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 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 is an input parameter. This record must follow the format shown in the following tables.

Table 21. Log record formats for COBOL, C, assembler, Pascal, and PL/I programs for the AIBTDLI, ASMTDLI, CBLTDLI, CEETDLI, CTDLI, and PASTDLI interfaces

LL	ZZ	C	Text
2	2	1	Variable

Table 22. Log record formats for COBOL, C, assembler, Pascal, and PL/I programs for the PLITDLI interface

LLLL	ZZ	C	Text
4	2	1	Variable

The fields must be:

**LL or LLLL**

Specifies a 2-byte field (or, for PL/I, a 4-byte-long field) to contain the length of the record. The length of the record is equal to LL + ZZ + C + text of the record. When you calculate the length of the log record, you must account for all 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, 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 different log codes to distinguish between different types of information.

You can issue the LOG call:

- In a batch program, and the record is written to the IMS log
- In an online program in the DBCTL environment, and the record is written to the DBCTL log
- In the IMS DB/DC environment, and the record is written to the IMS 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 four bytes, or the I/O area specified in the IOASIZE keyword of the PSBGEN statement of the PSB.

For function shipping in the CICS environment, the local and remote CICS must both be DBCTL.

## PCB call (CICS online programs only)

The PCB call is used to schedule a PSB call.

The ODBA interface does not support this call.

### Format

►►—PCB—*psb name*—*uibptr*——*sysserve*—►►

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

### Parameters

The AIB is not valid for PCB calls.

#### *psb name*

Specifies the PSB. An asterisk can be used for the parameter to indicate the default. This parameter is an input parameter.

#### *uibptr*

Specifies a pointer, which is set to the address of the UIB after the call. This parameter is an output parameter.

#### *sysserve*

Specifies an optional 8-byte field that contains either IOPCB or NOIOPCB. This parameter is an input parameter.

### Usage

Before a CICS online program can issue any DL/I calls, it must indicate to DL/I its intent to use a particular PSB. A PCB call accomplishes this and also obtains the address of the PCB list in the PSB. When you issue a PCB call, specify:

- The call function: PCBb
- The PSB you want to use, or an asterisk to indicate that you want to use the default name. The default PSB name is not necessarily the name of the program issuing the PCB call, because that program could have been called by another program.
- A pointer, which is set to the address of the UIB after the call.  
For more information on defining and establishing addressability to the UIB, see the topic "Specifying the UIB (CICS Online Programs Only)" in *IMS Version 13 Application Programming*.
- The system service call parameter that names an optional 8-byte field that contains either IOPCB or NOIOPCB.

### Restrictions

For function shipping in the CICS environment, the local and remote CICS must both be DBCTL.

## RCMD call

A Retrieve Command (RCMD) call enables an automated operator (AO) application program retrieve the second and subsequent command response segments after an ICMD call.

### Format

▶▶—RCMD—*aib*—*i/o area*—————▶▶

### Parameters

#### *aib*

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

These fields must be initialized in the AIB:

#### **AIBID**

Eye catcher. 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 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 that is passed from IMS to the AO application program. 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 program retrieve the second and subsequent command response segments resulting from an ICMD call.

**Related reading** For more information on the AOI exits, see *IMS Version 13 Exit Routines*.

The following table shows, by IMS environment, the types of AO application programs that can issue RCMD. RCMD is also supported from a CPI-C driven program.

Table 23. RCMD support by application region type

Application region type	IMS environment		
	DBCTL	DB/DC	DCCTL
DRA thread	Yes	Yes	N/A



Table 23. RCMD support by application region type (continued)

Application region type	IMS environment		
	DBCTL	DB/DC	DCCTL
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 one time for each response segment that is issued by IMS.

### Restrictions

An ICMD call must be issued before an RCMD call.

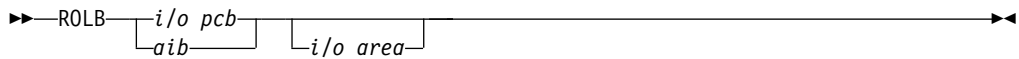
### ROLB call

The Roll Back (ROLB) call is used to dynamically back out database changes and return control to your program.

For more information on the ROLB call, see the topic "Maintaining Database Integrity" in *IMS Version 13 Application Programming*.

The ODBA interface does not support this call.

### 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 for the call. This parameter is an input and output parameter.

#### *aib*

Specifies the AIB for the call. This parameter is an input and output parameter. These fields must be initialized in the AIB:

#### **AIBID**

Eye catcher. 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 specified in the call list.

#### *i/o area*

Specifies the area in your program where IMS returns the first message segment. This parameter is an output parameter.

### Restrictions

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

### ROLL call

The Roll (ROLL) call is used to abnormally terminate your program and to dynamically back out database changes.

For more information on the ROLL call, see the topic "Maintaining Database Integrity" in *IMS Version 13 Application Programming*.

The ODBA interface does not support this call.

### Format

►► ROLL ◀◀

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

When you issue a ROLL call, IMS terminates the application program with a U0778 abend.

### Restrictions

Unlike the ROLB call, the ROLL call does not return control to the program.

### ROLS call

The Roll Back to SETS (ROLS) call is used to back out to a processing point set by a prior SETS or SETU call.

For more information on the ROLS call, see the topic "Maintaining Database Integrity" in *IMS Version 13 Application Programming*.

### Format

►► ROLS ◀◀

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

## Parameters

### *db pcb*

Specifies the DB PCB for the call. This parameter is an input and output parameter.

### *i/o pcb*

Specifies the I/O PCB for the call. This parameter is an input and output parameter.

### *aib*

Specifies the AIB for the call. This parameter is an input and output parameter. These fields must be initialized in the AIB:

#### **AIBID**

Eye catcher. 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, or the name of a DB PCB.

#### **AIBOALEN**

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

### *i/o area*

Specifies the I/O area has the same format as the I/O area supplied on the SETS call. This parameter is an output parameter.

### *token*

Specifies the area in your program that contains a 4-byte identifier. This parameter is an input parameter.

## Usage

When you use the Roll Back to SETS (ROLS) call to back out to a processing point set by a prior SETS or SETU, the ROLS enables you to continue processing or to back out to the prior commit point and place the input message on the suspend queue for later processing.

Issuing a ROLS call for a DB PCB can result in the user abend code 3303.

## Restrictions

For function shipping in the CICS environment, the local and remote CICS must both be DBCTL.

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 a 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 is accepted even if unsupported PCBs exist or an external subsystem is used. For more information on the SETS and SETU calls, see the topic "Maintaining Database Integrity" in *IMS Version 13 Application Programming*.

### 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 for the call. SETS and SETU must refer to the I/O PCB. This parameter is an input and output parameter.

#### *aib*

Specifies the AIB for the call. This parameter is an input and output parameter. These fields must be initialized in the AIB:

#### **AIBID**

Eye catcher. 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 specified in the call list.

#### *i/o area*

Specifies the area in your program that contains the data to be returned on the corresponding ROLS call. This parameter is an input parameter.

#### *token*

Specifies the area in your program that contains a 4-byte identifier. This parameter is an input parameter.

### Usage

The SETS and SETU format and parameters are the same, except for the call functions, SETS and SETU.

The SETS and SETU calls provide the backout points that IMS uses in the ROLS call. The ROLS call operates with the SETS and SETU call backout points.

The meaning of the SC status code for SETS and 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 either that unsupported PCBs exist in the PSB or the application program made calls to an external subsystem.

## Restrictions

For function shipping in the CICS environment, the local and remote CICS must both be DBCTL.

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. The SETU call is valid, but not functional, if unsupported PCBs exist in the PSB or if the program uses an external subsystem.

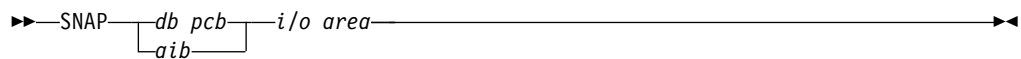
Before a ROLS call, you can specify a maximum of 255 SETS calls with the same token and still back out to the correct message level. After 255 SETS calls, the messages continue to back out, but only to the same message level as at 255th SETS call. The SETS token count resets to zero during sync point processing.

## SNAP call

The SNAP call is used to collect diagnostic information.

**This topic contains Product-sensitive Programming Interface information.**

## Format



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

## Parameters

### *db pcb*

Specifies the address that refers to a full-function PCB that is defined in a calling program PSB. This parameter is an input and output parameter.

### *aib*

Specifies the AIB for the call. This parameter is an input and output parameter. These fields must be initialized in the AIB:

#### **AIBID**

Eye catcher. 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 full-function DB PCB.

**AIBOALEN**

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

*i/o area*

Specifies the area in your program that contains SNAP operation parameters. This parameter is an input parameter. The following figure shows the SNAP operation parameters you specify, including:

- Length for bytes 1 through 2
- Destination for bytes 3 through 10
- Identification for bytes 11 through 18
- SNAP options for bytes 19 through 22

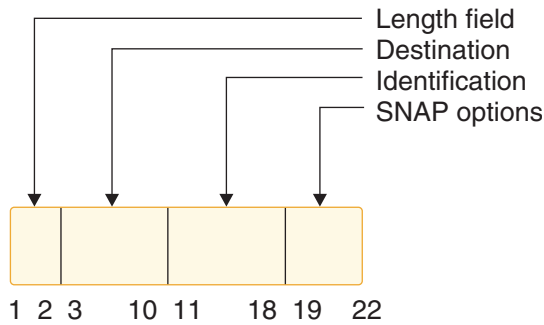


Figure 2. I/O area for SNAP operation parameters

The following table explains the values that you can specify.

Table 24. SNAP operation parameters

Byte	Value	Meaning															
1-2	xx	This 2-byte binary field specifies the length of the SNAP operation parameters. The length must include this 2-byte length field.  When you do not specify operation parameters, IMS uses default values. This chart lists the lengths that result from your parameter specifications.															
		<table border="1"> <thead> <tr> <th>If you supply values for:</th> <th>And IMS supplies default values for:</th> <th>Then the length (in hexadecimal) is:</th> </tr> </thead> <tbody> <tr> <td>Destination, Identification, SNAP options</td> <td></td> <td>16</td> </tr> <tr> <td>Destination, Identification</td> <td>SNAP options</td> <td>12</td> </tr> <tr> <td>Destination</td> <td>Identification, SNAP options</td> <td>10</td> </tr> <tr> <td></td> <td>Destination, Identification, SNAP options</td> <td>2</td> </tr> </tbody> </table>	If you supply values for:	And IMS supplies default values for:	Then the length (in hexadecimal) is:	Destination, Identification, SNAP options		16	Destination, Identification	SNAP options	12	Destination	Identification, SNAP options	10		Destination, Identification, SNAP options	2
If you supply values for:	And IMS supplies default values for:	Then the length (in hexadecimal) is:															
Destination, Identification, SNAP options		16															
Destination, Identification	SNAP options	12															
Destination	Identification, SNAP options	10															
	Destination, Identification, SNAP options	2															
If you specify another length, IMS uses default values for the destination, identification, and SNAP operation parameters.																	

Table 24. SNAP operation parameters (continued)

Byte	Value	Meaning
3-10		This 8-byte field tells IMS where to send SNAP output. You can direct output to the IMS log by specifying LOG or bbbbbb
		Directs the output to the IMS log. This is the default destination.
	<i>dcbaddr</i>	Directs the output to the data set defined by this DCB address.  The application program must open the data set before the SNAP call refers to it. This option is valid only in a batch environment. The output data set must conform to the rules for a z/OS SNAP data set.
	<i>ddname</i>	Directs the output to the data set defined by the corresponding DD statement. The DD statement must conform to the rules for a z/OS SNAP data set. The data set specified by <i>ddname</i> is opened and closed for this SNAP request.  In a DB/DC environment, you must supply the DD statement in the JCL for the control region.  If the destination is invalid, IMS directs output to the IMS log.
11-18	<i>cccccccc</i>	This is an eight-character name you can supply to identify the SNAP. If you do not supply a name, IMS uses the default value, NOTGIVEN.
19-22	<i>cccc</i>	This four-character field identifies which data elements you want the SNAP output to include. YYYYN is the default.
19		<b>Buffer Pool:</b>
	Y	Dump all buffer pools and sequential buffering control blocks with a SNAP call.
	N	Do not dump buffer pools or sequential buffering control blocks with a SNAP call.
20		<b>Control Blocks:</b>
	Y	Dump control blocks related to the current DB PCB with a SNAP call.
	N	Do not dump control blocks related to the current DB PCB with a SNAP call.
21	Y	Dump all control blocks for this PSB with a SNAP call. Specifying Y in byte 21 produces a snap dump for the current DB PCB request in byte 20 to Y, regardless of the current value.
	N	Do not dump all control blocks for this PSB with a SNAP call. In this case, the current DB PCB SNAP request in position 20 is used as specified.
19-21	ALL	This is equivalent to specifying YYY in positions 19-21.
22		<b>Region:</b>
	Y	Dump the entire region on the DCB address or data set <i>ddname</i> that you supplied in bytes 3-10 with a SNAP call. IMS processes this request before it acts on any SNAP requests made in bytes 19-21. If the destination is the IMS log, IMS does not dump the entire region. Instead, it processes the request as if you had specified ALL.
	N	Do not dump the entire region with a SNAP call.
	S	Dump subpools 0-127 with a SNAP call.

After the SNAP call, IMS can return the AB, AD, or bb (blank) status code. For a description of these codes and the response required, see *IMS Version 13 Messages and Codes, Volume 4: IMS Component Codes*.

## Usage

Any application program can issue this call.

## Restrictions

For function shipping in the CICS environment, the local and remote CICS must both be DBCTL.

## STAT call

The Statistics (STAT) call is used in a CICS, IMS online, or batch program to obtain database statistics that might be useful for performance monitoring.

**This topic contains Product-sensitive Programming Interface information.**

## Format

▶▶ STAT db pcb  
aib i/o area—stat function—▶▶

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

## Parameters

### *db pcb*

Specifies the DB PCB used to pass status information to the application program. The VSAM statistics used by the data sets associated with this PCB are not related to the type of statistics that is returned from the STAT call. This PCB must reference a full-function database. This parameter is an input and output parameter.

### *aib*

Specifies the AIB for the call. This parameter is an input and output parameter. These fields must be initialized in the AIB:

#### **AIBID**

Eye catcher. 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 full-function DB PCB.

#### **AIBOALEN**

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

### *i/o area*

Specifies an area in the application program that is large enough to hold the



requested statistics. This parameter is an output parameter. In PL/I, this parameter should be specified as a pointer to a major structure, array, or character string.

#### *stat function*

Specifies a 9-byte area whose content describes the type and format of the statistics required. The first 4 bytes define the type of statistics requested and byte 5 defines the format to be provided. The remaining 4 bytes contain EBCDIC blanks. If the *stat function* that is provided is not one of the defined functions, an AC status code is returned. This parameter is an input parameter. The 9-byte field contains:

- 4 bytes that define the type of statistics you want:
  - DBAS** OSAM database buffer pool statistics
  - DBES** OSAM database buffer pool statistics, enhanced or extended
  - VBAS** VSAM database subpool statistics
  - VBES** VSAM database subpool statistics, enhanced or extended
- 1 byte that gives the format of the statistics:
  - F** Full statistics to be formatted. If you specify F, your I/O area must be at least 360 bytes for DBAS or VBAS and 600 bytes for DBES or VBES.
  - O** Full OSAM database subpool statistics in a formatted form. If you specify O, your I/O area must be at least 360 bytes.
  - S** Summary of the statistics to be formatted. If you specify S, your I/O area must be at least 120 bytes for DBAS or VBAS and 360 bytes for DBES or VBES.
  - U** Full statistics to be unformatted. If you specify U, your I/O area must be at least 72 bytes.
- 4 bytes of EBCDIC blanks for normal or enhanced STAT call, or `bE1b`
  - Restriction:** The extended format parameter is supported by the DBESO, DBESU, and DBESF functions only.
  - Extended OSAM buffer pool statistics can be retrieved by including the parameter `bE1b` following the enhanced call function. The extended STAT call returns all of the statistics returned with the enhanced call, plus the statistics on the coupling facility buffer invalidates, OSAM caching, and sequential buffering IMMED and SYNC read counts.

## Usage

The STAT call can be helpful in debugging because it retrieves IMS database statistics. It is also helpful in monitoring and tuning for performance. The STAT call retrieves OSAM database buffer pool statistics and VSAM database buffer supports.

When you request VSAM statistics, each issued STAT call retrieves the statistics for a subpool. Statistics are retrieved for all VSAM local shared resource pools in the order in which they are defined. For each local shared resource pool, statistics are retrieved in ascending order based on buffer size. Statistics for index subpools always follow those for data subpools if any index subpool exists in the shared resource pool. The index subpools are also retrieved in ascending order based on buffer size.

For more information on the STAT call, see *IMS Version 13 Application Programming*.

## Restrictions

For function shipping in the CICS environment, the local and remote CICS must both be DBCTL.

### Related concepts:

 Retrieving database statistics: the STAT call (Application Programming)

## SYNC call

The Synchronization Point (SYNC) call is used to release resources that IMS has locked for the application program.

The ODBA interface does not support this call.

## Format



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

## Parameters

### *i/o pcb*

Specifies the IO PCB for the call. This parameter is an input and output parameter.

### *aib*

Specifies the AIB for the call. This parameter is an input and output parameter. These fields must be initialized in the AIB:

#### **AIBID**

Eye catcher. 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.

## Usage

SYNC commits the changes your program has made to the database, and establishes places in your program from which you can restart, if your program terminates abnormally.

## Restrictions

The SYNC call is valid only in non-message driven BMPs; you cannot issue a SYNC call from an CPI-C driven application program.

For important considerations about using the SYNC call, see *IMS Version 13 Database Administration*.

### TERM call (CICS online programs only)

The Terminate (TERM) call is used to terminate a PSB in a CICS online program.

The ODBA interface does not support this call.

#### Format



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

#### Usage

If your program needs to use more than one PSB, you must issue a TERM call to release the first PSB it uses and then issue a second PCB call to schedule the second PSB. The TERM call also commits database changes.

The only parameter in the TERM call is the call function: TERM or bbb. When your program issues the call, CICS terminates the scheduled PSB, causes a CICS sync point, commits changes, and frees resources for other tasks.

#### Restrictions

For function shipping in the CICS environment, the local and remote CICS must both be DBCTL.

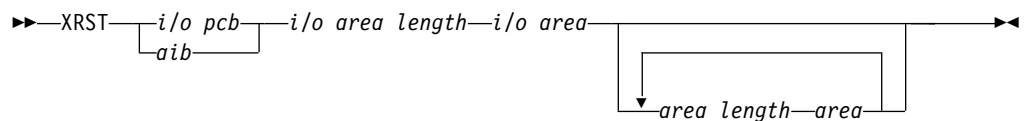
### XRST call

The Extended Restart (XRST) call is used to restart your program.

If you use the symbolic Checkpoint call in your program, you must precede it with an XRST call that specifies checkpoint data of blanks.

The ODBA interface does not support this call.

#### 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 for the call. XRST must refer to the I/O PCB. This parameter is an input and output parameter.

*aib*

Specifies the AIB for the call. This parameter is an input and output parameter. These fields must be initialized in the AIB:

**AIBID**

Eye catcher. 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 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 14-byte area in your program. This area must be either set to blanks if you are starting your program normally or, if performing an extended restart, have a checkpoint ID.

*area length*

Specifies a 4-byte field in your program that contains the length (in binary) of the area to restore. This parameter is an input parameter. You can specify up to seven area lengths. For each area length, you must specify the *area* parameter. All seven *area* parameters (and corresponding *area length* parameters) are optional. When you restart the program, IMS restores only the areas specified on the CHKP call.

The number of areas you specify on an XRST call must be less than or equal to the number of areas you specify on a CHKP call.

*area*

Specifies the area in your program that you want IMS to restore. You can specify up to seven areas. Each area specified must be preceded by an *area length*. This is an input parameter.

## Usage

Programs that want 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.

To determine whether to perform a normal start or a restart, IMS evaluates 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. However, an altered I/O area indicates that you are restarting your program. Consequently, your program must handle the specified data areas that were previously saved and that are now restored.

### *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 (left-most 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 parameter 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 DFS0540I, 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 DFS0540I supplies the checkpoint ID and the time stamp.

The system message DFS682I 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.

At 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 (IIIIDDHHMMSST).

If the program being restarted is in a DL/I batch region, the IMSLOGR DD statement that defines the log data set must be supplied in the JCL. IMS reads these data sets and searches for the checkpoint records that have the ID that was specified.

However, if the program being restarted is in a BMP region and all of the following conditions are met, an IMSLOGR DD statement is not required:

- The BMP program is restarted with CKPTID=LAST.
- The BMP program is restarted on the same IMS system with the same job name, same PSB, and same program name that was used when it abended.
- IMS has not been cold-started since the BMP program abended.

- The checkpoint records that are needed to restart the program are on an OLDS data set that has not been archived and reused since the time of the abend, or the SLDSREAD logger function is active in IMS.

If any of the preceding conditions are not met, you must supply an IMSLOGR DD statement that points to the data set that contains the required checkpoint records.

If an IMSLOGR DD statement is supplied, it must contain the required checkpoint log records. IMS does not automatically locate and retrieve checkpoint records for a BMP if an IMSLOGR DD statement is present. Only the IMSLOGR DD data set is searched and, if the record is not found, the BMP program terminates with abend U0102.

**Note:** A DD DUMMY statement is permissible for an IMSLOGR DD statement and is treated as if no IMSLOGR DD statement was supplied.

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 (IIIIDDHHMMSST).

Also check the status code in the I/O PCB. The only successful status code for an XRST call are blanks.

#### *Position in the database after issuing XRST*

The XRST call attempts to reposition all databases to the position that was held when the last checkpoint was taken. This is done by including each PCB and PCB key feedback area in the checkpoint record. Issuing XRST causes the key feedback area from the PCB in the checkpoint record to be moved to the corresponding PCB in the PSB that is being restarted. Then IMS issues a GU call, qualified with the concatenated key (using the C command code), for each PCB that held a position when the checkpoint was taken.

After the XRST call, the PCB reflects the results of the GU repositioning call, not the value that was present when the checkpoint was taken. The GU call is not made if the PCB did not hold a position on a root or lower-level segment when the checkpoint was taken. A GE status code in the PCB means that the GU for the concatenated key was not fully satisfied. The segment name, segment level, and key feedback length in the PCB reflect the last level that was satisfied on the GU call. A GE status code can occur because IMS is unable to find a segment that satisfies the segment search argument that is associated with a Get call for one of the following reasons:

- The call preceding the checkpoint call was a DLET call issued against the same PCB. In this case, the position is correct because the position after the Get call does not find its target is the same position that would exist following the DLET call.

**Restriction:** Avoid taking a checkpoint immediately after a DLET call.

- The segment was deleted by another application program between the time your program terminated abnormally and the time you restarted your program. A GN call issued after the restart returns the first segment that follows the deleted segment or segments.

This explanation assumes that position at the time of checkpoint was on a segment with a unique key. XRST cannot reposition to a segment if that segment or any of its parents have a non-unique key.

For a DEDB, the GC status code is received when position is not on a segment but at a unit-of-work (UOW) boundary. Because the XRST call attempts to reestablish position on the segment where the PCB was positioned when the symbolic checkpoint was taken, the XRST call does not reestablish position on a PCB if the symbolic checkpoint is taken when the PCB contains a GC status code.

If your program accesses GSAM databases, the XRST call also repositions these databases.

During GSAM XRST processing, a check is made to determine if the GSAM output data set to be repositioned is empty and if the abending job had previously inserted records in the data set.

### 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 right-most 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 application programs.

---

## Transaction management

Use the following reference information to make DL/I calls for transaction management.

### DL/I calls for transaction management

Use these DL/I calls with IMS TM to perform transaction management functions in your application programs.

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 transaction management 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 syntax diagrams. See language-specific information (for COBOL, C language,

Pascal, PL/I, and assembler language) in the topic "Defining Application Program Elements" in *IMS Version 13 Application Programming* for the complete structure.

### Transaction Management Call Summary

The following table summarizes the parameters that are valid for each of the transaction management message calls. The following table lists the function code, its meaning, use, parameters, and in which regions it is valid. 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 the topic "Formatting DL/I Calls for Language Interfaces" in *IMS Version 13 Application Programming* for language-specific information.

**Related reading:** For information on writing calls with programming language interfaces, see the topic "Defining Application Program Elements" in *IMS Version 13 Application Programming*.

Table 25. 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
ICAL	IMS Call	Sends a synchronous request for data or services to a non-IMS application program or service that runs in a distributed environment	aib, request area, response 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
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



Table 25. Summary of TM message calls (continued)

Function Code	Meaning	Use	Parameters	Valid for
SETO	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

**Related reading:** DCCTL users can issue calls using GSAM database PCBs, which are described in *IMS Version 13 Application Programming*.

**Related reference:**

“DL/I calls for IMS DB system services” on page 35

“DL/I calls for database management” on page 1

“EXEC DLI commands” on page 162

**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**



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

**Parameters**

*i/o pcb*

Specifies the I/O PCB, the first PCB address in the list that is passed to the program. 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**

Eye catcher. 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 used for the call. This parameter is an input and output parameter.

**I/O Area**

The following tables show the format of the parameter list in the I/O area before the AUTH call is issued.

*I/O area before the AUTH call*

*Table 26. I/O area before the AUTH call is issued for AIBTDLI, ASMTDLI, CBLTDLI, CEETDLI, CTDLI, and PASTDLI interfaces*

Field Name	Field Length
LL	2
ZZ	2
CLASSNAME	8
RESOURCE	8
USERDATA	8

*Table 27. I/O area before the AUTH call is issued for the PLITDLI interface*

Field Name	Field Length
LLLL	4
ZZ	2
CLASSNAME	8
RESOURCE	8
USERDATA	8

**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:

- TRANbbbb
- DATABASE
- SEGMENTbb
- FIELDbbb
- OTHERbbb

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).

The following tables show the I/O area after the AUTH call.

*I/O area after the AUTH call*

*Table 28. I/O area after the AUTH call is issued for AIBTDLI, ASMTDLI, CBLTDLI, CEETDLI, CTDLI, and PASTDLI interfaces*

Field Name	Field Length
LL	2
ZZ	2
FEEDBACK	2
EXITRC	2
STATUS	2
RESERVED	16
UL	2
USERDATA	Variable

*Table 29. I/O area after the AUTH call is issued for the PLITDLI interface*

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

**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 (DFSCTRNO 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 information units. For additional information, see *IMS Version 13 System Administration* and *IMS Version 13 Exit Routines*.

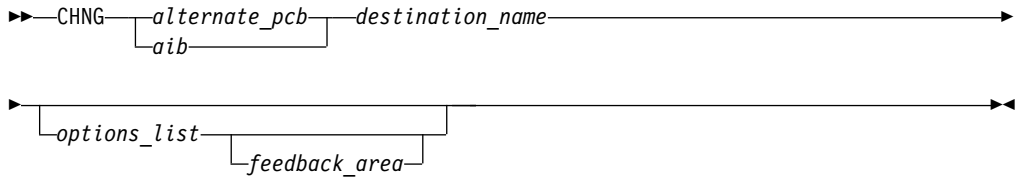
## Restrictions

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**

Eye catcher. 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 Version 13 Communications and Connections*.

The destination name may also be used to implement message switches from OTMA to non-OTMA destinations. In this case, the destination name must match the name of the routing descriptor in the DFSYDTx member of IMS.PROCLIB.

**Restriction:** Some destination names are invalid. For more information on resource naming rules, see *IMS Version 13 Communications and Connections*.

**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.

For more information on APPC, see *IMS Version 13 Communications and Connections*.

The format for the *options list* is shown here:

<b>LL or LLLL</b> <sup>1, 2, 3</sup>	<b>ZZ</b>	<b>keyword1=variable1</b>
Halfword length of the options string, including the 4-byte length of LLZZ or LLLLZZ.	Halfword of zero.	CHNG options separated by commas.

**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 following table shows the format for the feedback area passed to IMS in the call list:

<b>LL or LLLL</b> <sup>1, 2</sup>	<b>ZZ</b>
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:

<b>LLZZ or LLLLZZ</b>	<b>LL</b>	<i>feedback data</i>
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.

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 below.

### *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 Version 13 Exit Routines*.

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 Version 13 Communications and Connections*.



### *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

These 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 sync point.

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 by:

- 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:

<b>LL</b>	<i>outdes options</i>
Halfword length of the total OUTDES printer options, including the 2-byte length of LL.	Any valid combination of OUTDES printer options.
<b>Note:</b> Some options depend on the release level of MVS™.	

**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 SET0 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 SET0 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.

**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.

**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.

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 CPI Communications Reference. The format for the TPN option is as follows:

<b>LL</b>	<i>tpn</i>
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. 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**=**NC**

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=BM

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 Version 13 Communications and Connections*.

*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 1.5 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 z/OS 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 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.

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.

### Related reference:

“ISRT call” on page 112

“PURG call” on page 115

## 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, the first PCB address in the list that is passed to the program. 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**

Eye catcher. 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. 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.

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.

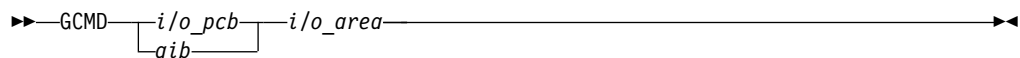
**Related reference:**

“GCMD call”

**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, the first PCB address in the list that is passed to the program. 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**

Eye catcher. 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, 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.

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.

#### Related reference:

“CMD call” on page 95

### 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.

### Format



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

### Parameters

#### *i/o pcb*

Specifies the I/O PCB, the first PCB address in the list that is passed to the program. 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**

Eye catcher. 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.

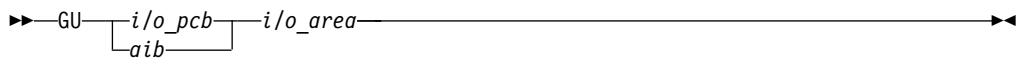
**Related reference:**

“GU call”

**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, the first PCB address in the list that is passed to the program. 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**

Eye catcher. 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.

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 the topic "I/O PCB mask" in *IMS Version 13 Application Programming*)
- 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 time stamp (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 the topic "Specifying the I/O PCB Mask" in *IMS Version 13 Application Programming*.

## 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.

### Related reference:

"GN call" on page 98

## ICAL call

The IMS Call (ICAL) call allows an application program that runs in the IMS TM environment to send a synchronous request for data or services to a non-IMS application program or service that runs in a z/OS or distributed environment, or to initiate a synchronous program switch to an IMS transaction.

### Format for the SENDRECV subfunction

►►—ICAL—*aib*—*request\_area*—*response\_area*—◄◄

### Format for the RECEIVE subfunction

►►—ICAL—*aib*—*response\_area*—◄◄

Call name	DB/DC	DBCTL	DCCTL	DB batch	TM batch
ICAL	X		X		

## Parameters

### *aib*

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

You must initialize the following fields in the AIB:

#### **AIBERRXT**

This 4-byte length field contains the additional error information that is returned by OTMA, IMS Connect, IMS TM Resource Adapter, the IMS Enterprise Suite SOAP Gateway server, or user-written IMS Connect client applications. The default is 0.

#### **AIBID**

Eye catcher. 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**

A 4-byte field that, when an ICAL call is issued, must contain the length of the request area.

When a response to an ICAL call is received, if the response data is too large to fit in the response area, the AIBOALEN field contains the total length of the entire response data. When the response area is too small to fit all of the response data, the AIB return code is X'100' and the AIB reason code is X'00C'. For any other return codes that are received with a response, this field is unchanged.

When partial data is returned, you can use the value of this field to determine how much space is required in the response data buffer. Your application program can then expand the buffer and issue an ICAL call with the RECEIVE subfunction code to retrieve the complete response message.

#### **AIBOAUSE**

A 4-byte field that, when an ICAL is issued, contains the length of the output response area that is specified in the call list.

When a response to an ICAL call is received, IMS updates the field to contain the length of the response message that is returned in the response area. If only partial data is returned because the response area is not large enough, AIBOAUSE contains the length of the data that is returned in the response area, and AIBOALEN contains the total length of the response message.

#### **AIBREASN**

AIB reason code.

#### **AIBRETRN**

AIB return code.

#### **AIBRSFLD**

The time to wait for the synchronous call process to complete. This 4-byte field contains a time value in 100th of a second.

The valid range is 0 - 999999. The system default is 10 seconds.

- If the specified value is larger than the maximum value, the maximum value is used.
- If the value is set to 0, then the timeout value that is specified in the OTMA descriptor is used. If there is no timeout value in the OTMA descriptor, the system default is used for the timeout.
- If the timeout value specified on the OTMA destination descriptor differs from the timeout value specified on the ICAL call, OTMA uses the smaller of the two values.

When the timeout value is reached, the IMS application that issues the synchronous callout request receives a return code of X'0100' and a reason code of X'0104'. The message is discarded.

#### **AIBRSNM1**

OTMA descriptor name. This 8-byte, alphanumeric, left-aligned field must contain the name of the OTMA descriptor that defines the destination of the IMS call.

#### **AIBRSNM2**

This 8-byte, alphanumeric, left-aligned field contains the logical terminal name used to override the LTERM name in the I/O PCB of the IMS application program for the target transaction of an ICAL call for synchronous program switch. The name specified in the AIB is used instead of any name specified in the OTMA destination descriptor. However, if no name is specified in AIBRSNM2, the name from the OTMA

descriptor is used. If no name is found in the descriptor or in the AIB, the IMS application terminal symbolic (PSTSYMBO) is used as the default logical terminal name for the target transaction.

#### **AIBSFUNC**

Subfunction code. This field must contain an 8-byte subfunction code. The valid subfunction codes are:

##### **SENDRECV**

The IMS application program uses this subfunction to send a message and wait for the response. This subfunction is used for synchronous program-to-program communication.

##### **RECEIVE**

The IMS application program uses this subfunction to retrieve the complete response data from a previous incomplete ICAL call. If a SENDRECV subfunction call completes with AIB return code X'0100' and reason code X'000C', the response data did not fit in the response area. The application program can expand the response area and then retrieve the complete response with the RECEIVE subfunction call.

#### **AIBUTKN**

Map name. If specified, this 1- to 8-byte alphanumeric, left-justified field contains the 1- to 8-character map name used for message formatting or service identification purpose. This map name is included in the OTMA state data prefix sent to the destination for callout.

#### ***request\_area***

Specifies the request area to use for this call. This parameter is an input parameter.

This request area contains the request message data that is sent from the IMS application program to the application that is specified in the OTMA descriptor. The AIBOALEN field specifies the length of the request message data. Because the ICAL call bypasses the IMS TM message queue, the format of the request area does not require the LLZZ fields.

If the OTMA descriptor specifies that the request message must be routed to another IMS application program (TYPE=IMSTRAN), the LLZZ fields and transaction code must be specified in the first 8 bytes of the data area that follows the LLZZ. For transactions specified with MULTSEG, the request data must include the entire multi-segment message. The standard IMS LLZZ format is required for each segment, but the transaction code is only required for the first segment.

**LL** Specifies the length of the segment.

**ZZ** Sets the segment to binary zeros.

#### ***response\_area***

Specifies the response area to use for this call. This parameter is an output parameter.

If the response area is not large enough to contain all of the returned data, IMS returns partial data. When partial data is returned, the AIBOAUUSE field contains the length of the returned data in the response area, and AIBOALEN contains the actual length of the response message.

Because an ICAL call for synchronous callout bypasses the IMS TM message queue, the format of the response area does not require the LLZZ fields. However, ICAL calls for synchronous program switch to another IMS application do require the LLZZ fields. The LLZZ fields for a synchronous

program switch are populated by the output from the target IMS application. Synchronous program switch requests do not bypass the message queue.

If the original request message was routed to another IMS application program, the response data follows the standard LLZZ format for each segment in the response message.

### **Usage: SENDRECV subfunction**

An ICAL call is used in an IMS application program for synchronous callout that does not use the IMS message queue. Because the IMS message queue is not used, synchronous callout messages are not constrained to the 32 KB message segment restriction.

However, An ICAL call that is used in an IMS application program for synchronous program switch processing to an IMS transaction does use the IMS message queue. The 32 KB message limit applies to synchronous program switch requests.

Before you run the IMS application that issues this call:

- The OTMA descriptor for the outbound destination routing information must be already defined.
- If the ICAL request is for synchronous callout, the external application or server that the IMS application is calling out to must be configured to listen for callout messages with the IMS OTMA RESUME TPIPE function. If the RESUME TPIPE is not set up before the ICAL call times out, a timeout error is returned to the IMS application.
- If the ICAL request is for synchronous program switch, the target is an IMS transaction that is defined with the TRANSACT macro or the equivalent type-2 commands CREATE TRAN and UPDATE TRAN. The transaction must be started.
- For a synchronous program switch request in a shared queues environment, all of the IMS systems in the same shared queues group must have a MINVERS value of 13.1.

When the synchronous callout timeout value is specified in both the OTMA destination descriptor and the DL/I ICAL call, IMS uses the lower value of the two.

For a synchronous program switch, the target transaction can be in the same IMS system, in an IMS that is accessible through shared queues, or in a remote IMS that is accessible with MSC. The synchronous program switch request is queued as an OTMA transaction, but OTMA is not required.

The target application of a synchronous program switch can issue an additional synchronous program switch request before returning to the original application program. You can chain together any number of synchronous program switch requests. However, consider the timeout value for each ICAL call when making nested synchronous program switch requests. Also, there must be an IMS dependent region available for each of the target transactions to be scheduled. Lastly, consider that a multi-switch program flow can hold database locks until the entire sequence of switches is resolved. Two or more applications in the same synchronous program switch chain can encounter database locking contention with each other.

If the ICAL call for a synchronous program switch request times out, or if more than one response is returned after the first one, IMS treats further responses as late messages. The default response to a late message is to dequeue it. If you want to retain late messages, you can configure a tpipe in the OTMA destination descriptor for request to hold the late responses, or you can code a DFSMSCE0 exit routine to reroute them.

Synchronous program switch requests made from Fast Path regions do not support late response messages. Any late response message is discarded, including subsequent redundant responses.

If a late response message for a synchronous program switch request is routed to an OTMA client, but the original request was not initiated from an OTMA client, you must use the DFSYIOE0 exit routine to re-build the default 1 KB OTMA message user data prefix for the response message.

If the destination descriptor for a synchronous program switch request is configured to queue late messages to a tpipe or reroute them with a DFSMSCE0 user exit routine, OTMA transaction expiration checking at the application GU time is disabled for the message.

Depending on the transaction security specifications (TRN=Y), the IMS region that is running the application that issues an ICAL request calls RACF and the DFSCTRN0 user exit to determine if the user is authorized to issue the ICAL call. For APPC or OTMA transactions, additional security specifications are checked. If the security level for APPC or OTMA is set to NONE, then RACF and the DFSCTRN0 user exit are not called even if TRN=Y is specified.

For a synchronous program switch request, IMS schedules the transaction as an OTMA transaction. The OTMA security configuration (NONE, CHECK, FULL, or PROFILE) is used even if OTMA is not active. The default security setting is FULL, which is also used if OTMA is not enabled for the IMS system.

You can change the synchronous program switch security configuration with by issuing the following command:

```
/SECURE OTMA TMEMBER DFSYICAL value
```

DFSYICAL is the dedicated synchronous program switch processing TMEMBER. It is not used for other types of requests. Replace *value* with NONE, CHECK, FULL, or PROFILE as appropriate.

When OTMA security is set to FULL for DFSYICAL, IMS always creates an ACEE in the dependent region when it is scheduled. IMS uses this ACEE if security checks are necessary.

When OTMA security is set to CHECK for DFSYICAL, IMS does not create an ACEE at scheduling time. IMS creates an ACEE in the control region if security checks are necessary.

When OTMA security is to set to NONE for DFSYICAL, no security check is performed.

### **Usage: RECEIVE subfunction**

When a SENDRECV subfunction call returns too much data to fit in the allocated response buffer (AIB return code X'0100' and reason code X'000C'), the value of the

AIBOLEN field is updated with the length of the complete response message. Expand the size of the response area and then issue an ICAL call with the RECEIVE subfunction code to retrieve the complete response message.

The complete response data for the original ICAL call is held in the IMS control region until one of the following events occurs:

- The application issues a new ICAL call with the SENDRECV subfunction code is issued
- The IMS application reaches a sync point or terminates abnormally
- The IMS application issues a ROLB or CHECKPOINT call

### Restrictions

ICAL calls for external callout have the following restrictions:

- Coordinated two-phase commit between the IMS application program and the external application program is not supported.
- An ICAL call cannot be issued from IMS in a shared-queues environment that is not connected to IMS Connect.

Synchronous program switch requests have the following restrictions:

- The OTMA Input/Output Edit exit routine (DFSYIOE0) is not called for a synchronous program switch request or response message.
- The TM and MSC Message Routing and Control exit routine (DFSMSCE0) is not called for a synchronous program switch request.
- The target transaction is not part of the RRS commit scope of the initiating application program.
- BMP and JBP applications cannot make synchronous program switch requests in a DBCTL environment.
- The target transaction has read-only access to Fast Path MSDBs.
- The target transaction cannot be an IMS conversational transaction.
- All of the participating IMS systems in a shared queues environment must have a DBRC MINVERS value of 13.1 or greater.

### Return codes and reason codes

The following table lists the return codes and reason codes for the ICAL call.

Table 30. Return codes and reason codes for the ICAL call

Return code	Reason code	Extended reason code	Description
X'0000'	X'0000'	X'0000'	Call was completed successfully. Proceed.
X'0100'	X'000C'	X'0000'	Partial output response data was returned.
			Issue a new ICAL call with the RECEIVE subfunction code and an expanded response data area to retrieve the complete response message.
X'0100'	X'000C'	X'000D'	An IMS informational or error message was returned in response to a synchronous program switch request.



Table 30. Return codes and reason codes for the ICAL call (continued)

Return code	Reason code	Extended reason code	Description
X'0100'	X'0100'	The default value is 0. If the value is non-zero, it is set by the external application.	Error message was returned in the output response data.
X'0100'	X'0100'	X'000D'	The synchronous program switch request was returned with an IMS message.
X'0100'	X'0100'	X'0004'	An IMS informational or error message was returned in response to a synchronous program switch request.
X'0100'	X'0104'	X'0004'	The request timed out. The ICAL was not sent to the external application.
X'0100'	X'0104'	X'0008'	The request timed out. The ICAL was sent, but the ACK was not received.
X'0100'	X'0104'	X'000C'	The request timed out. The ICAL was sent, but the response was not received.
X'0100'	X'0104'	X'0010'	The request timed out. The ICAL was sent, but IMS failed to process the response.
X'0100'	X'0104'	X'0020'	The request timed out. The ICAL request for synchronous program switch was sent, but no response was received.
X'0100'	X'0108'	The default value is 0. If the value is non-zero, it is set by the external application.	Request message was rejected by the external application.
X'0100'	X'010C'	X'0000'	The synchronous call was cleared by a command (such as a /STOP or /PSTOP command).
X'0100'	X'0110'	X'0000'	The request message was rejected because the specified transaction is not supported. Either the trancode was not found or the specified transaction was an IMS conversational transaction, a CPIC transaction, or an IMS command transaction.
X'0100'	X'0110'	X'0004'	The request message was rejected because the user is not authorized to issue a synchronous program switch request.
X'0100'	X'0110'	X'0005'	The request message was rejected because the tmember that IMS uses to process synchronous program switch requests (DFSYICAL) is stopped. Issue the command /START TMEMBER DFSYICAL to resolve the problem.

Table 30. Return codes and reason codes for the ICAL call (continued)

Return code	Reason code	Extended reason code	Description
X'0100'	X'0110'	X'0006'	The request message was rejected because the tpipe that IMS uses to process synchronous program switch requests (DFSTPIPE of the OTMA tmember DFSYICAL) is stopped. Issue the command /START TMEMBER DFSYICAL TPIPE DFSTPIPE to resolve the problem.
X'0100'	X'0110'	X'000D'	The request message was rejected because IMS failed to get an internal storage YTIB to process the message.
X'0100'	X'0110'	X'000E'	The request message was rejected because IMS failed to activate DFSYTIB0 to process the message.
X'0100'	X'0110'	X'0010'	The TMEMBER or TPIPE name for late response message routing is invalid because it contains invalid characters. Check the destination descriptor.
X'0100'	X'0110'	X'0011'	The TMEMBER or TPIPE name for late response message routing is missing from the destination descriptor. If either value is specified, both must be included.
X'0100'	X'0110'	X'0012'	The TMEMBER or TPIPE name for late response message routing is incorrect. Check the destination descriptor.
X'0100'	X'0110'	X'0013'	The SMEM and SYNCTP parameters are mutually exclusive.
X'0100'	X'0110'	X'0014'	The TPIPE name for late message processing is either missing or invalid in the destination descriptor.
X'0100'	X'0110'	X'0015'	The request message was rejected because the request was made in a shared queues environment with different IMS MINVERS values. The IMS systems in the shared queues group must have the MINVERS value 13.1.
X'0100'	X'0110'	X'0016'	The request is rejected due to OTMA global message flood condition. Too many OTMA message blocks (TIB) were allocated in the system.

Table 30. Return codes and reason codes for the ICAL call (continued)

Return code	Reason code	Extended reason code	Description
X'0100'	X'0110'	X'0020'	The request message was rejected because the input data length is incorrect. The length of the segment must match the LLZZ value specified on the request. The total length of all segments in the request must match the AIBOALEN value in the AIB.
X'0100'	X'0110'	X'0030'	The request message was rejected because the transaction is currently unavailable.
X'0100'	X'0110'	X'0031'	The request message was rejected because the transaction is stopped.
X'0100'	X'0110'	X'0033'	The request message was rejected because the destination name for the program switch is an RCNT.
X'0100'	X'0110'	X'0034'	The request message was rejected because the destination name for the program switch is a CNT.
X'0100'	X'0110'	X'0035'	The request message was rejected because the destination transaction can only accept a single input segment. Multiple input segments were specified for the request.
X'0100'	X'0110'	X'0036'	The request message was rejected because an IMS queue manager encountered an insert error.
X'0100'	X'0110'	X'0037'	The request message was rejected because an IMS queue manager encountered an internal error.
X'0100'	X'0110'	X'0038'	The request message was rejected because a queue overflow was detected.
X'0100'	X'0110'	X'0039'	The request message was rejected because IMS failed to process the Fast Path transaction.
X'0100'	X'0110'	X'003A'	The request message was rejected because IMS queue manager failed to update the message prefix.
X'0100'	X'0110'	X'003B'	The request message was rejected because IMS failed to enqueue the transaction.
X'0100'	X'0110'	X'0060'	The request message was rejected because the synchronous program switch was canceled before a reply was received.

Table 30. Return codes and reason codes for the ICAL call (continued)

Return code	Reason code	Extended reason code	Description
X'0100'	X'0110'	X'0061'	The request message was rejected because the target transaction does not reply to the IOPCB and does not perform a program-to-program switch. The ICAL is rejected to avoid a timeout. This rejection occurs when the REPLYCHK descriptor is set to YES for the destination transaction. If there is an asynchronous response for the ICAL, you can set REPLYCHK to NO and this ICAL is treated as valid.
X'0100'	X'0110'	X'0070'	IMS failed to process the response message for the synchronous program switch ICAL call. The length of an output message segment was greater than the 32K limit.
X'0100'	X'0110'	X'0071'	IMS failed to process the response message for the synchronous program switch ICAL call. IMS is running out of LUMP storage space to process the response message.
X'0100'	X'0110'	X'0072'	IMS failed to process the response message for the synchronous program switch ICAL call. IMS failed to allocate storage from subpool 231, which is required to process the response message.
X'0100'	X'0110'	X'0073'	IMS failed to retrieve the response message from the IMS message queue.
X'0104'	X'0210'	X'0000'	Input area length (AIBOALEN) is set to zero.
X'0104'	X'0214'	X'0000'	Output area length (AIBOAUSE) is set to zero.
X'0104'	X'0218'	X'0000'	Subfunction code is not known or invalid.
X'0104'	X'0610'	X'0000'	Request input area address parameter is missing
X'0104'	X'0614'	X'0000'	Response output area address parameter is missing.
X'0104'	X'1020'	X'0000'	Descriptor name is invalid.
X'0104'	X'1024'	X'0000'	Timeout value is invalid.


Table 30. Return codes and reason codes for the ICAL call (continued)

Return code	Reason code	Extended reason code	Description
X'0104'	X'1028'	X'0000'	The ICAL RECEIVE call was rejected because no additional response data is available. Either the additional response data from a previous ICAL SENDRECV call was already retrieved, or a subsequent ICAL SENDRECV call cleared the response buffer.
X'0108'	X'0008'	X'0000'	IMS failed to release PSTICALO (internal storage) for the ICAL call.
X'0108'	X'0010'	X'0000'	Unable to obtain private storage. The size of the input request data might be too large.
X'0108'	X'0570'	X'0000'	The ICAL RECEIVE call was rejected because the internal buffer storage at PSTICALO is invalid.
X'0108'	X'0580'	X'0004'	Unable to send the request message to the external application. IMS is shutting down.
X'0108'	X'0580'	X'0008'	Unable to send the request message to the external application. The IMS callout function is disabled.
X'0108'	X'0580'	X'000C'	Unable to send the request message to the external application. The OTMA member was not found or is inactive.
X'0108'	X'0580'	X'0010'	Unable to send the request message to the external application. The OTMA TPIPE was not found or is stopped.
X'0108'	X'0580'	X'0014'	Unable to send the request message to the external application. IMS failed to obtain storage to queue a request.
X'0108'	X'0580'	X'0018'	Unable to send the request message to the external application. IMS failed to obtain LUMP storage to process the message.
X'0108'	X'0580'	X'001C'	Unable to send the request message to the external application. IMS failed to contact OTMA to process the ICAL call.
X'0108'	X'0580'	X'0100'	IMS failed to obtain the required LUMP storage space to process the synchronous program switch request.
X'0108'	X'0580'	X'0104'	OTMA failed to process the synchronous program switch. See the associated X'67D0' log record.


Table 30. Return codes and reason codes for the ICAL call (continued)

Return code	Reason code	Extended reason code	Description
X'0108'	X'0584'	X'0004'	Unable to process the response output message from the external application. No data in the response message.
X'0108'	X'0584'	X'0008'	Unable to process the response output message from the external application. The XCF buffer length for the response message is incorrect.
X'0108'	X'0584'	X'000C'	Unable to process the response message from the external application. IMS failed to allocate storage for the response message processing.
X'0108'	X'0584'	X'0010'	Unable to process the response message from the external application. A null segment was found in a multi-segment response message.
X'0108'	X'0588'	The default value is 0. If the value is non-zero, it is set by IMS Connect.	IMS Connect failed to process the response. No response data returned.
X'0108'	X'058C'	The default value is 0. If the value is non-zero, it is set by IMS Connect.	IMS Connect failed to process the response. Complete or partial raw data from the external client application is returned.

**Related concepts:**

 OTMA descriptors (Communications and Connections)

**Related reference:**

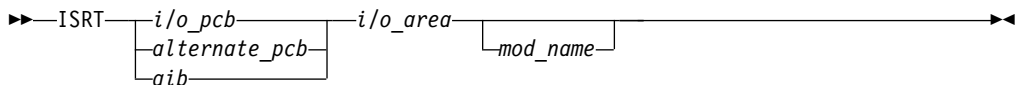
 AIB return and reason codes set by IMS (Messages and Codes)

**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*

Specifies the I/O PCB, the first PCB address in the list that is passed to the program. This parameter is an input and output parameter.

### *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**

Eye catcher. 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 TP 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.

*In the shared queues environment*

A STATUSQF can be received on an ISRT call in a shared queues environment if the MSGQ structure is full. If the MSGQ structure is full, one of the following can happen:

- If the ISRT is for a multi-segment message, STATUSQF will be received.
- If the ISRT for a multi-segment message still completes correctly (enough space) but not enough space is found to be available at PURG or CHKP time, the application will abend with ABENDU0370.
- If the ISRT is for a single segment message, STATUSQF can be received. If the program continues to insert further messages that cause all available device relative record number (DRRN) to be exhausted, IMS will fail with ABENDU0758. If the program issues a checkpoint before exhausting all available DRRN, queue buffers will be freed and the messages will be written on the log as “unresolved UOWEs.” Logs containing the original type01 and type03 log records are needed to later insert the messages in the structure if space becomes available and must not be reused. IMS will issue message DFS1994I to remind the user at every check point time.

*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 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.

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.

LL or LLLL <sup>1,2</sup>	ZZ <sup>2</sup>	II <sup>3</sup>	zz <sup>3</sup>
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



LL or LLLL <sup>1,2</sup>	ZZ <sup>2</sup>	II <sup>3</sup>	zz <sup>3</sup>
---------------------------	-----------------	-----------------	-----------------

**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).

**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.

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.

**Related reference:**

“CHNG call” on page 88

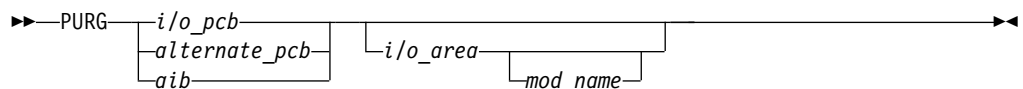
“PURG call”

**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.

**Format**



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

**Parameters**

*i/o pcb*

Specifies the I/O PCB, the first PCB address in the list that is passed to the program. This parameter is an input and output parameter.

*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**

Eye catcher. 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 TP 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 into 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.

### *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 Version 13 Exit Routines*.

### *In the shared queues environment*

A STATUSQF can be received on a PURG call in a shared queues environment if the MSGQ structure is full. If the MSGQ structure is full, one of the following can happen:

- If the PURG is for a multi-segment message, STATUSQF will be received.

- If the PURG for a multi-segment message still completes correctly (enough space) but not enough space is found to be available at PURG or CHKP time, the application will abend with ABENDU0370.

### 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 non-express 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 non-express 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. For conversational transactions, if the first ISRT is the SPA, the MOD name can either be provided on the SPA ISRT or on the first ISRT of a message segment.

This call is not supported in an IFP.

For synchronized APPC/OTMA conversations or OTMA commit-then-send (CM0) transactions with TMAMIPRG indicator set in the OTMA prefix, PURG calls on the TP PCB are ignored. The next ISRT call is processed for the next segment of the current message.

#### Related reference:

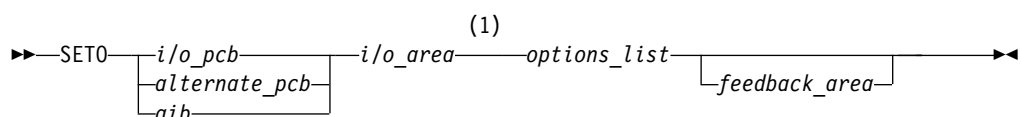
“CHNG call” on page 88

“ISRT call” on page 112

### 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
SET0	X		X		

**Parameters***i/o pcb*

Specifies the I/O PCB, the first PCB address in the list that is passed to the program. This parameter is an input and output parameter.

*alternate pcb*

Specifies the TP 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**

Eye catcher. 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 TP 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 SET0 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" and "APPC options".

The format for the *options list* is as follows:

LL or LLLL <sup>1,2</sup>	ZZ	keyword= <i>variable1</i>
Halfword length of the options string, including the 4-byte length of LLZZ or LLLLZZ.	Halfword of zero.	SETO options separated by commas.

**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 <sup>1,2</sup>	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	<i>feedback area</i>
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 SET0 call to provide print data set characteristics to the Spool API. When the SET0 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 SET0 call to prebuild the text units if they can be prebuilt with another programming technique.

**Related reading:** For more information about Spool API, see *IMS Version 13 Application Programming*.

### *In the OTMA environment*

An IMS application program that issues a SET0 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 Version 13 Exit Routines*.

Existing IMS application programs that issue SET0 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 SET0 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 SET0 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 SET0 call.

### *Advanced print function options*

The PRTO= keyword identifies the SET0 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:

<b>LL</b>	<i>outdes options</i>
Halfword length of the total OUTDES printer options, including the 2-byte length of <b>LL</b> .	Any valid combination of OUTDES printer options, separated by commas.
<b>Note:</b> For information about TSO OUTDES options, see <i>z/OS MVS Programming: Authorized Assembler Services Reference</i> . Some options depend on the release level of MVS.	

If z/OS 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 SET0 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 SET0 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 SET0 call is issued on a TP 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 Version 13 Communications and Connections*.

#### ***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 z/OS 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 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.

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.

### Related reference:

“REXXTDLI calls” on page 364



## DL/I calls for IMS TM system services

Use these DL/I calls with IMS Transaction Manager system services.

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 TP 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 the topic "Defining Application Program Elements" in *IMS Version 13 Application Programming* for the complete structure.

### System Service Call Summary

The following table 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. The following table lists the function code, its meaning, use, parameters, and in which regions it is valid. 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 the topic "Formatting DL/I Calls for Language Interfaces" in *IMS Version 13 Application Programming*.

For information on writing calls with programming language interfaces see the topic "Defining Application Program Elements" in *IMS Version 13 Application Programming*.

Table 31. 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

Table 31. Summary of system service calls (continued)

Function Code	Meaning and Use	Options	Parameters	Valid for
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)
GSCD <sup>1</sup>	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

Table 31. Summary of system service calls (continued)

Function Code	Meaning and Use	Options	Parameters	Valid for
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

**Note:**

1. GSCD is a Product-sensitive Programming Interface.

**Related reading:** DCCTL users can issue calls using GSAM database PCBs. GSAM databases are described in *IMS Version 13 Application Programming*.

**Related reference:**

- “DL/I calls for IMS DB system services” on page 35
- “DL/I calls for database management” on page 1
- “EXEC DLI commands” on page 162

**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**

▶▶ APSB *aib* ▶▶

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**

Eye catcher. 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 Version 13 Communications and Connections*.

## 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

▶▶—CHKP— $\left\{ \begin{array}{l} i/o\_pcb \\ aib \end{array} \right\}$ — $i/o\_area$ ————▶▶

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, the first PCB address in the list passed to the program, 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**

Eye catcher. 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 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.

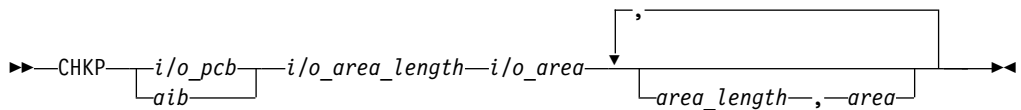
**Restrictions**

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, the first PCB address in the list passed to the program, 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**

Eye catcher. 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.

**Related reference:**

"XRST call" on page 159

**DPSB call**

The Deallocate PSB (DPSB) call frees a PSB that was allocated with the APSB call.

**Format**

►►—DPSB—*aib*—————►►

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**

Eye catcher. 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 the topic "CPI-C Driven Application Programs" in *IMS Version 13 Communications and Connections*.

**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**

►►—GMSG—*aib*—*i/o\_area*—————►►

## 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**

Eye catcher. 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 listed 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.

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 non-retrieved 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.

The following table 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 32. 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

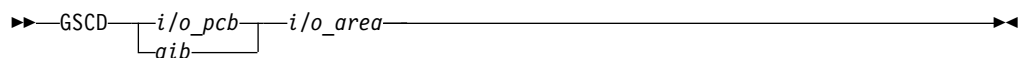
A CPI-C driven program must issue an APSB (allocate PSB) call before issuing GMSG.

## GSCD call

The Get System Contents Directory (GSCD) call retrieves the address of the IMS system contents directory (SCD) for batch programs.

**This topic contains Product-sensitive Programming Interface information.**

## Format



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

## Parameters

### *i/o pcb*

Specifies the PCB, the first PCB address in the list passed to the program, 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**

Eye catcher. 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 output parameter. For the GSCD 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 DLI or DBB batch application programs.

## 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

▶▶—ICMD—*aib*—*i/o\_area*—————▶▶

## 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**

Eye catcher. 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 *IMS Version 13 Operations and Automation*. 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 Version 13 Exit Routines*.

The following table shows, by IMS environment, the types of application programs that can issue ICMD. ICMD is also supported from a CPI-C driven application.

Table 33. ICMD 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

See *IMS Version 13 Operations and Automation* for a list of commands that can be issued using the ICMD call.

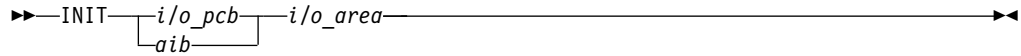
## Restrictions

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, the first PCB address in the list passed to the program. 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**

Eye catcher. 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. The I/O area of an INIT call can contain the character string "DBQUERY" or "VERSION(*dbname1=version,dbname2=version*)".

### Usage

The INIT call is valid for all IMS TM application programs.

#### *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.

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. The following tables contain sample I/O areas for the INIT call with DBQUERY.

*Table 34. INIT I/O area examples for all xxxTDLI interfaces except PLITDLI*

L	L	Z	Z	Character String
00	0B	00	00	DBQUERY

**Note:** The LL and ZZ fields are binary. The LL value X'0B' is a hexadecimal representation of decimal 11.

*Table 35. 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 pseudo-abend 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.

**bb** The data that can be accessed with this PCB can be used for all functions the PCB allows. DEDBs and MSDBs always have the bb.

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.

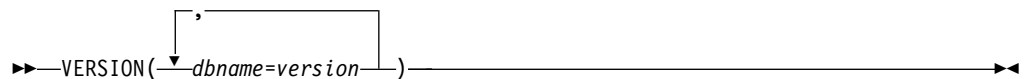
In DCCTL environments, the status code is NA.

**Specify a database version number: INIT VERSION(dbname=version)**

When database versioning is enabled, an application program can use the "VERSION" function to request a version of a database that is different from the version number that is specified for the application program on the PCB or from the default version that is returned by IMS. A version number specified on the INIT VERSION call takes precedence over all other version specifications and defaults.

When the INIT VERSION call is not issued prior to a DL/I to access a database, the version of the database that is returned to the application program is determined by the DBVER keyword of the PCB statement. If the DBVER keyword is not specified, IMS returns either the version of the database that is active in the ACB library or version 0 of the database, as determined by the DBLEVEL keyword in either the PSBGEN statement or the database section of the DFSDFxxx PROCLIB member.

In the I/O area, the VERSION function is specified by using the following format:



Each database name is specified by using alphabetic characters and can be specified only once. Specify only names of physical databases. The names of logical databases are not supported.

Each version is specified as a numeric value from 0 to 2147483647. The number that is specified must match a version number that is defined on a DBD for the named database and stored in the IMS catalog.

Calculate the size that is required for the I/O area by multiplying the number of databases that are specified in the input I/O area by 20.

For example, the following table contains a sample I/O area for the INIT VERSION call for assembler language, COBOL, C language, and Pascal. In the table, the LL value of X'3C' is the hexadecimal representation of decimal 60, the length in bytes that is required to hold the output in the I/O area when three database names are specified on input. The ZZ fields are binary.

*Table 36. INIT VERSION: Example format for ASMTDLI, CBLTDLI, CTDLI, and PASTDLI*

L	L	Z	Z	Character string
00	3C	00	00	VERSION (DBa=1,DBb=2,DBc=3)

The following table contains a sample I/O area for the INIT call with VERSION for PL/I. In the table, the LL value of X'3C' is the hexadecimal representation of decimal 60. The ZZ fields are binary.

*Table 37. INIT VERSION: Example format for PLITDLI*

L	L	L	L	Z	Z	Character string
00	00	00	3C	00	00	VERSION (DBa=1,DBb=2,DBc=3)

| **LL or LLLL**

| A 2-byte or 4-byte field that contains the total length of the I/O area. For  
| PL/I, the length of the LLLL field is considered 2 bytes even though it is a  
| 4-byte field. When you use the AIBTDLI interface, the length of the record  
| is equal to the total length of LL + ZZ + character string. For the PLITDLI  
| interface, the length of the record is equal to the total length of LLLL + ZZ  
| + required length for output, where LLLL is considered 2 bytes.

| **ZZ** A 2-byte field of binary zeros.

| **Character string**

| The function specification on input. The length that is specified in the LL  
| or LLLL is the length that is required for the output: 20 bytes for each  
| database that is specified in the input character string.

| **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 for application interfaces that use the AIB structure.

| **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 (DFSAIB) for the call. This parameter is an input and output parameter. These fields must be initialized in the AIB:

| **AIBID**

Eye catcher. 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
- ENVIRONb
- FINDbbbb
- LERUNOPT
- MSGINFOb
- PROGRAMb (Not supported with the ODBA interface)

| **AIBRSNM1**

Resource name. This 8-byte, left-justified field must contain the name of any named PCB in the PSB.



## **AIBOALEN**

I/O area length. This field must contain the length of the I/O area specified in the call list. This field is not changed by IMS.

### *i/o area*

| Specifies the data output area to use with the call. This parameter is an output  
| parameter. An I/O area is required for INQY subfunctions ENVIRONb,  
| MSGINFOb and PROGRAMb. It is not required for subfunctions DBQUERYb,  
| FINDbbbb, and LERUNOPT.

## **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.

## **Usage**

The INQY call operates in both batch and online IMS environments. IMS application programs can use the INQY call to request information regarding the output destination, the session status, the current execution environment, the availability of databases, and the PCB address, which is based on the PCB name. You must use the AIB when issuing an INQY call. Before you can issue an INQY call, initialize the fields of the AIB.

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 program receives.

The INQY call returns information to the caller's I/O area. The length of the data that is returned from the INQY call is passed back to the application program 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 information, 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 APPC, OTMA, TERMINAL, TRANSACT, and UNKNOWN.

**Related reading:** For more information about APPC and LU 6.2, see *IMS Version 13 Communications and Connections*.

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. The following tables list the output returned from the INQY null call. Refer to the notes associated with the table for further information about some of the entries.

*Table 38. INQY null data output for terminal-type destinations*

Information returned	Length in bytes	Actual value	Explanation
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 39. INQY null data output for transaction-type destinations*

Information returned	Length in bytes	Actual value	Explanation
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. <sup>1</sup>
Transaction Status	8	STARTED	The transaction can be scheduled. <sup>2</sup>
		STOPPED	The transaction cannot be scheduled. <sup>2</sup>
Destination PSB Name	8		This field gives the name of the destination PSB.
		b	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	b	The status is not available.
		ACTIVE	The MSC link session is active (remote transaction or a transaction that was rerouted to a remote IMS by the TM and Message Routing and Control user exit routine (DFSMSCE0)).
		INACTIVE	The MSC link session is inactive (remote transaction or a transaction that was rerouted to a remote IMS by the TM and Message Routing and Control user exit routine (DFSMSCE0)).
		STARTED	The program can be scheduled (local transaction).
		STOPPED	The program cannot be scheduled (local transaction).

Table 39. INQY null data output for transaction-type destinations (continued)

Information returned	Length in bytes	Actual value	Explanation
<b>Notes:</b>			
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 Destination 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 transaction was rerouted to a remote IMS by the TM and Message Routing and Control user exit routine (DFSMSCE0, the status returned is the MSNAME status.		

Table 40. INQY null data output for APPC-Type destinations

Information returned	Length in bytes	Actual value	Explanation
Destination Type	8	APPC	The destination is an LU 6.2 device.
APPC/MVS Side Information Entry Name <sup>1</sup>	8		This field provides the Side Name.
		b	The Side Name is not available.
Partner Logical Unit Name <sup>2</sup>	8		This field provides the partner LU name for the conversation.
		b	The partner LU name is not available.
Partner Mode Table Entry Name <sup>3</sup>	8		This field provides the Mode Name for the conversation.
		b	The Mode Name is not available.
User Identifier	8		This field provides the user ID.
		b	The user ID is not available.
Group Name	8		This field provides the Group Name.
		b	The Group Name is not available.
Synchronization Level <sup>4</sup>	1	C	The synchronization level is defined as CONFIRM.
		N	The synchronization level is defined as NONE.
Conversation Type <sup>5</sup>	1	B	The conversation is defined as BASIC.
		M	The conversation is defined as MAPPED.
Userid Indicator	1		The value of the Userid Indicator field indicates the contents of the user ID field. The Userid Indicator field has four possible values.
		U	The U value indicates the user's identification from the source terminal during signon.
		L	The L value indicates the LTERM name of the source terminal if signon is not active.
		P	The P value indicates the PSBNAME of the source BMP or transaction.
		O	The O value indicates some other name.

Table 40. INQY null data output for APPC-Type destinations (continued)

Information returned	Length in bytes	Actual value	Explanation
Address of TPN <sup>6</sup>	4		This is the address of the LL field of the Transaction Program Name. <sup>7</sup>
		0	The address of the Transaction Program Name is not available.

**Notes:**

1. If the call is issued against a TP PCB, the Side Name cannot be used and b 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.
2. If the call is issued against a TP 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.
3. If the call is issued against a TP PCB, the Mode Name cannot be used and b 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.
4. When the synchronization level is not available, IMS uses the default value of CONFIRM.
5. When the conversation type is not available, IMS uses the default value of MAPPED.
6. The pointer identifies a length field (LL), which contains the length of the TPN in binary, including the 2 bytes required for LL.
7. The TPN can be up to 64 bytes long.

Table 41. INQY null data output for OTMA-Type destinations

Information Returned	Length in Bytes	Actual Value	Explanation
Destination Type	8	OTMA	The destination is an OTMA client.
tpipe Name	8		This field provides the OTMA transaction pipe name.
		b	The tpipe Name is not available.
Member Name	16		This field provides the z/OS cross-system coupling facility (XCF) member name of the OTMA client.
		b	The Member Name is not available.
User Identifier	8		This field provides the User ID.
		b	The User ID is not available.
Group Name	8		This field provides the group name.
		b	The Group Name is not available.
Synchronization Level	1	S	The OTMA transaction pipe is synchronized.
		b	The OTMA transaction pipe is not synchronized.
Message Synchronization Level <sup>1</sup>	1	C	The synchronization level is defined as CONFIRM.
		N	The synchronization level is defined as NONE.

Table 41. INQY null data output for OTMA-Type destinations (continued)

Information Returned	Length in Bytes	Actual Value	Explanation
Userid Indicator	1		The value of the Userid Indicator field indicates the contents of the user ID field. The Userid Indicator field has four possible values.
		U	The U value indicates the user's identification from the source terminal during signon.
		L	The L value indicates the LTERM name of the source terminal if signon is not active.
		P	The P value indicates the PSBNAME of the source BMP or transaction.
		O	The O value indicates some other name.
Reserved for IMS	1		This field is reserved.

**Notes:**

1. When the synchronization level is not available, IMS uses the default value of CONFIRM.

Table 42. INQY null data output for unknown-type destinations

Information returned	Length in bytes	Actual value	Explanation
Destination Type	8	UNKNOWN	Unable to find destination.

The contents of the output fields vary depending on the type of PCB used for the INQY call. The following table shows how INQY output for APPC destinations varies depending on the PCB type. The PCB can be a TP PCB or an alternate PCB.

Table 43. INQY output and PCB type

Output field	TP 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
Userid Indicator	U or L or P or O	U or L or P or O	U or L or P or O
TPN Address	Address of the TPN character string	Address of the TPN character string or zero	Address of the TPN character string or zero

Table 43. INQY output and PCB type (continued)

Output field	TP PCB	Alternate PCB (Non-modifiable)	Alternate PCB (Modifiable)
TPN character string <b>Note:</b> 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 Version 13 Communications and Connections*.

### 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 INITDBQUERY call. The INQY DBQUERY call does not return information in the I/O area, but like the INIT DBQUERY call, it updates status codes in the database PCBs.

The application program is not made aware of the status of each PCB until an INQY FIND call is issued. To retrieve the status for a database, you must pass the DB PCB for that database in the INQY FIND call.

In addition to the INIT DBQUERY status codes, the INQY DBQUERY call returns these status codes in the I/O PCB:

- bb** 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 that is 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 the call has not been issued. 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 been issued. 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.

- bb** The data that can be accessed with this PCB can be used for all functions the PCB allows. DEDBs and MSDBs always have the bb.

### 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 IOPCBbbb. This includes the IMS identifier, release, region, and region type.

The INQY ENVIRON call returns character-string data. The output is left justified and padded with blanks on the right.

**Recommendations:** To account for expansion in the length of the reply data, specify an I/O area length of 512 bytes.

To reference the field that contains the recovery token or the application parameter string, code your application programs to locate the field by using the address of the field that is returned in the data output of the INQY ENVIRON call. This is the only valid programming technique to reference the recovery token field and the application parameter string field. No other programming technique should be used to reference these fields.

The recovery token or the application parameter string are optional and therefore are not always returned. If they are not returned, the value in the address field is zero.

For more information about the recovery token and application parameter fields, see note 2 after the following table.

The following table lists the output that is returned from the INQY ENVIRON call. Included with the information returned is the outputs byte length, the actual value, and an explanation.

*Table 44. INQY ENVIRON data output*

Information returned	Length in bytes	Actual value	Explanation
IMS Identifier	8		Provides the identifier from the execution 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)

Table 44. INQY ENVIRON data output (continued)

Information returned	Length in bytes	Actual value	Explanation
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.
		JBP	Indicates that the Java batch processing region is active.
		JMP	Indicates that the Java message processing region is active.
		MPP	Indicates that the Message Processing region is active.
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.
		b	Indicates that no associated transaction exists.
User Identifier <sup>1</sup>	8		Provides the user ID.
		b	Indicates that the user ID is unavailable.
Group Name	8		Provides the group name.
		b	Indicates that the group name is unavailable.
Status Group Indicator	4	A	Indicates an INIT STATUS GROUPE call is issued.
		B	Indicates an INIT STATUS GROUPB call is issued.
		b	Indicates that a status group is not initialized.
Address of Recovery Token <sup>2</sup>	4		Provides the address of the LL field, followed by the recovery token.
		0	Indicates that the recovery token is not available.
Address of the Application Parameter String <sup>2</sup>	4		Provides the address of the LL field, followed by the application program parameter string.
		0	Indicates that the APARM= parameter is not coded in the execution parameters of the dependent region JCL.
Shared Queues Indicator	4		Indicates IMS is not using Shared Queues.
		SHRQ	Indicates IMS is using Shared Queues.
User ID of Address Space	8		User ID of dependent address space.
User ID Indicator	1		Contains one of the following possible values to indicate the contents of the userid field:
		U	Indicates the user's identification from the source terminal during sign-on.
		L	Indicates the LTERM name of the source terminal in sign-on is not active.
		P	Indicates the PSBNAME of the source BMP or transaction.
		O	Indicates some other name.



Table 44. INQY ENVIRON data output (continued)

Information returned	Length in bytes	Actual value	Explanation
z/OS Resource Recovery Services (RRS) Indicator	3	b	Indicates that IMS has not expressed interest in the UR with RRS. Therefore, the application should refrain from performing any work that causes RRS to become the syncpoint manager for the UR because IMS will not be involved in the commit scope. For example, the application should not issue any outbound protected conversations.
		RRS	Indicates IMS has expressed interest in the UR with RRS. Therefore, IMS will be involved in the commit scope if RRS is the syncpoint manager for the UR.
IMS catalog enablement indicator	7	b	Indicates that the IMS catalog is not enabled in the DFSDFxxx PROCLIB member.  For information about setting up and enabling an IMS catalog, see IMS catalog definition and tailoring (System Definition).  For information about enabling the IMS catalog in the DFSDFxxx PROCLIB member, see DFSDFxxx member of the IMS PROCLIB data set (System Definition).
		CATALOG	Indicates that the IMS catalog is enabled. Database and application metadata is available in IMS.

**Notes:**

- 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 that is currently scheduled into the BMP region.
  - For message-driven BMP regions that have completed a successful GU call and for any MPP region, the PSTUSID field is derived which is usually the input terminal's RACF ID. If the terminal has not signed on to RACF, the ID is the input terminal's LTERM.
- The pointer is an address that identifies a length field (LL) which contains the length of the recovery token or application program parameter string in binary, including the two bytes required for LL. Use this pointer to set up addressability of the AIB between releases in a batch program.

### Querying the input message information: INQY MSGINFO

To obtain information regarding the current input message, use the INQY call with the MSGINFO subfunction. The only valid PCB name that can be passed in the AIBRSNM1 field is IOPCBbbb. The output returns the version number and the output fields for the message information. The INQY MSGINFO call returns the response in the I/O area.

The following table lists the output that is returned from the INQY MSGINFO call. Included with the information returned is the byte length, the actual value, and an explanation of the output.

Table 45. INQY MSGINFO data output

Information returned	Length in bytes	Actual value	Explanation
Version number	4	1	Output response version 1.

Table 45. INQY MSGINFO data output (continued)

Information returned	Length in bytes	Actual value	Explanation
Origin IMSID	8		The IMS identifier from which the input message originated.
Reserved for IMS	68		This field is reserved for future output expansion.

### 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 names that can be passed in AIBRSNM1 are IOPCBbbb or the name of the alternate PCB (TP 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 for LE overrides: INQY LERUNOPT

When the LERUNOPT call is issued with the LERUNOPT subfunction, IMS determines if LE overrides are allowed based on the LEOPT system parameter. The LE override parameters are defined to IMS through the UPDATE LE command. IMS checks to see if there are any overrides applicable to the caller based on the specific combinations of transaction name, lterm name, userid, or program name in the callers environment. IMS will return the address of the string to the caller if an override parameter is found. The LE overrides are used by the IMS supplied CEEBXITA exit, DFSBXITA, to allow dynamic overrides for LE runtime parameters.

The call string must contain the function code and the AIB address. The I/O area is not a required parameter and will be ignored if specified. The only valid PCB name that can be passed in AIBRSNM1 is IOPCB. The AIBOALEN and AIBOAUASE fields are not used.

The rules for matching an entry that results in it being returned on a DL/I INQY LERUNOPT call are:

- An MPP or JMP region uses transaction name, lterm, userid, and program to match with each entry.
- An IFB, JBP, or non-message driven BMP uses program name to match with each entry. If an entry has a defined filter for transaction name, lterm, or userid, it does not match. Message driven BMPs also use transaction name.
- The entries are scanned to find the entry with the most filter matches. The first entry in the list with the most exact filter matches is selected. The scan stops with an entry found with all of the filters matching the entry.

**Note:** Searching table entries may cause user confusion because of the way entries are built and searched. For example, assume there are two entries in the table that match on the filters specified on the DL/I INQY call. The first transaction matches on transaction name and lterm name. The second entry

matches on transaction name and program name. IMS chooses the first entry because it was the first entry encountered with highest number of filter matches. If the second entry is now updated with a longer parameter string, which causes a new entry to be built, it will be added to the head of the queue. The next search would result in the entry with transaction name and program name being selected. This could result in a set of runtime options being selected that were not expected by the user.

*Environments:* The LERUNOPT subfunction can be specified from DB/DC, DBCTL, and DCCTL environments. Overrides are based on a combination of transaction name, lterm name, user ID, and program name in MPP and JMP regions. IFP, BMP, and JBP regions will have overrides based on program name. Message driven BMP regions can also use transaction name.

*Return and reason codes:* AIB return and reason codes must be checked to determine if the call has been successfully completed. AIBRSA2 is used to return the address of the parameter string if override parameters are available for the caller.

### 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. 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

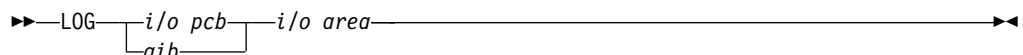
Table 46. Subfunction, PCB, and I/O area combinations for the INQY call

Subfunction	I/O PCB	Alternate PCB	DB PCB	I/O Area Required
FIND	OK	OK	OK	NO
ENVIRON	OK	NO	NO	YES
DBQUERY	OK	NO	NO	NO
LERUNOPT	OK	NO	NO	NO
PROGRAM	OK	NO	NO	YES
MSGINFO	OK	NO	NO	YES

### 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, the first PCB address in the list passed to the program, 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**

Eye catcher. This 8-byte field must contain DFSAIBbbb.

#### **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 the following tables.

Table 47. Log record formats for COBOL, PL/I, C language, Pascal, and assembler for AIBTDLI, ASMTDLI, CBLTDLI, CEETDLI, CTDLI, and PASTDLI interfaces

Field Name	Field Length
LL	2
ZZ	2
C	1
Text	Variable

Table 48. Log record formats for COBOL, PL/I, C language, Pascal, and assembler for PLITDLI interface

Field Name	Field Length
LLLL	4
ZZ	2
C	1
Text	Variable

The fields must be as follows:

#### **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

►►—RCMD—*aib*—*i/o area*—►►

## 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

Eye catcher. 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 Version 13 Exit Routines*.

RCMD is also supported from a CPI-C driven application program.

*Table 49. 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.

## 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, the first PCB address in the list passed to the program. 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**

Eye catcher. 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.


The following processing considerations apply to modified message-driven IMS applications issuing the IMS ROLB call that can receive protected input messages from OTMA or APPC/MVS and issue outbound protected work to other z/OS Resource Recovery Services (RRS) resource managers:

- If a modified message-driven IMS application program with protected input issues a ROLB call, the ROLB call is isolated to the IMS application without affecting the entire protected unit of work. After the ROLB call is issued, the protected input message remains in process for the IMS application until a commit point is reached.
- If a modified message-driven IMS application program issues an outbound protected conversation, the outbound protected conversation is not included in the ROLB processing (that is, the outbound protected conversation is not backed out as part of the ROLB call). The modified message-driven IMS application program is responsible for explicitly cleaning up any outbound protected work to be backed out.

## Restrictions

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

### Related concepts:

 Backing out to a prior commit point: ROLL, ROLB, and ROLS calls (Application Programming)

## ROLL call

The Roll (ROLL) call backs out output messages sent by a conversational application program and terminates the conversation.

## Format

▶▶—ROLL—▶▶

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.

## Restrictions

The ROLL call cannot use the AIBTDLI interface.

### Related concepts:

➞ Backing out to a prior commit point: ROLL, ROLB, and ROLS calls (Application Programming)

➞ Administering APPC/IMS and LU 6.2 devices (Communications and Connections)

### Related reference:

➞ Non-Discardable Messages user exit (NDMX) (Exit Routines)

## 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 the topic "Backing out to a Prior Commit Point: ROLL, ROLB, and ROLS Calls" in *IMS Version 13 Application Programming*.

## 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, the first PCB address in the list passed to the program. 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**

Eye catcher. 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 DFS21711 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 Version 13 Communications and Connections*.

## 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.

### Related reference:

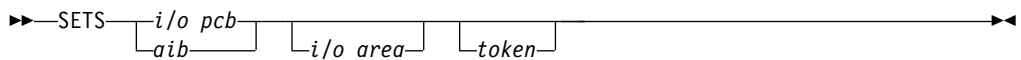
“SETS/SETU call”

## 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 is not 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, the first PCB address in the list passed to the program. 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**

Eye catcher. 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 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.

Before a ROLS call, you can specify a maximum of 255 SETS calls with the same token and still back out to the correct message level. After 255 SETS calls, the messages continue to back out, but only to the same message level as at 255th SETS call. The SETS token count resets to zero during sync point processing.

You may specify a maximum of 255 SETS calls with the same token before a ROLS call and still be able to back out to the correct message level. After 255 SETS calls, the messages will continue to back out to the same message level as at 255th SETS call. The SETS token count is reset to zero during sync point processing.

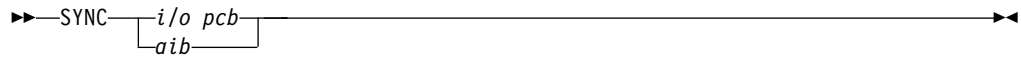
**Related reference:**

“ROLS call” on page 155

**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, the first PCB address in the list passed to the program. 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**

Eye catcher. 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.

## 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.

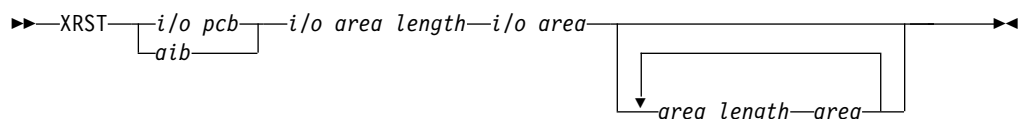
You cannot issue a SYNC call from a CPI Communications driven application program.

## 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.

## 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, the first PCB address in the list passed to the program. 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**

Eye catcher. 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 IOPCBbb.

#### **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 14-byte area in your program. This area must be either set to blanks if you are starting your program normally or, if you are 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 want 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 parameter 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:
  - III 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 a DL/I batch region, the IMSLOGR DD statement that defines the log data set must be supplied in the JCL. IMS reads these data sets and searches for the checkpoint records that have the ID that was specified.

However, if the program being restarted is in a BMP region and all of the following conditions are met, an IMSLOGR DD statement is not required:

- The BMP program is restarted with CKPTID=LAST.
- The BMP program is restarted on the same IMS system with the same job name, same PSB, and same program name that was used when it abended.
- IMS has not been cold-started since the BMP program abended.

- The checkpoint records that are needed to restart the program are on an OLDS data set that has not been archived and reused since the time of the abend, or the SLDSREAD logger function is active in IMS.

If any of the preceding conditions are not met, you must supply an IMSLOGR DD statement that points to the data set that contains the required checkpoint records.

If an IMSLOGR DD statement is supplied, it must contain the required checkpoint log records. IMS does not automatically locate and retrieve checkpoint records for a BMP if an IMSLOGR DD statement is present. Only the IMSLOGR DD data set is searched and, if the record is not found, the BMP program terminates with abend U0102.

**Note:** A DD DUMMY statement is permissible for an IMSLOGR DD statement and is treated as if no IMSLOGR DD statement was supplied.

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 (IIIIDDHHMMSST).

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.

#### Related reference:

“CHKP (symbolic) call” on page 127

---

## EXEC DLI commands

The EXEC DLI commands are the only commands that are allowed for EXEC DLI. These commands can be used to read and update DL/I databases with a batch program, a BMP region (running DBCTL or DB/DC), or a CICS program using DBCTL.

### System service commands

The following system service commands require that you first issue the SCHD command with the SYSSERVE keyword:

- ACCEPT command
- DEQ command
- LOG command
- QUERY command
- REFRESH command
- ROLS command



- SETS command
- SETU command
- STAT command

The following system service commands are valid in batch or BMP regions or programs without first issuing the SCHK command with the SYSSERVE keyword:

- CHKP command
- ROLB command
- ROLL command
- SYMCHKP command
- XRST command

The following system service commands are valid in an online CICS program using DBCTL:

- ACCEPT
- DEQ
- LOG
- QUERY
- REFRESH
- ROLS
- SETS
- STAT

To issue system service commands, the input/output PCB (I/O PCB) is required.

The examples in the following topics use the PL/I delimiter. Code the commands in free form: Where keywords, operands, and parameters are shown separated by commas, no blanks can appear immediately before or after the comma. Where keywords, operands, and parameters are shown separated by blanks, you can include as many blanks as you want. The format of the commands is the same for users of COBOL, PL/I, or assembler language.

**Related reference:**

“DL/I calls for IMS TM system services” on page 123

“DL/I calls for IMS DB system services” on page 35

“DL/I calls for database management” on page 1

“DL/I calls for transaction management” on page 81

 PCBs and PSB (Application Programming)

## Summary of EXEC DLI commands

A summary of all the EXEC DLI commands is provided in the following table.

The table lists the EXEC DLI commands and specifies if they are valid in the Batch, Batch-Oriented BMP, or CICS with DBCTL environment.

Table 50. Summary of EXEC DLI commands.

Request Type	Program Characteristics		
	Batch	Batch- Oriented BMP	CICS with DBCTL <sup>1</sup>
ACCEPT command <sup>4</sup>	Yes	Yes	Yes

Table 50. Summary of EXEC DLI commands (continued).

Request Type	Program Characteristics		
	Batch	Batch- Oriented BMP	CICS with DBCTL <sup>1</sup>
CHKP command <sup>4</sup>	Yes	Yes	No
DEQ command <sup>4</sup>	Yes	Yes	Yes
DLET command <sup>4</sup>	Yes	Yes	Yes
Get commands (GU, GHU, GN, GHN, GNP, GHNP) <sup>4</sup>	Yes	Yes	Yes
GMSG command <sup>5</sup>	No	Yes	Yes
ICMD command <sup>5</sup>	No	Yes	Yes
ISRT command <sup>5</sup>	Yes	Yes	Yes
LOAD command	Yes	No	No
LOG command <sup>4</sup>	Yes	Yes	Yes
POS command <sup>4</sup>	No	Yes	Yes
QUERY command <sup>4</sup>	Yes	Yes	Yes
RCMD command <sup>5</sup>	No	Yes	Yes
REFRESH command <sup>4</sup>	Yes	Yes	Yes
REPL command <sup>4</sup>	Yes	Yes	Yes
RETRIEVE command	Yes	Yes	No
ROLB command	Yes	Yes	No
ROLL command	Yes	Yes	No
ROLS command <sup>2,4</sup>	Yes	Yes	Yes
SCHD command	No	No	Yes
SETS command <sup>2,4</sup>	Yes	Yes	Yes
SETU command	Yes	Yes	No
STAT command <sup>2,4</sup>	Yes	Yes	Yes
SYMCHKP command	Yes	Yes	No
TERM command	No	No	Yes
XRST command	Yes	Yes	No

**Notes:**

1. In a CICS remote DL/I environment, commands in the CICS with DBCTL column are supported if you are shipping a function to a remote CICS that uses DBCTL.
2. ROLS and SETS commands are not valid when the PSB contains a DEDB.
3. STAT is a Product-sensitive Programming Interface.
4. These commands are supported in the AIB format.
5. These commands are described in the AOI documentation.

**Related concepts:**

 [IMS Automated Operator Interface \(AOI\) \(Operations and Automation\)](#)

## ACCEPT command

The Accept (ACCEPT) command is used to tell IMS to return a status code to your program, rather than abend the transaction.

## Format

►► EXEC—DLI—ACCEPT STATUSGROUP('A')  
                                  └─ACCEPT STATUSGROUP('B')—┘

## Options

### STATUSGROUP('A')

Informs IMS that the application is prepared to accept status codes regarding unavailability. IMS then returns a status code instead of pseudoabending if a call issued later requires access to unavailable data.

This is a required option.

### STATUSGROUP('B')

Informs IMS that the application is prepared to accept status codes regarding unavailability and deadlock occurrence. IMS returns a status code instead of pseudoabending if a call issued later requires access to unavailable data or deadlock occurrence.

## Usage

Use the ACCEPT command to tell IMS to return a status code instead of abending the program. These status codes result because PSB scheduling completed without all of the referenced databases being available.

## Example

```
EXEC DLI ACCEPT STATUSGROUP('A');
```

This example shows how to specify the ACCEPT command.

## CHKP command

The Checkpoint (CHKP) command is used to issue a basic checkpoint and to end a logical unit of work. You cannot use this command in a CICS program.

## Format

►► EXEC—DLI—CHECKPOINT—ID(*area*)  
                                  └─CHKP—┘                  └─ID('literal')—┘

## Options

### ID(*area*)

Contains the checkpoint ID. Specifies the name of an area in your program containing the checkpoint ID. The area pointed to is eight bytes. If you are using PL/I, specify this option as a pointer to a major structure, an array, or a character string.

### ID('literal')

'literal' is an 8-byte checkpoint ID, enclosed in quotation marks. In CHKP commands the area pointed to is 8 bytes long.

## Usage

The two kinds of commands that allow you to make checkpoints are: the CHKP, or basic Checkpoint command, and the SYMCHKP, or Symbolic Checkpoint command.

Batch programs can use either the symbolic or the basic command.

Both checkpoint commands make it possible for you to commit your program's changes to the database and to establish places from which the program can be restarted, should it terminate abnormally.

You must not use the `CHKPT=EOV` parameter on any `DD` statement to take an IMS checkpoint.

Both commands cause a loss of database position at the time the command is issued. Position must be reestablished by a `GU` command or other method of establishing position.

It is not possible to re-establish position in the midst of nonunique keys or nonkeyed segments.

You can issue the basic `CHKP` command to commit your program's changes to the database and establish places from which your program can be restarted. When you issue a basic `CHKP` command, you must provide the code for restarting your program.

When you issue a `CHKP` command, you specify the ID for the checkpoint. You can supply either the name of a data area in your program that contains the ID, or you can supply the actual ID, enclosed in single quotes.

### Examples

```
EXEC DLI CHKP ID(chkpid);  
EXEC DLI CHKP ID('CHKP0007');
```

### *Explanation*

These examples show how to specify the `CHKP` command.

### Restrictions

Restrictions for the `CHKP` command:

- You cannot use this command in a CICS program.
- You must first define an I/O PCB for your program before you can use the `CHKP` command.
- You cannot reestablish position in the midst of nonunique keys or nonkeyed segments.

## DEQ command

The Dequeue (`DEQ`) command is used to release a segment that is retrieved with the `LOCKCLASS` option.

### Format

```
►►—EXEC—DLI—DEQ—LOCKCLASS(data_value)—◄◄
```

### Option

#### **LOCKCLASS(*data\_value*)**

Specifies that you want to release the lock that is being held as the result of an

earlier GU, GN, or GNP command that had a LOCKCLASS option with the same data\_value. Data\_value must be a 1-byte alphabetic character in the range of B to J.

For full function, specify the LOCKCLASS option followed by the lock class of that segment (for example, LOCKCLASS('B')). If the option is not followed by a letter (B-J), EXECDLI sets a status code of GL and initiates an ABENDU1041.

DEQ commands are not supported for Fast Path.

## Usage

Use the DEQ command to release locks on segments that were retrieved using the LOCKCLASS option. Using LOCKCLASS on Get commands allows you to reserve segments for exclusive use by your transaction. No other transaction is allowed to update these reserved segments until either your transaction reaches a sync point or the DEQ command has been issued, thereby releasing the locks on these reserved segments. The LOCKCLASS option lets your application program leave these segments and retrieve them later without any changes having been added.

## Example

Your program can use the LOCKCLASS option as follows:

```
EXEC DLI DEQ LOCKCLASS(data_value)
EXEC DLI GU SEGMENT(PARTX)
      SEGMENT(ITEM1) LOCKCLASS('B') INTO(PTAREA1);
EXEC DLI GU SEGMENT(PARTX)
      SEGMENT(ITEM2) LOCKCLASS('C') INTO(PTAREA2);
EXEC DLI DEQ LOCKCLASS('B');
```

### Explanation

This example shows the format of the DEQ command, where data\_value is a 1-byte alphabetic character in the range B to J. The DEQ command releases the lock that was gotten and held with a LOCKCLASS of 'B' for the PARTX segment as a result of the first GU. The lock that was gotten with a LOCKCLASS of 'C' on the PARTX segment during the second GU remains held.

## Restriction

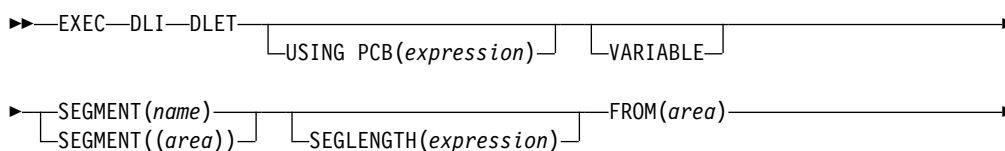
Restrictions for the DEQ command:

- To use this command you must first define an I/O PCB for your program.

## DLET command

The Delete (DLET) command is used to remove a segment and its dependents from the database.

### Format



SETZERO(*data\_value*)

## Options

### USING PCB(*expression*)

Specifies the DB PCB you want to use for the command. Its argument can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number.

### VARIABLE

Indicates that a segment is variable-length.

### SEGMENT(*name*)

Qualifies the command, specifying the name of the segment type you want to retrieve, insert, delete, or replace.

### SEGMENT((*area*))

Is a reference to an area in your program containing the name of the segment type. You can specify an area instead of specifying the name of the segment in the command.

### SEGLENGTH(*expression*)

Specifies the length of the I/O area into which the segment is retrieved. Its argument can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number. (It is required in COBOL programs for any SEGMENT level that specifies the INTO or FROM option.)

**Requirement:** The value specified for SEGLENGTH must be greater than or equal to the length of the longest segment that can be processed by this call.

### FROM(*area*)

Specifies an area containing the segment to be added, replaced, or deleted. Use FROM to insert one or more segments with one command.

### SETZERO(*data\_value*)

Specifies setting a subset pointer to zero.

## Usage

You use the DLET command to delete a segment and its dependents from the database. You must first retrieve segments you want to delete, just as if you were replacing segments. The DLET command deletes the retrieved segment *and* its dependents, if any, from the database.

## Example

“Evelyn Parker has moved away from this area. Her patient number is 10450. Delete her record from the database.”

### *Explanation*

You want to delete all the information about Evelyn Parker from the database. To do this, you must delete the PATIENT segment. When you do this, DL/I deletes all the dependents of that segment. This is exactly what you want DL/I to do—there is no reason to keep such segments as ILLNESS and TREATMNT for Evelyn Parker if she is no longer one of the clinic's patients.

Before you can delete the patient segment, you have to retrieve it:

```
EXEC DLI GU
    SEGMENT(PATIENT) INTO(PATAREA) WHERE (PATNO=PATNO1);
```

To delete this patient's database record, you issue a DLET command and use the FROM option to give the name of the I/O area that contains the segment you want deleted:

```
EXEC DLI DLET SEGMENT(PATIENT) FROM(PATAREA);
```

When you issue this command, the PATIENT segment, and its dependents—the ILLNESS, TREATMNT, BILLING, PAYMENT, and HOUSHOLD segments—are deleted.

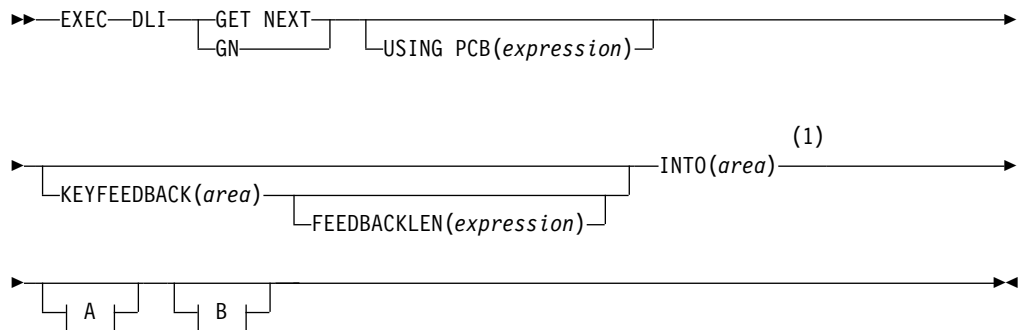
## Restrictions

You cannot issue any commands using the same PCB between the retrieval command and the DLET command, and you can issue only one DLET command for each GET command.

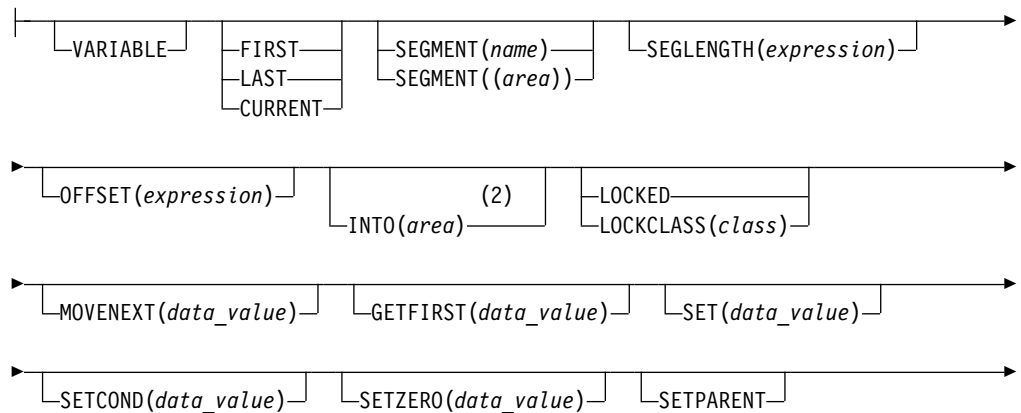
## GN command

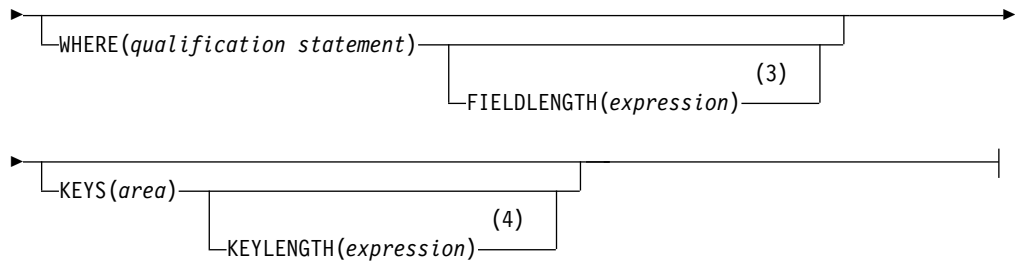
The Get Next (GN) command is used to retrieve segments sequentially.

### Format

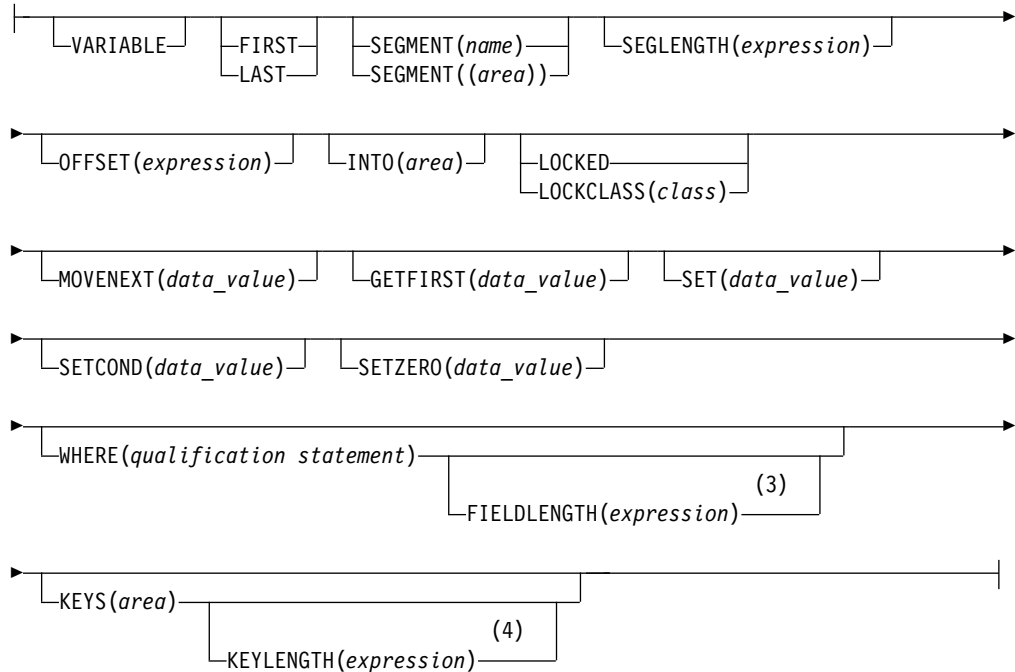


### A For each parent segment (optional):





**B For the object segment (optional):**



**Notes:**

- 1 If you leave out the SEGMENT option, specify the INTO option as shown.
- 2 Specify INTO on parent segments for a path command.
- 3 If you use multiple qualification statements, specify a length for each, using FIELDLENGTH. For example: **FIELDLENGTH(24,8)**
- 4 You can use either the KEYS option or the WHERE option, but not both on one segment level.

**Options**

**USING PCB(expression)**

Specifies the DB PCB you want to use for the command. Its argument can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number.

**KEYFEEDBACK(area)**

Specifies an area into which the concatenated key for the segment is placed. If the area is not long enough, the key is truncated.

**FEEDBACKLEN(expression)**

Specifies the length of the key feedback area into which you want the



concatenated key retrieved. Its argument can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number. (It is required in COBOL programs and optional in PL/I and assembler language programs.)

**INTO(area)**

Specifies an area into which the segment is read.

**VARIABLE**

Indicates that a segment is variable-length.

**FIRST**

Specifies that you want to retrieve the first segment occurrence of a segment type, or that you want to insert a segment as the first occurrence.

**LAST**

Specifies that you want to retrieve the last segment occurrence of a segment type, or that you want to insert a segment as the last segment occurrence.

**CURRENT**

Qualifies the command, and specifies that you want to use the level of and levels above the current position as qualifications for this segment.

**SEGMENT(name), SEGMENT((area))**

Qualifies the command, specifying the name of the segment type or the area of your program containing the name of the segment type that you want to retrieve.

You can have as many levels of qualification for a GN command as there are levels in the database's hierarchy. Using fully qualified commands with the WHERE or KEYS option clearly identifies the hierarchical path and the segment you want, and is useful in documenting the command. However, you do not need to qualify a GN command, because you can specify a GN command without the SEGMENT option.

Once you have established position in the database record, issuing a GN command without a SEGMENT option retrieves the next segment occurrence in sequential order.

If you specify a SEGMENT option without a KEYS or WHERE option, IMS DB retrieves the first occurrence of that segment type it encounters by searching forward from current position. With an unqualified GN command, the segment type you retrieve might not be the one you expected, so you should specify an I/O area large enough to contain the largest segment your program has access to. (After successfully issuing a retrieval command, you can find out from the DIB the segment type retrieved.)

If you fully qualify your command with a WHERE or KEYS option, you would retrieve the next segment in sequential order, as described by the options.

Including the WHERE or KEYS options for parent segments defines the segment occurrences that are to be part of the path to the segment you want retrieved. Omitting the SEGMENT option for a level, or including only the SEGMENT option without a WHERE option, indicates that any path to the option satisfies the command. DL/I uses only the qualified parent segments and the lowest-level SEGMENT option to satisfy the command. DL/I does not assume a qualification for the missing level.

**SEGLNGTH(expression)**

Specifies the length of the I/O area into which the segment is retrieved. Its argument can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program

containing a number. (It is required in COBOL programs for any SEGMENT level that specifies the INTO or FROM option.)

**Requirement:** The value specified for SEGLENGTH must be greater than or equal to the length of the longest segment that can be processed by this call.

**OFFSET(expression)**

Specifies the offset to the destination parent. It can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number. Use OFFSET when you process concatenated segments in logical relationships. OFFSET is required whenever the destination parent is a variable-length segment.

**LOCKED**

Specifies that you want to retrieve a segment for the exclusive use of your program, until a checkpoint or sync point is reached. This option performs the same function as the Q command code, and it applies to both Fast Path and full function. A 1-byte alphabetic character of 'A' is automatically appended as the class for the Q command code.

**LOCKCLASS(class)**

Specifies that you want to retrieve a segment for the exclusive use of your program until a DEQ command is issued or until a checkpoint or sync point is reached. (DEQ commands are not supported for Fast Path.) Class is a 1-byte alphabetic character (B-J), representing the lock class of the retrieved segment.

For full-function code, the LOCKCLASS option followed by a letter (B-J) designates the class of the lock for the segment. An example is LOCKCLASS('B'). If LOCKCLASS is not followed by a letter in the range B to J, then EXECCLI sets a status code of GL and initiates an ABENDU1041.

Fast Path does not support LOCKCLASS but, for consistency between full function and Fast Path, you must specify LOCKCLASS('x'), where x is a letter in the range B to J. An example is LOCKCLASS('B'). If LOCKCLASS is not followed by a letter in the range B to J, then EXECCLI sets a status code of GL and initiates an ABENDU1041.

**MOVENEXT(data\_value)**

Specifies a subset pointer to be moved to the next segment occurrence after your current segment.

**GETFIRST(data\_value)**

Specifies that you want the search to start with the first segment occurrence in a subset.

**SET(data\_value)**

Specifies unconditionally setting a subset pointer to the current segment.

**SETCOND(data\_value)**

Specifies conditionally setting a subset pointer to the current segment.

**SETZERO(data\_value)**

Specifies setting a subset pointer to zero.

**SETPARENT**

Sets parentage at the level you want.

**FIELDLENGTH(expression)**

Specifies the length of the field value in a WHERE option.

**KEYLENGTH(expression)**

Specifies the length of the concatenated key when you use the KEYS option. It can be any expression in the host language that converts to the integer data

type; if it is a variable, it must be declared as a binary halfword value. For IBM COBOL for z/OS & VM (or VS COBOL II), PL/I, or assembler language, KEYLENGTH is optional. For COBOL programs that are not compiled with the IBM COBOL for z/OS & VM (or the VS COBOL II) compiler, you must specify KEYLENGTH with the KEYS option.

#### **KEYS(area)**

Qualifies the command with the segment's concatenated key. You can use either KEYS or WHERE for a segment level, but not both.

“Area” specifies an area in your program containing the segment's concatenated key.

#### **WHERE(qualification statement)**

Qualifies the command, specifying the segment occurrence. Its argument is one or more qualification statements, each of which compares the value of a field in a segment to a value you supply. Each qualification statement consists of:

- The name of a field in a segment
- The relational operator, which indicates how you want the two values compared
- The name of a data area in your program containing the value that is compared against the value of the field

## **Usage**

Use the GN command to sequentially retrieve segments from the database. Each time you issue a GN command, IMS DB retrieves the next segment, as described by the options you include in the command. Before issuing a GN command, you should establish position in the database record by issuing a GU command.

You do not have to use a segment option with a GN command. However, you should qualify your GN commands as much as possible with the KEYS or WHERE options after the SEGMENT option.

## **Examples**

### *Example 1*

“We need a list of all patients who have been to this clinic.”

*Explanation:* To answer this request, your program would issue a command qualified with the segment name PATIENT until DL/I returned a GB status code to the program. (GB means that DL/I reached the end of the database before being able to satisfy your command). This command looks like this:

```
EXEC DLI GN  
  SEGMENT(PATIENT) INTO(PATAREA);
```

Each time your program issued this command, the current position moves forward to the next database record.

### *Example 2*

“What are the names of the patients we have seen since the beginning of this month?”

*Explanation:* A GN command that includes one or more WHERE or KEYS options retrieves the next occurrence of the specified segment type that satisfies the

command. To answer this request, the program issues the following GN command until DL/I returned a GB status code. The example shows the command you use at the end of April, 1988 (assuming ILLDATE1 contains 198804010):

```
EXEC DLI GN
  SEGMENT(PATIENT) INTO(PATAREA)
  SEGMENT(ILLNESS) INTO(ILLAREA) WHERE(ILLDATE>=ILLDATE1);
```

**Example 3**

```
EXEC DLI GN INTO(PATAREA);
```

*Explanation:* If you just retrieved the PATIENT segment for patient 04124 and then issued this command, you retrieve the first ILLNESS segment for patient 04124.

**Restrictions**

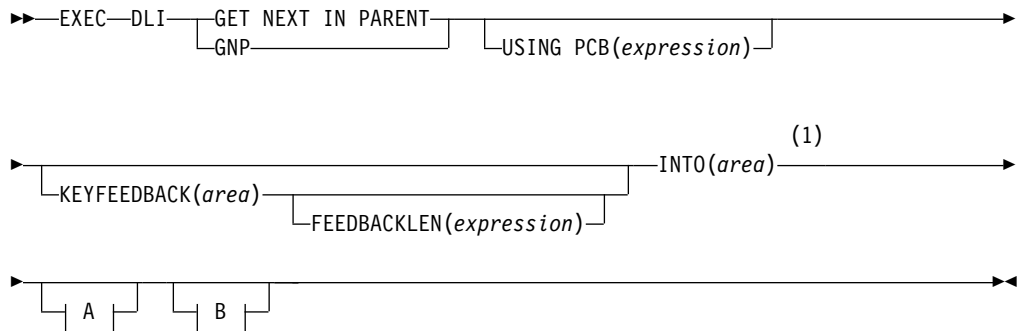
With an unqualified GN command, the retrieved segment type might not be the one expected. Therefore, specify an I/O area large enough to contain the largest segment accessible to your program.

Use either the KEYS option or the WHERE option, but not both on one segment level.

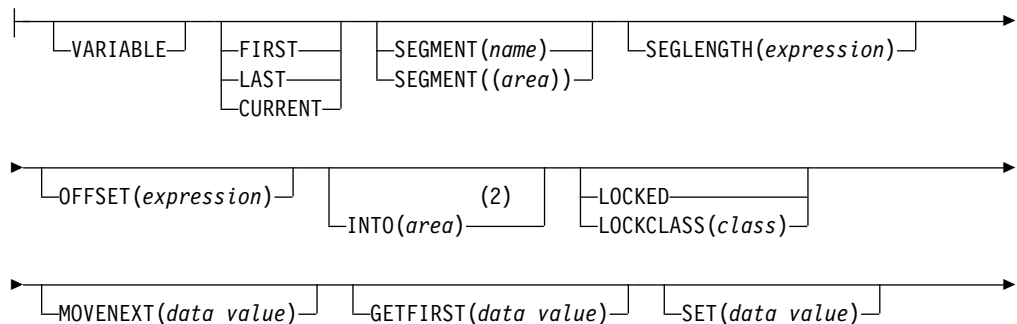
**GNP command**

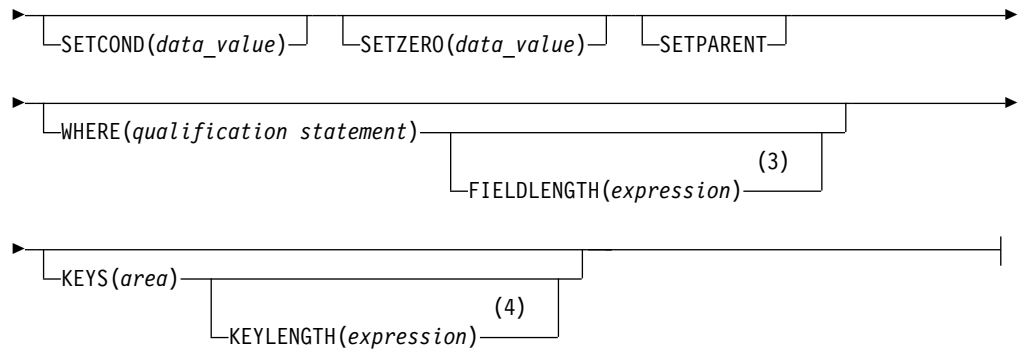
The Get Next in Parent (GNP) command is used to retrieve dependent segments sequentially.

**Format**

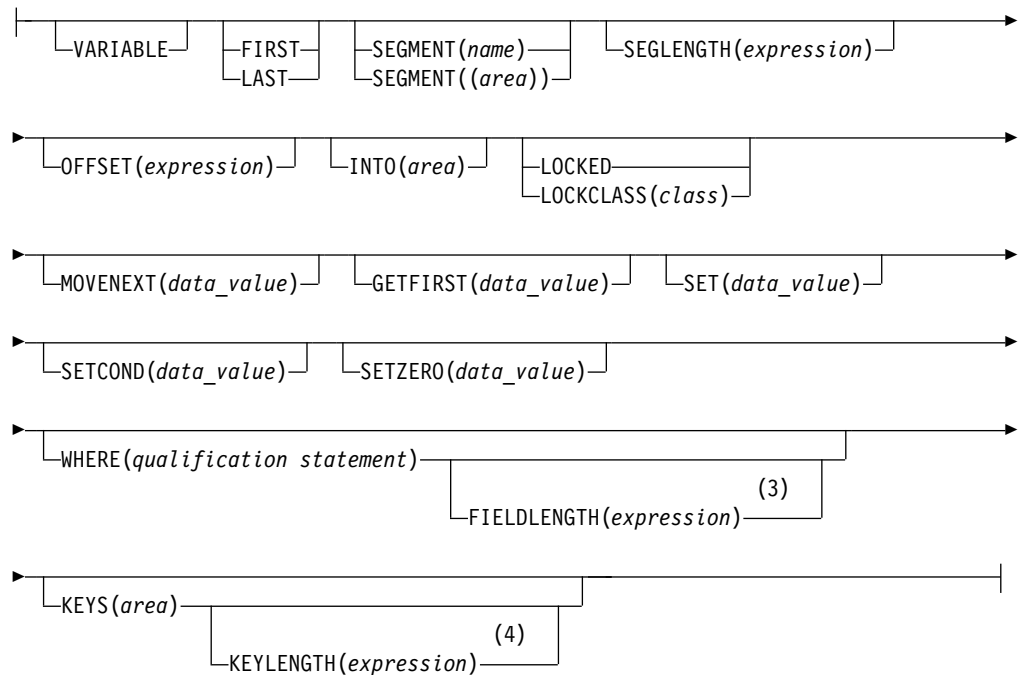


**A For each parent segment (optional):**





### B For the object segment (optional):



### Notes:

- 1 If you leave out the SEGMENT option, specify the INTO option as shown.
- 2 Specify INTO on parent segments for a path command.
- 3 If you use multiple qualification statements, specify a length for each, using FIELDLENGTH. For example: **FIELDLENGTH(24,8)**
- 4 You can use either the KEYS option or the WHERE option, but not both on one segment level.

### Options

You can qualify your GNP command by using SEGMENT and WHERE options.

If you do not qualify your command, IMS DB retrieves the next sequential segment under the established parent. If you include a SEGMENT option, IMS DB retrieves the first occurrence of that segment type that it finds by searching forward under the established parent.

You can have as many levels of qualification for a GNP command as there are levels in the database's hierarchy. However, you should not qualify your command in a way that causes DL/I to move off of the segment type you have established as a parent for the command.

**USING PCB(expression)**

Specifies the DB PCB you want to use for the command. Its argument can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number.

**KEYFEEDBACK(area)**

Specifies an area into which the concatenated key for the segment is placed. If the area is not long enough, the key is truncated. Use this to retrieve a segment's concatenated key.

**FEEDBACKLEN(expression)**

Specifies the length of the key feedback area into which you want the concatenated key retrieved. Its argument can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number. (It is required in COBOL programs and optional in PL/I and assembler language programs.)

**INTO(area)**

Specifies an area into which the segment is read. Use this to retrieve one or more segments with one command.

**VARIABLE**

Indicates that a segment is variable-length.

**FIRST**

Specifies that you want to retrieve the first segment occurrence of a segment type, or that you want to insert a segment as the first occurrence. Use this to retrieve the first segment occurrence of a segment type.

**LAST**

Specifies that you want to retrieve the last segment occurrence of a segment type, or that you want to insert a segment as the last segment occurrence. Use this to retrieve the last segment occurrence of a segment type.

**CURRENT**

Qualifies the command, and specifies that you want to use the level of and levels above the current position as qualifications for this segment. Use this to retrieve a segment based on your current position.

**SEGLENGTH(expression)**

Specifies the length of the I/O area into which the segment is retrieved. Its argument can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number. (SEGLENGTH is required in COBOL programs for any SEGMENT level that specifies the INTO or FROM option.)

**Requirement:** The value specified for SEGLENGTH must be greater than or equal to the length of the longest segment that can be processed by this call.

**OFFSET(expression)**

Specifies the offset to the destination parent. The argument can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number. Use OFFSET when you process concatenated segments in logical relationships. OFFSET is required whenever the destination parent is a variable-length segment.

**LOCKED**

Specifies that you want to retrieve a segment for the exclusive use of your program, until a checkpoint or sync point is reached. Use this to reserve a segment for the exclusive use of your program. This option performs the same function as the Q command code, and it applies to both Fast Path and full function. A 1-byte alphabetic character of 'A' is automatically appended as the class for the Q command code.

**LOCKCLASS(class)**

Specifies that you want to retrieve a segment for the exclusive use of your program until a DEQ command is issued or until a checkpoint or sync point is reached. (DEQ commands are not supported for Fast Path.) Class is a 1-byte alphabetic character (B-J), representing the lock class of the retrieved segment.

For full-function code, the LOCKCLASS option followed by a letter (B-J) designates the class of the lock for the segment. An example is LOCKCLASS('B'). If LOCKCLASS is not followed by a letter in the range B to J, then EXECDLI sets a status code of GL and initiates an ABENDU1041.

Fast Path does not support LOCKCLASS but, for consistency between full function and Fast Path, you must specify LOCKCLASS('x'), where x is a letter in the range B to J. An example is LOCKCLASS('B'). If LOCKCLASS is not followed by a letter in the range B to J, then EXECDLI sets a status code of GL and initiates an ABENDU1041.

**MOVENEXT(data\_value)**

Specifies a subset pointer to be moved to the next segment occurrence after your current segment.

**GETFIRST(data\_value)**

Specifies that you want the search to start with the first segment occurrence in a subset.

**SET(data\_value)**

Specifies unconditionally setting a subset pointer to the current segment.

**SETCOND(data\_value)**

Specifies conditionally setting a subset pointer to the current segment.

**SETZERO(data\_value)**

Specifies setting a subset pointer to zero.

**SETPARENT**

Sets parentage at the level you want.

**WHERE(qualification statement)**

Qualifies the command, specifying the segment occurrence. Its argument is one or more qualification statements, each of which compares the value of a field in a segment to a value you supply. Each qualification statement consists of:

- The name of a field in a segment
- The relational operator, which indicates how you want the two values compared
- The name of a data area in your program containing the value that is compared against the value of the field

**FIELDLENGTH(expression)**

Specifies the length of the field value in a WHERE option.

**KEYS(area)**

Qualifies the command with the segment's concatenated key. You can use either KEYS or WHERE for a segment level, but not both.

“Area” specifies an area in your program containing the segment's concatenated key.

#### **KEYLENGTH(expression)**

Specifies the length of the concatenated key when you use the KEYS option. It can be any expression in the host language that converts to the integer data type; if it is a variable, it must be declared as a binary halfword value. For IBM COBOL for z/OS & VM (or VS COBOL II), PL/I, or assembler language, KEYLENGTH is optional. For COBOL programs that are not compiled with the IBM COBOL for z/OS & VM (or VS COBOL II) compiler, you must specify KEYLENGTH with the KEYS option.

#### **SEGMENT(name), SEGMENT((area))**

Qualifies the command, specifying the name of the segment type or the area in your program containing the name of the segment type that you want to retrieve, insert, delete, or replace.

You can have as many levels of qualification for a GNP command as there are levels in the database's hierarchy. Using fully qualified commands with the WHERE or KEYS option clearly identifies the hierarchic path and the segment you want, and is useful in documenting the command. However, you do not need to qualify a GNP command at all, because you can specify a GNP command without the SEGMENT option.

Once you have established position in the database record, issuing a GNP command without a SEGMENT option retrieves the next segment occurrence in sequential order.

If you specify a SEGMENT option without a KEYS or WHERE option, IMS DB retrieves the first occurrence of that segment type it encounters by searching forward from current position. With an unqualified GNP command, the segment type you retrieve might not be the one you expected, so you should specify an I/O area large enough to contain the largest segment your program has access to. (After successfully issuing a retrieval command, you can find out from DIB the segment type retrieved.)

If you fully qualify your command with a WHERE or KEYS option, you would retrieve the next segment in sequential order, as described by the options.

Including the WHERE or KEYS options for parent segments defines the segment occurrences that are to be part of the path to the segment you want retrieved. Omitting the SEGMENT option for a level, or including only the SEGMENT option without a WHERE option, indicates that any path to the option satisfies the command. DL/I uses only the qualified parent segments and the lowest-level SEGMENT option to satisfy the command. DL/I does not assume a qualification for the missing level.

## **Usage**

The Get Next in Parent (GNP) command makes it possible to limit the search for a segment; you can retrieve only the dependents of a particular parent. You must have established parentage before issuing a GNP command.

## **Examples**

### *Example 1*

“We need the complete record for Kate Bailey. Her patient number is 09080.”



*Explanation:* To satisfy this request, you want only to retrieve the dependent segments of the patient whose patient number is 09080; you do not want to retrieve all the dependents of each patient. To do this, use the GU command to establish your position and parentage on the PATIENT segment for Kate Bailey. Then continue to issue a GNP without SEGMENT or WHERE options until DL/I returns all the dependents of that PATIENT segment. (A GE status code indicates that you have retrieved all the dependent segments.) To answer this request, your program can issue these commands:

```
EXEC DLI GU
      SEGMENT(PATIENT) INTO(PATAREA)
      WHERE (PATNO=PATN01);
EXEC DLI GNP
      INTO(ILLAREA);
```

A GNP command without SEGMENT or WHERE options retrieves the first dependent segment occurrence under the current parent. If your current position is already on a dependent of the current parent, this command retrieves the next segment occurrence under the parent.

With an unqualified GNP command, the segment type you retrieve might not be the one you expected, so you should specify an I/O area large enough to contain the largest segment your program has access to. (After successfully issuing a GNP command, you can find out from the DIB the segment type retrieved.)

### *Example 2*

“Which doctors have been prescribing acetaminophen for headaches?”

*Explanation:* A GNP command with only a SEGMENT option sequentially retrieves the dependent segments of the segment type you have specified under the established parent. Suppose that for this example, the key of ILLNESS is ILLNAME, and the key of TREATMNT is MEDICINE. You want to retrieve each TREATMNT segment where the treatment was acetaminophen. The name of the doctor who prescribed the treatment is part of the TREATMNT segment. (Assume that data area ILLNAME1 contains HEADACHE, and MEDIC1 contains ACETAMINOP). To answer this request, you can issue these commands:

```
EXEC DLI GN
      SEGMENT(ILLNESS) WHERE (ILLNAME=ILLNAME1);
EXEC DLI GNP
      SEGMENT(TREATMNT) WHERE (MEDICINE=MEDIC1);
```

To process this, your program continues issuing the GNP command until DL/I returned a GE (not found) status code, then your program retrieves the next headache segment and retrieves the TREATMNT segments for it. Your program does this until there were no more ILLNESS segments where the ILLNAME was headache.

## **Restrictions**

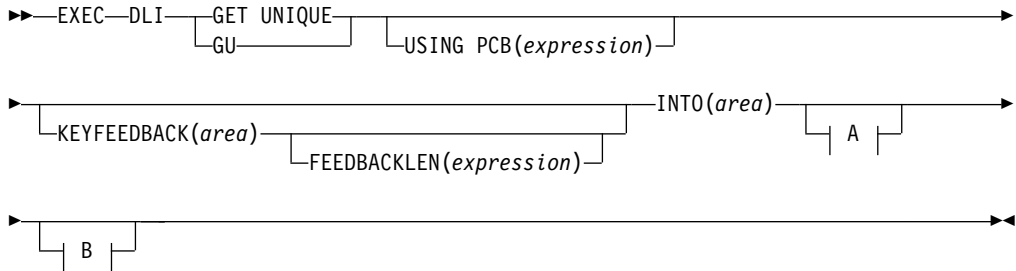
Restrictions for GNP command:

- You must have established parentage before issuing this command.
- You cannot qualify your GNP command in a way that causes DL/I to move off of the segment type you have established as the parent for the command.
- You can retrieve only the dependents of a particular parent.

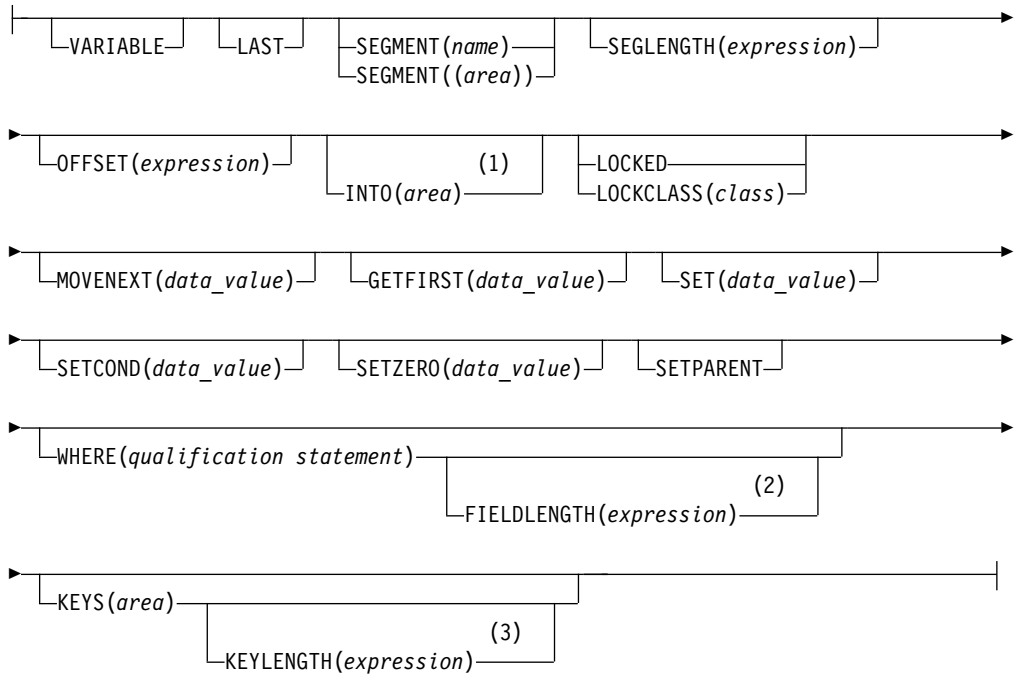
## GU command

The Get Unique (GU) command is used to directly retrieve specific segments, and to establish a starting position in the database for sequential processing.

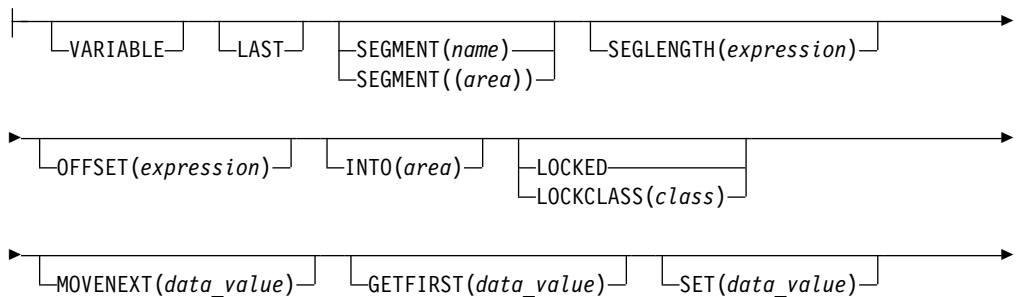
### Format

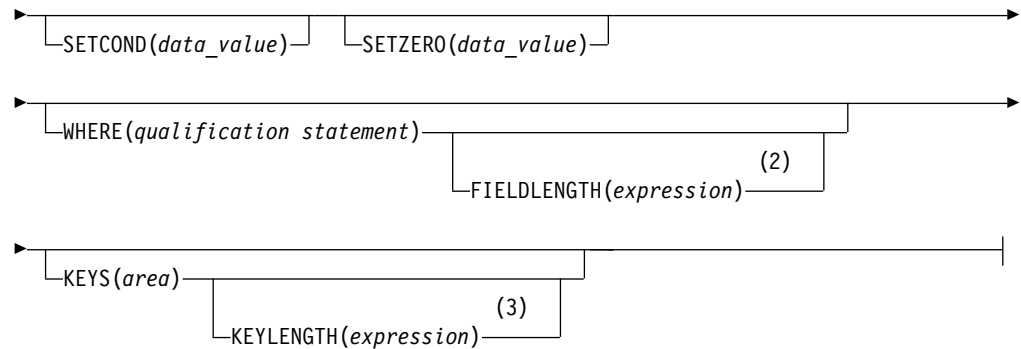


#### A:



#### B:





**Notes:**

- 1 Specify INTO on parent segments for a path command.
- 2 If you use multiple qualification statements, specify a length for each, using FIELDLENGTH. For example: **FIELDLENGTH(24,8)**
- 3 You can use either the KEYS option or the WHERE option, but not both on one segment level.

**Options**

**USING PCB(expression)**

Specifies the DB PCB you want to use for the command. Its argument can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number.

**KEYFEEDBACK(area)**

Specifies an area into which the concatenated key for the segment is placed. If the area is not long enough, the key is truncated.

**FEEDBACKLEN(expression)**

Specifies the length of the key feedback area into which you want the concatenated key retrieved. Its argument can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number. (It is required in COBOL programs and optional in PL/I and assembler language programs.)

**INTO(area)**

Specifies an area into which the segment is read.

**VARIABLE**

Indicates that a segment is variable-length.

**LAST**

Specifies that you want to retrieve the last segment occurrence of a segment type, or that you want to insert a segment as the last segment occurrence.

**SEGMENT(name), SEGMENT((area))**

Qualifies the command, specifying the name of the segment type or the area in your program containing the name of the segment type that you want to retrieve, insert, delete, or replace.

To retrieve the first occurrence of a segment type, you need only specify the SEGMENT option. You can specify as many levels of qualification as there are hierarchic levels defined by the PCB you are using.

To establish position at the beginning of the database, issue a GU command with a SEGMENT option that names the root segment type.

If you leave out SEGMENT options for one or more hierarchic levels, DL/I assumes a segment qualification for that level. The qualification that DL/I assumes depends on your current position.

- If DL/I has a position established at the missing level, DL/I uses the segment on which position is established.
- If DL/I does not have a position established at the missing level, DL/I uses the first occurrence at that level.
- If DL/I moves forward from a position established at a higher level, DL/I uses the first occurrence at the missing level that falls within the new path.
- If you leave out a SEGMENT option for the *root* level, and DL/I has position established on a root, DL/I does not move from that root when trying to satisfy the command.

You can have as many levels of qualification for a GU command as there are levels in the database's hierarchy. Using fully qualified commands with the WHERE or KEYS option clearly identifies the hierarchic path and the segment you want, and is useful in documenting the command. However, you do not need to qualify a GU command at all, because you can specify a GU command without the SEGMENT option.

If you specify a SEGMENT option without a KEYS or WHERE option, IMS DB retrieves the first occurrence of that segment type it encounters by searching forward from current position. With an unqualified GU command, the segment type you retrieve might not be the one you expected, so you should specify an I/O area large enough to contain the largest segment your program has access to. (After successfully issuing a retrieval command, you can find out from DIB the segment type retrieved.)

If you fully qualify your command with a WHERE or KEYS option, you would retrieve the next segment in sequential order, as described by the options.

Including the WHERE or KEYS options for parent segments defines the segment occurrences that are to be part of the path to the segment you want retrieved. Omitting the SEGMENT option for a level, or including only the SEGMENT option without a WHERE option, indicates that any path to the option satisfies the command. DL/I uses only the qualified parent segments and the lowest-level SEGMENT option to satisfy the command. DL/I does not assume a qualification for the missing level.

#### **SEGLength(expression)**

Specifies the length of the I/O area into which the segment is retrieved. Its argument can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number. (SEGLength is required in COBOL programs for any SEGMENT level that specifies the INTO or FROM option.)

**Requirement:** The value specified for SEGLength must be greater than or equal to the length of the longest segment that can be processed by this call.

#### **OFFSET(expression)**

Specifies the offset to the destination parent. The argument can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number. Use OFFSET when you process concatenated segments in logical relationships. OFFSET is required whenever the destination parent is a variable-length segment.

#### **LOCKED**

Specifies that you want to retrieve a segment for the exclusive use of your

program, until a checkpoint or sync point is reached. This option performs the same function as the Q command code. It applies to both Fast Path and full function. A 1-byte alphabetic character of 'A' is automatically appended as the class for the Q command code.

**LOCKCLASS(class)**

Specifies that you want to retrieve a segment for the exclusive use of your program until a DEQ command is issued or until a checkpoint or sync point is reached. (DEQ commands are not supported for Fast Path.) Class is a 1-byte alphabetic character (B-J), representing the lock class of the retrieved segment.

For full-function code, the LOCKCLASS option followed by a letter (B-J) designates the class of the lock for the segment. An example is LOCKCLASS('B'). If LOCKCLASS is not followed by a letter in the range B to J, then EXECCLI sets a status code of GL and initiates an ABENDU1041.

Fast Path does not support LOCKCLASS but, for consistency between full function and Fast Path, you must specify LOCKCLASS('x'), where x is a letter in the range B to J. An example is LOCKCLASS('B'). If LOCKCLASS is not followed by a letter in the range B to J, then EXECCLI sets a status code of GL and initiates an ABENDU1041.

**MOVENEXT(data\_value)**

Specifies a subset pointer to be moved to the next segment occurrence after your current segment.

**GETFIRST(data\_value)**

Specifies that you want the search to start with the first segment occurrence in a subset.

**SET(data\_value)**

Specifies unconditionally setting a subset pointer to the current segment.

**SETCOND(data\_value)**

Specifies conditionally setting a subset pointer to the current segment.

**SETZERO(data\_value)**

Specifies setting a subset pointer to zero.

**SETPARENT**

Sets parentage at the level you want.

**FIELDLENGTH(expression)**

Specifies the length of the field value in a WHERE option.

**KEYLENGTH(expression)**

Specifies the length of the concatenated key when you use the KEYS option. The argument can be any expression in the host language that converts to the integer data type; a variable must be declared as a binary halfword value. For IBM COBOL for z/OS & VM (or VS COBOL II), PL/I, or assembler language, KEYLENGTH is optional. For COBOL programs that are not compiled with the IBM COBOL for z/OS & VM (or VS COBOL II) compiler, you must specify KEYLENGTH with the KEYS option.

**WHERE(qualification statement)**

Use WHERE to further qualify your GU commands after using SEGMENT. If you fully qualify a GU command, you can retrieve a segment regardless of your position in the database record.

**KEYS(area)**

Use KEYS to further qualify your GU commands and specify the segment occurrence by using its concatenated key.

If you specify a SEGMENT option without a KEYS or WHERE option, IMS DB retrieves the first occurrence of that segment type it encounters by searching forward from current position. With an unqualified GU command, the segment type you retrieve might not be the one you expected, so you should specify an I/O area large enough to contain the largest segment your program has access to. (After successfully issuing a retrieval command, you can find out from DIB the segment type retrieved.)

If you fully qualify your command with a WHERE or KEYS option, you would retrieve the next segment in sequential order, as described by the options.

Including the WHERE or KEYS options for parent segments defines the segment occurrences that are to be part of the path to the segment you want retrieved. Leaving the SEGMENT option out for a level, or including only the SEGMENT option without a WHERE option, indicates that any path to the option satisfies the command. DL/I uses only the qualified parent segments and the lowest level SEGMENT option to satisfy the command. DL/I does not assume a qualification for the missing level.

## Usage

Use the GU command to retrieve specific segments from the database, or to establish a position in the database for sequential processing.

You must at least specify the SEGMENT option with a GU command to indicate the segment type you want to retrieve. (IMS DB retrieves the first occurrence of the segment you named in the SEGMENT argument.)

When you need to retrieve a specific occurrence of a segment type, you can further qualify the command by using the WHERE or KEYS option after the SEGMENT option.

You probably want to further qualify your GU commands with the WHERE or KEYS option, and specify a specific occurrence of a segment type. If you fully qualify a GU command, you can retrieve a segment regardless of your position in the database record.

## Examples

### *Example 1*

“What illness was Robert James here for most recently? Was he given any medication on that day for that illness? His patient number is 05136.”

*Explanation:* This example requests two pieces of information. To answer the first part of the request and retrieve the most recent ILLNESS segment, issue this GU command (assuming that PATNO1 contains 05163):

```
EXEC DLI GU
  SEGMENT(PATIENT) WHERE(PATNO=PATNO1)
  SEGMENT(ILLNESS) INTO(AREA);
```

Once you had retrieved the ILLNESS segment with the date of the patient's most recent visit to the clinic, you can issue another command to find out whether he was treated during that visit. If the date of his most recent visit was January 5, 1988, you can issue the following command to find out whether or not he was treated on that day for that illness (assuming PATNO1 contains 05163, and DATE1 contains 19880105):

```
EXEC DLI GU
  SEGMENT(PATIENT) WHERE(PATNO=PATNO1)
  SEGMENT(ILLNESS) WHERE(ILLDATE=DATE1)
  SEGMENT(TREATMNT) INTO(TRTAREA) WHERE(DATE=DATE1);
```

### Example 2

“What is Joan Carter currently being treated for? Her patient number is 10320.”

```
EXEC DLI GU
  SEGMENT(PATIENT) WHERE(PATNO=PATNO1)
  SEGMENT(ILLNESS) INTO(ILLAREA);
```

*Explanation:* In this example you want the ILLNESS segment for the patient whose patient number is 10320.

### Example 3

```
EXEC DLI GU
  SEGMENT(PATIENT)
  SEGMENT(ILLNESS)
  SEGMENT(TREATMNT) INTO(AREA);
```

*Explanation:* This example retrieves the first TREATMNT segment and specifies the three levels of qualification.

## Restriction

You must at least specify the SEGMENT option to indicate the segment type you want to retrieve.

## ISRT command

The Insert (ISRT) command is used to add one or more segments to the database.

### Format

```
EXEC DLI INSERT [ISRT] USING PCB(expression) A B
```

#### A For each parent segment (optional):

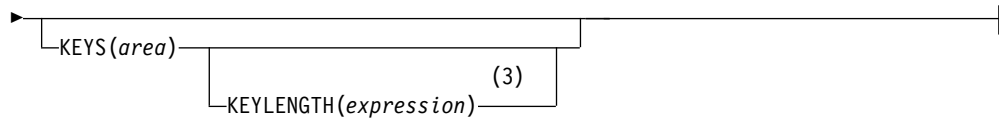
```

[VARIABLE]
[FIRST]
[LAST]
[CURRENT]
[SEGMENT(name)]
[SEGMENT((area))]
[SEGLength(expression)]

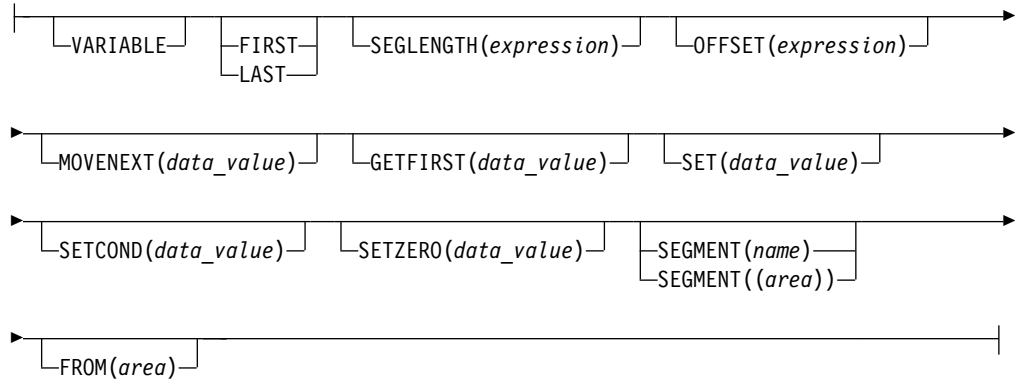
(1)
[FROM(area)]
[MOVENEXT(data_value)]
[GETFIRST(data_value)]

[SET(data_value)]
[SETCOND(data_value)]
[SETZERO(data_value)]

[WHERE(qualification statement)]
(2)
[FIELDLENGTH(expression)]
```



**B For the object segment (required):**



**Notes:**

- 1 Specify FROM on parent segments for a path command.
- 2 If you use multiple qualification statements, specify a length for each, using FIELDLENGTH. For example: **FIELDLENGTH(24,8)**
- 3 You can use either the Keys option or the Where option, but not both on one segment level.

**Options**

**USING PCB(expression)**

Specifies the DB PCB you want to use for the command. Its argument can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number.

**VARIABLE**

Indicates that a segment is variable-length.

**FIRST**

Specifies that you want to retrieve the first segment occurrence of a segment type, or that you want to insert a segment as the first occurrence. Use FIRST to insert a segment as a first occurrence of a segment type.

**LAST**

Specifies that you want to retrieve the last segment occurrence of a segment type, or that you want to insert a segment as the last segment occurrence. Use LAST to insert a segment as the last occurrence of a segment type.

**CURRENT**

Qualifies the command, and specifies that you want to use the level of and levels above the current position as qualifications for this segment. Use CURRENT to insert a segment based on your current position.

**SEGMENT(name), SEGMENT((area))**

Qualifies the command, specifying the name of the segment type or the area in the program containing the name of the segment type that you want to retrieve, insert, delete, or replace.



You must include at least a SEGMENT option for each segment you want to add to the database. Unless ISRT is a path command, the lowest level SEGMENT option specifies the segment being inserted. You cannot use a WHERE or KEYS option for this level.

If a segment has a unique key, DL/I inserts the segment in its key sequence. (If the segment does not have a key, or has a nonunique key, DL/I inserts it according to the value specified for the RULES parameter during DBDGEN.)

If you specify a SEGMENT option for only the lowest level segment, and do not qualify the parent segments with SEGMENT, WHERE, or KEYS options, you must make sure that the current position is at the correct place in the database to insert the segment. The SEGMENT option that DL/I assumes depends on your current position in the database record:

- If DL/I has a position established at the missing level, DL/I uses the segment on which position is established.
- If DL/I does not have a position established at the missing level, DL/I uses the first occurrence at that level.
- If DL/I moves forward from a position established at a higher level, DL/I uses the first occurrence at the missing level that falls within the new path.
- If you leave out a SEGMENT option for the *root* level, and DL/I has position established on a root, DL/I does not move from that root when trying to satisfy the command.

It is good practice to always provide qualifications for higher levels to establish the position of the segment being inserted.

If you are inserting a root segment, you need only specify a SEGMENT option. DL/I determines the correct place for its insertion in the database by the key taken from the I/O area. If the segment you are inserting is not a root segment, but you have just inserted its immediate parent, the segment can be inserted as soon as it is built in the I/O area just by using a SEGMENT option for it in the ISRT command. You need not code the parent level segments to establish your position.

When you specify multiple parent segments, you can mix segments with and without the WHERE option. If you include only SEGMENT options on parent segments, DL/I uses the first occurrence of each segment type to satisfy the command.

#### **SEGLength(expression)**

Specifies the length of the I/O area from which the segment is obtained. Its argument can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number. (It is required in COBOL programs for any SEGMENT level that specifies the INTO or FROM option.)

**Requirement:** The value specified for SEGLength must be greater than or equal to the length of the longest segment that can be processed by this call.

#### **FROM(area)**

Specifies an area containing the segment to be added, replaced, or deleted. Use FROM to insert one or more segments with one command.

#### **MOVENEXT(data\_value)**

Specifies a subset pointer to be moved to the next segment occurrence after your current segment.

**GETFIRST(data\_value)**

Specifies that you want the search to start with the first segment occurrence in a subset.

**SET(data\_value)**

Specifies unconditionally setting a subset pointer to the current segment.

**SETCOND(data\_value)**

Specifies conditionally setting a subset pointer to the current segment.

**SETZERO(data\_value)**

Specifies setting a subset pointer to zero.

**WHERE(qualification statement)**

Qualifies the command, specifying the segment occurrence. Its argument is one or more qualification statements, each of which compares the value of a field in a segment to a value you supply. Each qualification statement consists of:

- The name of a field in a segment
- The relational operator, which indicates how you want the two values compared
- The name of a data area in your program containing the value that is compared against the value of the field

WHERE establishes position on the parents of a segment when you are inserting that segment. You can do this by specifying a qualification of WHERE or KEYS for the higher level SEGMENT options.

When you specify multiple parent segments, you can mix segments with and without the WHERE option. If you include only SEGMENT options on parent segments, DL/I uses the first occurrence of each segment type to satisfy the command.

**FIELDLENGTH(expression)**

Specifies the length of the field value in a WHERE option.

**KEYS(area)**

Qualifies the command with the segment's concatenated key. You can use either KEYS or WHERE for a segment level, but not both.

KEYS can be used to qualify a parent segment. Instead of using WHERE, you can specify KEYS and use the concatenated key of the segment as qualification. You can use the KEYS option once for each command, immediately after the highest level SEGMENT option.

“Area” specifies an area in your program containing the segment's concatenated key.

**KEYLENGTH(expression)**

Specifies the length of the concatenated key when you use the KEYS option. It can be any expression in the host language that converts to the integer data type; if it is a variable, it must be declared as a binary halfword value. For IBM COBOL (or VS COBOL II), PL/I, or assembler language, KEYLENGTH is optional. For COBOL programs that are not compiled with the IBM COBOL for MVS & VM (or VS COBOL II) compiler, you must specify KEYLENGTH with the KEYS option.

## Usage

To add new segments to an existing database, use the ISRT command. When you issue the ISRT command, DL/I takes the data from the I/O area you have named

in the FROM option and adds the segment to the database. (The initial loading of a database requires using the LOAD command, instead of the ISRT command.)

You can use ISRT to add new occurrences of an existing segment type to a HIDAM, HISAM, or HDAM database. For an HSAM database, you can add new segments only by reprocessing the whole database or by adding the new segments to the end of the database.

Before you can issue the ISRT command to add a segment to the database, your program must build the segment to be inserted in an I/O area. If the segment has a key, you must place the correct key in the correct location in the I/O area. If field sensitivity is used, the fields must be in the order defined by the PSB for the application's view of the segment.

If you are adding a root segment occurrence, DL/I places it in the correct sequence in the database by using the key you supply in the I/O area. If the segment you are inserting is not a root, but you have just inserted its parent, you can insert the child segment by issuing an insert request qualified with only the segment name. You must build the new segment in your I/O area before you issue the ISRT request. You also qualify insert requests with the segment name when you add a new root segment occurrence. When you are adding new segment occurrences to an existing database, the segment *type* must have been defined in the DBD. You can add new segment occurrences directly or sequentially after you have built them in the program's I/O area.

If the segment type you are inserting has a unique key field, the location where DL/I adds the new segment occurrence depends on the value of its key field. If the segment does not have a key field, or if the key is not unique, you can control where the new segment occurrence is added by specifying either the FIRST, LAST, or HERE insert rule. Specify the rules on the RULES parameter of the SEGM statement for the database.

## Examples

### *Example 1*

“Add information to the record for Chris Edwards about his visit to the clinic on February 1, 1993. His patient number is 02345. He had a sore throat.”

*Explanation:* First, build the ILLNESS segment in your program's I/O area. Your I/O area for the ILLNESS segment looks like this:

```
19930201SORETHROAT
```

Use the command to add this new segment occurrence to the database is:

```
EXEC DLI ISRT
      SEGMENT(PATIENT) WHERE (PATNO=PATN01)
      SEGMENT(ILLNESS) FROM(ILLAREA);
```

### *Example 2*

“Add information about the treatment to the record for Chris Edwards, and add information about the illness.”

*Explanation:* You build the TREATMNT segment in a segment I/O area. The TREATMNT segment includes the date, the medication, amount of medication, and the doctor's name:

19930201MYOCINbbb  
0001TRIEBbbbbbb  
&b

The following command adds both the ILLNESS segment and the TREATMNT segment to the database:

```
EXEC DLI ISRT
      SEGMENT(PATIENT) WHERE (PATNO=PATN01)
      SEGMENT(ILLNESS) FROM(ILLAREA)
      SEGMENT(TREATMNT) FROM(TRETAREA);
```

### Example 3

```
EXEC DLI ISRT
      SEGMENT(ILLNESS) KEYS(CONKEY)
      SEGMENT(TREATMNT) FROM(TRETAREA);
```

*Explanation:* Using this command is the same as having a WHERE option qualified on the key field for the ILLNESS and PATIENT segments.

## Restrictions

Restrictions the ISRT command:

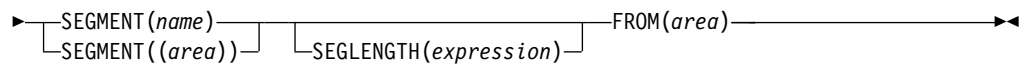
- You cannot issue the ISRT command until you have built a new segment in the I/O area.
- You must specify at least one SEGMENT option for each segment being added to the database.
- When inserting a segment, you must have position established on the parents of the segment.
- If you specify a SEGMENT option for only the lowest level segment, and do not qualify the parent segments with SEGMENT, WHERE, or KEYS options, be sure that current position is at the correct place in the database to insert the segment.
- If you use a FROM option for a segment, you cannot qualify the segment by using the WHERE or KEYS option; DL/I uses the key field value specified in the I/O area as qualification.
- You must use a separate I/O area for each segment type you want to add.
- You cannot mix SEGMENT options with and without the FROM option. When you use a FROM option for a parent segment, you must use a FROM option for each dependent segment. (You can begin the path at any level, but you must not leave out any levels.)
- You can only use the FIRST option with segments that have either no keys or have a nonunique key with HERE specified on the RULES operand of the SEGM statement in the DBD.
- You can only use the LAST option when the segment has no key or a nonunique key, and the INSERT rule for the segment is either FIRST or HERE.

## LOAD command

The Load (LOAD) command is used to add a segment sequentially while loading the database.

### Format

►►—EXEC—DLI—LOAD—┬──────────────────────────────────┬──────────┐  
                          └──USING PCB(expression)──┘ └──VARIABLE──┘



## Options

### USING PCB(expression)

Specifies the DB PCB you want to use. Its argument can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number.

### VARIABLE

Indicates that a segment is variable-length.

### SEGMENT(name)

Specifies the name of the segment type you want to retrieve, insert, delete, or replace.

### SEGMENT((area))

A reference to an area in your program containing the name of the segment type. You can specify an area instead of the name of the segment in the command.

### SEGLENGTH(expression)

Specifies the length of the I/O area from which the segment is obtained. Its argument can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number. (SEGLENGTH is required in COBOL programs for any SEGMENT level that specifies the INTO or FROM option.)

**Requirement:** The value specified for SEGLENGTH must be greater than or equal to the length of the longest segment that can be processed by this call.

### FROM(area)

Specifies an area containing the segment to be added, replaced, or deleted.

## Usage

The LOAD command is used for database load programs, which are described in *IMS Version 13 Database Administration*.

## Example

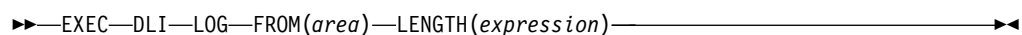
```

EXEC DLI LOAD
  SEGMENT(ILLNESS) FROM(ILLAREA);
  
```

## LOG command

The Log (LOG) command is used to write information to the system log.

### Format



## Options

### FROM(area)

Specifies an area containing the segment to be added, replaced, or deleted.

### LENGTH(expression)

Specifies the length of an area.

## Usage

You use the LOG command to write information to the system log.

## Example

```
EXEC DLI LOG
      FROM(ILLAREA) LENGTH(18);
```

## Restriction

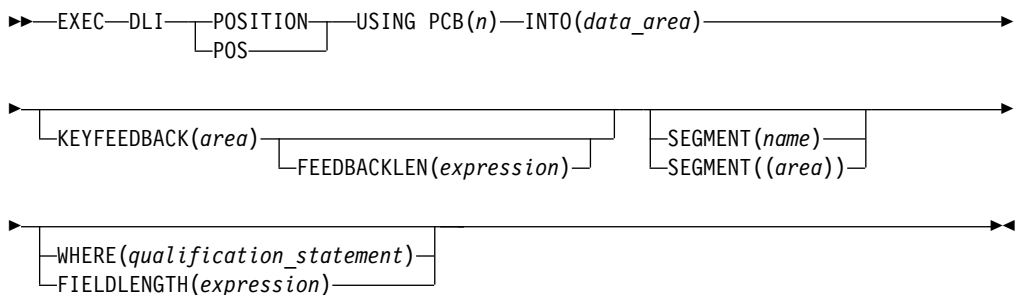
Restrictions for the LOG command:

- To use this command you must first define an I/O PCB for your program.

## POS command

The Position (POS) command retrieves the location of either a dependent or the segment.

## Format



## Options

### USING PCB(n)

Specifies the DB PCB you want to use for the command. Its argument can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number.

### INTO(data\_area)

Specifies an area into which the segment is read.

### KEYFEEDBACK(area)

Specifies an area into which the concatenated key for the segment is placed. If the area is not long enough, the key is truncated.

### FEEDBACKLEN(expression)

Specifies the length of the key feedback area into which you want the concatenated key retrieved. Its argument can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number. (FEEDBACKLEN is required in COBOL programs and optional in PL/I and assembler language programs.)

### SEGMENT(name)

Qualifies the command, specifying the name of the segment type you want to retrieve, insert, delete, or replace.

**SEGMENT((area))**

Is a reference to an area in your program containing the name of the segment type. You can specify an area instead of specifying the name of the segment in the command.

**WHERE(qualification statement)**

Qualifies the command, specifying the segment occurrence. Its argument is one or more qualification statements, each of which compares the value of a field in a segment to a value you supply.

**FIELDLENGTH(expression)**

Specifies the length of the field value in a WHERE option.

**Usage**

Use the POS command to:

- Retrieve the location of a specific sequential dependent segment, including the last one inserted
- Determine the amount of unused space within each DEDB area

If the area specified by the POS command is unavailable, the I/O area is unchanged and an FH status code is returned.

**Restriction**

The POS command is for DEDBs only.

**QUERY command**

The Query (QUERY) command obtains status code and other information in the DL/I interface block (DIB), which is a subset of the IMS PCB.

**Format**

►►—EXEC—DLI—QUERY—USING—PCB(*expression*)—————►►

**Options**

USING PCB(*expression*) is required. No other options are allowed with the QUERY command.

**Usage**

For full-function databases, the DIB should contain NA, NU, TH or blanks. For an explanation of the codes, see *IMS Version 13 Messages and Codes, Volume 4: IMS Component Codes*.

Use the QUERY command after scheduling the PSB but before making the database call. If the program has already issued a call using the DB PCB, you then use the REFRESH command to update the information in the DIB.

**Example***Example 1*

```
EXEC DLI QUERY USING PCB(expression);
```

*Explanation:* This example shows how to specify the QUERY command. In this example, (n) specifies the PCB.

### **Example 2**

```
EXEC DLI REFRESH DBQUERY;
```

*Explanation:* If your program has already issued a call using the DB PCB name, use the REFRESH command to update the information in the DIB. The REFRESH command updates all DB PCBs. You can issue it only one time.

## **Restrictions**

Restrictions for the QUERY command:

- To use this command you must first define an I/O PCB for your program.
- You cannot reestablish position in the midst of nonunique keys or nonkeyed segments.

## **REFRESH command**

The Refresh (REFRESH) command is used to obtain the most recent information from the DIB for the most recently issued command.

### **Format**

▶▶—EXEC—DLI—REFRESH—DBQUERY—————▶▶

### **Options**

DBQUERY is required. Other options are not allowed with the REFRESH command.

### **Usage**

The REFRESH command is used with the QUERY command.

The QUERY command is used after scheduling the PSB but before making the first database call. If the program has already issued a call using the DB PCB, use the REFRESH command to update the information in the DIB.

The REFRESH command updates all DB PCBs. It can be issued only once.

### **Example**

```
EXEC DLI REFRESH DBQUERY;
```

### *Explanation*

This example shows how to specify the REFRESH command.

## **Restrictions**

Restrictions for the REFRESH command:

- To use this command, you must first define an I/O PCB for your program.
- You cannot reestablish position in the midst of nonunique keys or nonkeyed segments.
- You can issue this command only one time.



## REPL command

The Replace (REPL) command is used to replace a segment, usually to change the values of one or more of its fields.

### Format

►► EXEC—DLI—REPLACE  
                  REPL—                  USING PCB(*expression*) | A | B |

#### A For each parent segment (optional):

|—VARIABLE—|—SEGMENT(*name*)—|—SEGLENGTH(*expression*)—|  
                  |—SEGMENT((*area*))—|

►—OFFSET(*expression*)—|—FROM(*area*)—|—MOVENEXT(*data\_value*)—|

►—SET(*data\_value*)—|—SETCOND(*data\_value*)—|—SETZERO(*data\_value*)—|

#### B For the object segment (required):

|—VARIABLE—|—SEGMENT(*name*)—|—SEGLENGTH(*expression*)—|  
                  |—SEGMENT((*area*))—|

►—OFFSET(*expression*)—|—FROM(*area*)—|—MOVENEXT(*data\_value*)—|

►—SET(*data\_value*)—|—SETCOND(*data\_value*)—|—SETZERO(*data\_value*)—|

## Options

### USING PCB(*expression*)

Specifies the DB PCB you want to use for the command. Its argument can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number.

### VARIABLE

Indicates that a segment is variable-length.

### SEGMENT(*name*)

Qualifies the command, specifying the name of the segment type you want to retrieve, insert, delete, or replace.

### SEGMENT((*area*))

Is a reference to an area in your program containing the name of the segment type. You can specify an area instead of specifying the name of the segment in the command.

### SEGLNGTH(*expression*)

Specifies the length of the I/O area from which the segment is obtained. Its argument can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program

containing a number. (It is required in COBOL programs for any SEGMENT level that specifies the INTO or FROM option.)

**Requirement:** The value specified for SEGLENGTH must be greater than or equal to the length of the longest segment that can be processed by this call.

**OFFSET(expression)**

Specifies the offset to the destination parent. It can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number. You use OFFSET when you process concatenated segments in logical relationships. It is required whenever the destination parent is a variable length segment.

**FROM(area)**

Specifies an I/O area containing the segment to be added, replaced or deleted. You can replace more than the segment by including the FROM option after the corresponding SEGMENT option for each segment you want to replace. Including FROM options for one or more parent segments is called a path command.

The argument following FROM identifies an I/O area that you have defined in your program. You must use a separate I/O area for each segment type you want to replace.

**MOVENEXT(data\_value)**

Specifies a subset pointer to be moved to the next segment occurrence after your current segment.

**SET(data\_value)**

Specifies unconditionally setting a subset pointer to the current segment.

**SETCOND(data\_value)**

Specifies conditionally setting a subset pointer to the current segment.

**SETZERO(data\_value)**

Specifies setting a subset pointer to zero.

## Usage

You must qualify the REPL command with at least one SEGMENT and FROM option, which together indicate the retrieved segments you want replaced.

If the Get command that preceded the REPL command was a path command, and you do not want to replace all of the retrieved segments or the PSB does not have replace sensitivity for all of the retrieved segments, you can indicate which of the segments are not to be replaced by omitting the SEGMENT option.

If your program attempts to do a path replace of a segment where it does not have replace sensitivity, the data for the segment in the I/O area for the REPL command must be the same as the segment returned on the preceding GET command. If the data changes in this situation, the transaction is abended and no data is changed as a result of the Replace command.

Notice that the rules for a REPL path command differ from the rules for an ISRT path command. You cannot skip segment levels to be inserted with an ISRT command, as you can with a REPL command.

To update information in a segment, you can use the REPL command. The REPL command replaces data in a segment with data you supply in your application program. First, you must retrieve the segment into an I/O area. You then modify

the information in the I/O area and replace the segment with the REPL command. For your program to successfully replace a segment, that segment must already have been defined as replace-sensitive in the PCB by specifying PROCOPT=A or PROCOPT=R on the SENSEG statement in the PCB.

You cannot issue any commands using the same PCB between a Get command and the REPL command, and you can issue only one REPL command for each Get command.

## Examples

### Example 1

```
EXEC DLI GU SEGMENT(PATIENT) INTO(PATAREA);  
EXEC DLI REPL SEGMENT(PATIENT) FROM(PATAREA);
```

*Explanation:* This example shows that you cannot issue any commands using the same PCB between the Get command and the REPL command, and you can issue only one REPL command for each Get command. If you issue this commands and wanted to modify information in the segment again, you must first reissue the GU command, before reissuing the REPL command.

### Example 2

“We have received a payment for \$65.00 from a patient whose ID is 08642. Update the patient's billing record and payment record with this information, and print a current bill for the patient.”

*Explanation:* The four parts to satisfying this processing request are:

1. Retrieve the BILLING and PAYMENT segments for the patient.
2. Calculate the new values for these segments by subtracting \$65.00 from the value in the BILLING segment, and adding \$65.00 to the value in the PAYMENT segment.
3. Replace the values in the BILLING and PAYMENT segments with the new values.
4. Print a bill for the patient, showing the patient's name, number, address, the current amount of the bill, and the amount of the payments to date.

To retrieve the BILLING and PAYMENT segments, issue a GU command. Because you also need the PATIENT segment when you print the bill, you can include INTO following the SEGMENT options for the PATIENT segment and for the BILLING segment:

```
EXEC DLI GU  
    SEGMENT(PATIENT) INTO(PATAREA) WHERE (PATNO=PATNO1)  
    SEGMENT(BILLING) INTO(BILLAREA)  
    SEGMENT(PAYMENT) INTO(PAYAREA);
```

After you have calculated the current bill and payment, you can print the bill, then replace the billing and payment segments in the database. Before issuing the REPL command, you must change the segments in the I/O area.

Because you have not changed the PATIENT segment, you do not need to replace it when you replace the BILLING and PAYMENT segments. To indicate to DL/I that you do not want to replace the PATIENT segment, you do not specify the SEGMENT option for the PATIENT segment in the REPL command.

```
EXEC DLI REPL
  SEGMENT(BILLING) FROM(BILLAREA)
  SEGMENT(PAYMENT) FROM(PAYAREA);
```

This command tells DL/I to replace the BILLING and PAYMENT segments, but not to replace the PATIENT segment.

These two examples are called *path commands*. You use a path REPL command to replace more than one segment with one command.

### Example 3

“Steve Arons, patient number 10250, has moved to a new address in this town. His new address is 4638 Brooks Drive, Lakeside, California. Update the database with his new address.”

*Explanation:* You need to retrieve the PATIENT segment for Steve Arons and replace the address portion of the segment. To retrieve the PATIENT segment, you can use this GU command (assuming PATNO1 contains 10250):

```
EXEC DLI GU
  SEGMENT(PATIENT) INTO(PATAREA) WHERE (PATNO=PATNO1);
```

Since you are not replacing the first two fields of the PATIENT segment (PATNO and NAME), you do not have to change them in the I/O area. Place the new address in the I/O area following the PATNO and NAME fields. Then you issue the REPL command:

```
EXEC DLI REPL
  SEGMENT(PATIENT) FROM(PATAREA);
```

### Example 4

```
EXEC DLI GU SEGMENT(PATIENT) INTO(PATAREA)
  WHERE (PATNO=PATNO1)
  SEGMENT(ILLNESS) INTO(ILLAREA)
  SEGMENT(TREATMNT) INTO(TRETAREA);
EXEC DLI REPL SEGMENT(PATIENT) FROM(PATAREA)
  SEGMENT(TREATMNT) FROM(TRETAREA);
```

*Explanation:* This example assumes that you want to replace the PATIENT and TREATMNT segments for patient number 10401, but you do not want to change the ILLNESS segment. To do this issue this command (assuming PATNO1 contains 10401).

## Restrictions

Restrictions for the REPL command:

- You cannot issue any commands using the same PCB between the Get command and the REPL command.
- You can issue only one REPL command for each Get command.
- To modify information in a segment, you must first reissue the GU command before reissuing the REPL command.
- You must qualify the REPL command with at least one SEGMENT option and one FROM option.
- If you use a FROM option for a segment, you cannot qualify the segment by using the WHERE or KEYS option; DL/I uses the key field value specified in the I/O area as qualification.

## RETRIEVE command

Use the RETRIEVE command to determine current position in the database in batch and BMP programs.

### Format

```
▶▶—EXEC—DLI—RETRIEVE—USING PCB(expression)—KEYFEEDBACK(area)—————▶▶  
▶— FEEDBACKLEN(expression)—————▶▶
```

### Options

#### USING PCB(*expression*)

Specifies the DB PCB you want to use for the command. Its argument can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number.

**expression** specifies the PCB for which you want to retrieve the concatenated key. It can be any expression in the host language that converts to the integer data type. You can specify either a number or a reference to a halfword containing a number. The value must be a positive integer not greater than the number of PCBs generated for the PSB. The first PCB in the list, the I/O PCB, is 1. The first DB PCB in the list is 2, the second is 3, and so forth.

#### KEYFEEDBACK(*area*)

Specifies an area into which the concatenated key for the segment is placed. If the area is not long enough, the key is truncated.

#### FEEDBACKLEN(*expression*)

Specifies the length of the key feedback area into which you want the concatenated key retrieved. Its argument can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number. (It is required in COBOL programs and optional in PL/I and assembler language programs.)

**expression** is the length of the key feedback I/O area. It can be any expression in the host language that converts to integer data type; you can specify either a number or a reference to a halfword containing a number. For IBM COBOL for z/OS & VM (or VS COBOL II), PL/I, or assembler language, FEEDBACKLEN is optional. For COBOL programs that are not compiled with the IBM COBOL for z/OS & VM (or VS COBOL II) compiler, you must specify FEEDBACKLEN with the KEYFEEDBACK option.

### Usage

If your program issues symbolic checkpoint commands it must also issue the extended RESTART (XRST) command or the RETRIEVE command. The RETRIEVE command is issued once, at the start of your program. You can use the RETRIEVE command to start your program normally, or to restart it in case of an abnormal termination.

You can use the RETRIEVE command from a specific checkpoint id or a time/date stamp. Because the RETRIEVE command attempts to reposition the database, your program also needs to check for correct position.

After issuing the RETRIEVE command, the segment type and level on which the position is established is returned to the DIBSEGM and DIBSEGLV fields in the

DIB. The value in DIBKFBL is set to the actual length of the concatenated key. The DIBSTAT field contains the value returned from the GU repositioning, not the XRST command.

The RESTART command attempts to reposition DL/I databases by issuing an internal GU qualified with the concatenated key. It is your responsibility to verify that your position in the database from the GU repositioning is the correct position for the checkpoint ID used in the XRST command. You can use the RETRIEVE command to retrieve the concatenated key used with the GU repositioning, and determine your current position in all the PCBs your program accesses.

### Example

```
EXEC DLI RETRIEVE USING PCB(2) KEYFEEDBACK(KEYAREA);  
EXEC DLI RETRIEVE USING PCB(5) KEYFEEDBACK(KEYAREA);
```

### Explanation

These RETRIEVE commands retrieve the concatenated key for the first and fourth DB PCBs. (The first PCB in the list is the I/O PCB, so the first DB PCB is the second one in the list.) After issuing the first RETRIEVE command, you can determine your position in the first DB PCB by examining the concatenated key in KEYAREA, and the values returned in the DIBSEGM and DIBSEGLV fields in the DIB. After issuing the second RETRIEVE command, you can determine your position in the fourth DB PCB by examining the same fields.

### Restrictions

Restrictions for the RETRIEVE command:

- You cannot use this command in a CICS program.
- To use this command, you must first define an I/O PCB for your program.
- You cannot reestablish position in the midst of nonunique keys or nonkeyed segments.
- You cannot use this command unless the system log is stored on direct access storage and dynamic backout has been specified. You must also specify BKO=Y in the parm field of your JCL when you execute the program.

## ROLB command

The Rollback (ROLB) command is used to dynamically back out changes and return control to your program. You cannot use this command in a CICS program.

### Format

►►—EXEC—DLI—ROLB—◄◄

### Options

No options are allowed with the ROLB command.

### Usage

When a batch or BMP program determines that some of its processing is invalid, two commands make it possible for the program to remove the effects of its inaccurate processing. These are the rollback commands, ROLL and ROLB.

The ROLB command is valid in batch programs when the system log is stored on direct access storage and dynamic backout has been specified through the use of the BKO execution parameter.

Issuing the ROLB causes IMS DB to back out any changes your program has made to the database since its last checkpoint, or since the beginning of the program if your program has not issued a checkpoint. When you issue a ROLB command, IMS DB returns control to your program after backing out the changes, so that your program can continue processing with the next statement after the ROLB command.

### Example

```
EXEC DLI ROLB;
```

### *Explanation*

This example shows how to dynamically back out changes and return control to your program with the ROLB command.

### Restrictions

Restrictions for the ROLB command:

- You cannot use this command in a CICS program.
- You must first define an I/O PCB for your program before you can use this command.
- You cannot reestablish position in the midst of nonunique keys or nonkeyed segments.
- You cannot use this command when the system log is stored on direct access storage and dynamic backout has not been specified.

**Related reference:**

“ROLL command”

## ROLL command

The Roll (ROLL) command is used to dynamically back out changes. You cannot use this command in a CICS program;

### Format

▶▶—EXEC—DLI—ROLL—————▶▶

### Options

No options are allowed with the ROLL command.

### Usage

When a batch program determines that some of its processing is invalid, two commands make it possible for the program to remove the effects of its inaccurate processing. These are the rollback commands, ROLL and ROLB.

You can use ROLL in batch programs.

Issuing the ROLL causes CICS and DL/I to back out any changes your program has made to the database since its last checkpoint, or since the beginning of the

program provided your program has not issued a checkpoint. When you issue a ROLL command, DL/I terminates your program after backing out the updates.

### Example

```
EXEC DLI ROLL;
```

### Explanation

This example shows how to dynamically back out changes with the ROLL command.

If you use the ROLL command, IMS terminates the program with user abend code U0778. This type of abnormal termination does not produce a storage dump.

### Restrictions

Restrictions for the ROLL command:

- You cannot use this command in a CICS program.
- You must first define an I/O PCB for your program before you can use this command.
- You cannot reestablish position in the midst of nonunique keys or nonkeyed segments.
- You cannot use this command when the system log is stored on direct access storage and dynamic backout has been specified. You must also specify BKO=Y in the parm field of your JCL when you execute the program.

### Related reference:

“ROLB command” on page 200

## ROLS command

The Rollback to SETS or SETU (ROLS) command is used to back out to a processing point set by an earlier SETS command.

### Format

```
▶▶—EXEC—DLI—ROLS—┬—USING PCB(expression)———▶▶  
└—TOKEN(token)—AREA(data_area)—┘
```

### Options

#### USING PCB(expression)

Specifies the DB PCB you want to use. Its argument can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number.

#### TOKEN(token)

A 4-byte token associated with the current processing point. If you specify both TOKEN and AREA, the ROLS command backs out to the SETS or SETU you specified.

#### AREA(data\_area)

The name of the area to be restored to the program when a ROLS command is issued. The first 2 bytes of the data-area field contain the length of the



data-area, including the length itself. The second 2 bytes must be set to X'0000'. If you specify both TOKEN and AREA, the ROLS command backs out to the SETS you specified.

The ROLS call has two formats: with TOKEN and AREA (for IOPCB only) and without TOKEN and AREA (for IOPCB or DBPCB).

## Usage

Use the SETS and ROLS commands to define multiple points at which to preserve the state of DL/I full-function databases and to return to these points later. (For example, you can use them so your program can handle situations that can occur when PSB scheduling completes without all of the referenced DL/I databases being available.)

Use of the SETS and ROLS commands apply only to DL/I full-function databases. This means that if a logical unit of work (LUW) is updating types of recoverable resources other than full-function databases, for example, VSAM files, the SETS and ROLS requests have no effect on the non-DL/I resources. The backout points are *not* CICS commit points; they are intermediate backout points that apply only to DBCTL resources. It is up to you to ensure the consistency of all the resources involved.

You can use the ROLS command to backout to the state all full-function databases were in before either a specific SETS or SETU request or the most recent commit point.

## Examples

### *Example 1*

```
EXEC DLI ROLS TOKEN(token1) AREA(data_area)
```

*Explanation:* In this example (for IOPCB only), backout takes place to the corresponding TOKEN, as specified by a prior SETS call, and control returns to the application.

### *Example 2*

```
EXEC DLI ROLS USING PCB(PCB5)
```

*Explanation:* In this example, for IOPCB or DBPCB, backout takes place to the prior sync point and the application is pseudoabend with a U3033 status code. Control does **not** return to the application.

In this example, PCB5 is the number of a DB PCB that has received a 'data unavailable' status code. This command results in the same action that would have occurred had the program not issued an ACCEPT STATUSGROUPA command. (See the topic "Data Availability Enhancements" in IMS Version 13 Application Programming.)

### *Example 3*

```
EXEC DLI ROLS
```

*Explanation:* In this example, for IOPCB or DBPCB, backout takes place to the prior sync point and the application is pseudoabend with a U3033, provided the

previous reference to that PCB gave an unavailable status code. Control does **not** return to the application.

## Restrictions

Restrictions for the ROLS command:

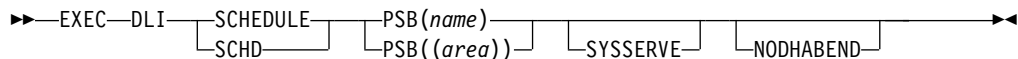
- To use this command you must first define an I/O PCB for your program.
- You cannot reestablish position in the midst of nonunique keys or nonkeyed segments.
- You cannot use this command when the system log is stored on direct access storage and dynamic backout has been specified. You must also specify BKO=Y in the parm field of your JCL when you execute the program.

## SCHD command

The Schedule (SCHD) command is used to schedule a PSB in a CICS online program.

For information on the I/O PCB, see the topic "PCBs and PSB" in *IMS Version 13 Application Programming*.

## Format



## Options

### PSB(name)

Specifies the name of the PSB available to your application program that you want to schedule with the SCHD command.

### PSB((area))

Specifies an 8-byte data area in your program that contains the name of the PSB available to your program that you want to schedule with the SCHD command.

### SYSSERVE

Specifies that the application program can handle an I/O PCB and might issue a system service request in the logical unit of work (LUW).

### NODHABEND

Specifies that a CICS transaction does not fail with a DHxx abend.

Should a schedule fail under EXEC DLI, a status code might be returned in the DIB, causing a CICS transaction to fail with a DHxx abend. This option prevents this. Following an unsuccessful SCHD command, the control, as well as the status code in the DIB are passed back to the application program, which can then take the appropriate action.

## Usage

Before you can access DL/I databases from a CICS program, you must notify DL/I that your program will be accessing a database by scheduling a PSB. Do this by issuing the SCHD command. When you no longer plan to use a PSB, or you want to schedule a subsequent PSB (one or more), you must terminate the previous PSB

with the TERM command. (For more information on the I/O PCB and PSB, see the topic "PCBs and PSB" in *IMS Version 13 Application Programming*)

The SCHD command can be specified two ways, as shown by the following code examples.

### Example

```
EXEC DLI SCHD PSB(psbname)SYSSERVE;  
EXEC DLI SCHD PSB((AREA));
```

### Explanation

These examples show two ways to schedule a PSB in a CICS program.

## SETS command

The Set a Backout Point (SETS) command is used to define points in your application at which to preserve the state of the DL/I databases before initiating a set of DL/I requests to perform a function. Your application can issue a ROLS command later if it cannot complete the function.

### Format

```
▶▶ EXEC—DLI—SETS —————▶  
                        |  
                        | TOKEN(mytoken)—AREA(data_area) |  
                        |
```

### Options

#### TOKEN(mytoken)

A 4-byte token associated with the current processing point.

#### AREA(data\_area)

The name of the area to be restored to the program when a SETS command is issued. The first 2 bytes of the data-area field contain the length of the data-area, including the length itself. The second 2 bytes must be set to X'0000'.

### Usage

You can use the SETS command to define multiple points at which to preserve the state of the DL/I databases and to return to these points later. For example, you can use the SETS command to allow your program to handle situations that can occur when PSB scheduling completed without all of the referenced DL/I databases being available.

The SETS command applies only to DL/I full-function databases. If a logical unit of work (LUW) is updating types of recoverable resources other than full-function databases, for example VSAM files, the SETS command has no effect on the non-DL/I resources. The backout points are *not* CICS commit points; they are intermediate backout points that apply only to DBCTL resources. It is up to you to ensure the consistency of all the resources involved.

### Example

```
EXEC DLI SETS TOKEN(mytoken) AREA(data_area)
```

### Explanation

This example shows how to specify the SETS command.

## Restrictions

Restrictions for the SETS command:

- To use this command you must first define an I/O PCB for your program.
- You cannot reestablish position in the midst of nonunique keys or nonkeyed segments.
- In batch, you can only use this command when the system log is stored on direct access storage and dynamic backout has been specified. You must also specify BKO=Y in the parm field of your JCL when you execute the program.
- It is rejected when the PSB contains a DEDB or MSDB PCB, or when the call is made to a DB2 database.
- It is valid, but not functional, if unsupported PCBs exist in the PSB or if the program uses an external subsystem.

## SETU command

The Set a Backout Point Unconditionally (SETU) command is identical to the SETS command except that it does not get rejected if unsupported PCBs are in the PSB or if the program uses an external subsystem.

### Format

▶▶—EXEC—DLI—SETU———▶  
                                  └─TOKEN(mytoken)—AREA(data\_area)—┘

### Options

#### TOKEN(mytoken)

A 4-byte token associated with the current processing point.

#### AREA(data\_area)

The name of the area to be restored to the program when a SETU command is issued. The first 2 bytes of the data-area field contain the length of the data-area, including the length itself. The second 2 bytes must be set to X'0000'.

### Usage

You can use the SETU command to define multiple points at which to preserve the state of the DL/I databases and to return to these points later. For example, you can use the SETU command to allow your program to handle situations that can occur when PSB scheduling completed without all of the referenced DL/I databases being available.

The SETU command applies only to DL/I full-function data bases. If a logical unit of work (LUW) is updating types of recoverable resources other than full-function databases, such as VSAM files, the SETU command has no effect on the non-DL/I resources. The backout points are *not* CICS commit points; they are intermediate backout points that apply only to DBCTL resources. It is up to you to ensure the consistency of all the resources involved.

## Example

```
EXEC DLI SETU TOKEN(mytoken) AREA(data_area)
```

### Explanation

This example shows how to specify the SETU command.

## Restrictions

Restrictions for the SETU command:

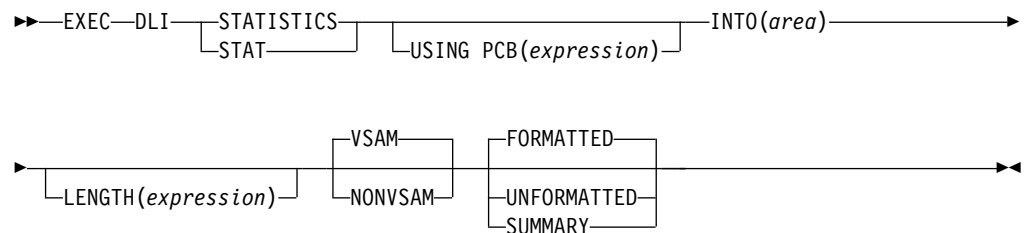
- You cannot use this command in a CICS program.
- To use this command you must first define an I/O PCB for your program.
- You cannot reestablish position in the midst of nonunique keys or nonkeyed segments.
- You cannot use this command when the system log is stored on direct access storage and dynamic backout has been specified. You must also specify BKO=Y in the parm field of your JCL when you execute the program.

## STAT command

The Statistics (STAT) command is used to obtain IMS database statistics that you can use in debugging your program.

**This topic contains Product-sensitive Programming Interface information.**

### Format



## Options

### USING PCB(expression)

Specifies the DB PCB you want to use. Its argument can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number.

### INTO(area)

Specifies an area into which the data is read.

### LENGTH(expression)

Specifies the length of an area.

### VSAM/NONVSAM

Specifies database type.

### FORMATTED/UNFORMATTED/SUMMARY

Specifies type of output.

## Usage

The STAT command is described in *IMS Version 13 Application Programming*.

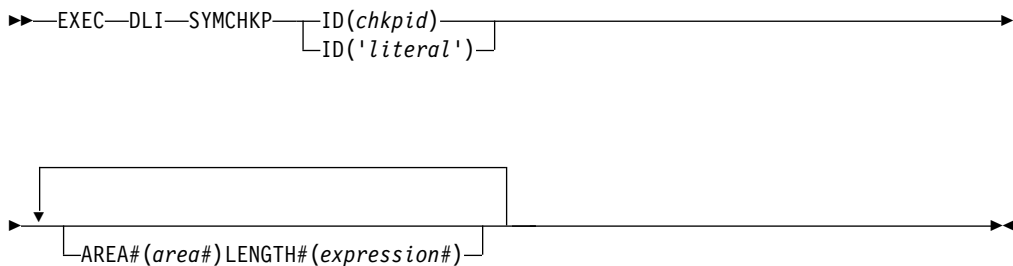
## Example

For examples of the STAT command, see *IMS Version 13 Application Programming*.

## SYMCHKP command

The Symbolic Checkpoint (SYMCHKP) command is used to issue a symbolic checkpoint and to end a logical unit of work.

### Format



### Options

#### ID(chkpid)

Is the name of an 8-byte area in your program containing the checkpoint ID. If you are using PL/I, specify this parameter as a pointer to a major structure, an array, or a character string.

#### ID('literal')

Is the 8-byte checkpoint ID, enclosed in quotation marks.

#### AREA#(area#)

Specifies the areas in your program you want IMS to checkpoint. You do not need to specify any area to checkpoint; however, you cannot specify more than seven areas. If you specify more than one area, you must include all intervening areas. For example, if you specify AREA3, you must also specify AREA1 and AREA2. The areas you specify using the SYMCHKP command must be the same and in the areas specified in the XRST command.

#### LENGTH#(expression#)

Can be any expression in the host language that converts to the integer data type; you can specify either a number or a reference to a halfword containing a number. For IBM COBOL for z/OS & VM (or VS COBOL II), PL/I, or assembler language programs, LENGTH1 to LENGTH7 are optional. For COBOL programs that are not compiled with the IBM COBOL for z/OS & VM (or VS COBOL II) compiler, LENGTHx (where x is 1 to 7) is required for each AREAx (where x is 1 to 7) that you specify.

## Usage

The two kinds of commands that allow you to make checkpoints are: the CHKP, or basic Checkpoint command, and the SYMCHKP, or Symbolic Checkpoint command.

Batch programs can use either the symbolic or the basic command.

Both checkpoint commands make it possible for you to commit your program's changes to the database and to establish places from which the program can be restarted, should it terminate abnormally. You must not use the `CHKPT=EOV` parameter on any DD statement to take an IMS checkpoint.

Refer to *IMS Version 13 Application Programming* for an explanation of when and why you should issue checkpoints in your program. Both commands cause a loss of database position at the time the command is issued. Position must be reestablished by a GU command or other method of establishing position.

In addition to committing your program's changes to the database and establishing places from which your program can be restarted, the Symbolic Checkpoint command:

- Works with the Extended Restart (XRST) command to restart your program if it terminates abnormally.
- Can save as many as seven data areas in your program, which are restored when your program is restarted. You can save variables, counters, and status information.

### Example

```
EXEC DLI SYMCHKP
      ID(chkpid)
      AREA1(area1) LENGTH1(expression1)
      ...
      AREA7(area7) LENGTH7(expression7)
```

### Explanation

This example shows how to issue a symbolic checkpoint and to end a logical unit of work with a SYMCHKP command.

### Restrictions

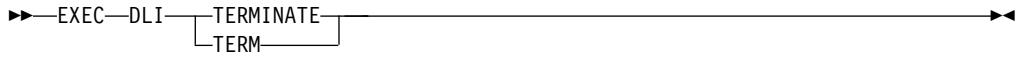
Restrictions for the SYMCHKP command:

- If you issue this command, you must also issue the XRST command.
- You cannot use this command in a CICS program.
- To use the SYMCHKP command you must first define an I/O PCB for your program.
- You cannot reestablish position in the midst of nonunique keys or nonkeyed segments.
- The areas you specify using the SYMCHKP command must be the same, and in the same order, as the areas specified in the XRST command.
- If you specify more than one area, you must specify all intervening areas. For example, if you specify AREA3, you must also specify AREA1 and AREA2.
- When specifying expression1 with a COBOL program that is not compiled with the IBM COBOL for z/OS & VM (or the VS COBOL II) compiler, LENGTHx (where x is 1 to 7) is required for each AREAx (where x is 1 to 7) that you specify.

## TERM command

The Terminate (TERM) command is used to terminate a PSB in a CICS online program.

## Format



## Options

No options are allowed with the TERM command.

## Usage

If you want to use a PSB other than the one already scheduled, use the TERM command to release the PSB.

When you issue the TERM command, all database changes are committed and cannot be backed out. Because returning to CICS also terminates the PSB and commits changes, you need not use the TERM command unless you want to schedule another PSB, or commit database changes before returning to CICS.

No options are allowed with the TERM command. If your program subsequently needs a PSB that has terminated, it must reschedule that PSB by issuing another SCHD command.

In most applications, you do not need to use the TERM command.

## Example

```
EXEC DLI TERM
```

### *Explanation*

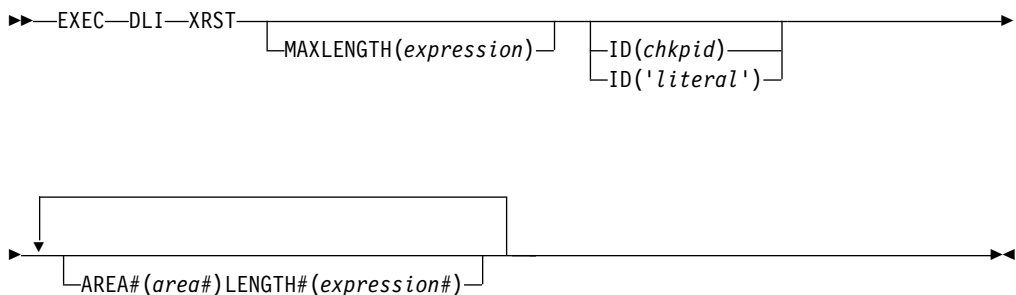
This example shows how to terminate a PSB with the TERM command.

## XRST command

The Extended Restart (XRST) command is used to issue an extended restart, and to perform a normal start or an extended restart from a checkpoint ID or time/date stamp.

If you use Symbolic Checkpoint commands in your program, you must use the XRST command.

## Format





## Options

### **MAXLENGTH(expression)**

Specifies the length of an area from which a program is restarted. This parameter is the longest segment in the PSB, or of all the segments in a path, if you use path commands in your program. It can be any expression in the host language that converts to the integer data type. You can specify either a number or a reference to a halfword containing a number. MAXLENGTH is not required, and defaults to 512 bytes.

### **ID(chkpid) ID('literal')**

This parameter is either the name of a 30-byte area in your program or a 30-byte checkpoint ID, enclosed in quotation marks. This parameter is optional; you can specify a checkpoint ID or a time/date stamp in the parm field of your JCL instead. If you specify both, IMS uses the value in the parm field of the EXEC statement. If you are starting your program normally, do not specify a checkpoint ID, or ensure that the field pointed to by the *chkpid* contains 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.

You can issue a XRST command after supplying a time/date stamp of IIIIDDDHHMMSST, or from a specific checkpoint in your program by supplying a checkpoint ID. IIIIDDD is the region ID and day; HHMMSST is the actual time in hours, minutes, seconds, and tenths of seconds. The system message DFS0540I supplies the checkpoint ID and time/date stamp.

If you are using PL/I, specify *chkpid* as a pointer to a major structure, an array, or a character string.

### **AREA#(area#)**

Area# specifies the first area in your program you want to restore. You can specify up to seven areas. You are not required to specify any areas; however, if you specify more than one area, you must include all intervening areas. For example, if you specify AREA3, you must also specify AREA1, and AREA2. The areas you specify on the XRST command must be the same—and in the same order—as the areas you specify on the SYMCHKP command. When you restart the program, only the areas you specified in the SYMCHKP command are restored.

### **LENGTH#(expression#)**

Specifies the length of an area from which a program is restarted. Its argument can be any expression in the host language that converts to the integer data type; you can specify either a number or a reference to a halfword containing a number. For IBM COBOL for z/OS & VM (or VS COBOL II), PL/I, or assembler language programs LENGTH1 to LENGTH7 are optional. For COBOL programs that are not compiled with the IBM COBOL for z/OS & VM (or VS COBOL II) compiler, LENGTHx (where x is 1 to 7) is required for each AREAx (where x is 1 to 7) that you specify. Each qualification statement consists of:

- The name of a field in a segment
- The relational operator, which indicates how you want the two values compared
- The name of a data area in your program containing the value that is compared against the value of the field

## Usage

If your programs issues Symbolic Checkpoint commands it must also issue the Extended Restart (XRST) command. The XRST is issued once, at the start of your program. You can use the XRST command to start your program normally, or to extend restart it in case of an abnormal termination.

You can extend restart your program from a specific checkpoint ID, or a time/date stamp. Because the XRST attempts to reposition the database, your program also needs to check for correct position.

After issuing the XRST command, you should test the DIBSEGM field in the DIB. After a normal start, the DIBSEGM field should contain blanks. At the completion of an Extended Restart, the DIBSEGM field will contain a checkpoint ID. Normally, XRST will return the 8-byte symbolic checkpoint ID, followed by 4 blanks. If the 8-byte ID consists of all blanks, then XRST will return the 14-byte time-stamp ID. The only successful status code for an XRST command is a blank status code. If DL/I detects any error while processing the XRST command, your program abends.

## Example

```
EXEC DLI XRST MAXLENGTH(expression)
      ID(chkpid)
      AREA1(area1) LENGTH1(expression1)
      ...
      AREA7(area7) LENGTH7(expression7)
```

### *Explanation*

This example shows how to specify the XRST command.

## Restrictions

Restrictions for the XRST command:

- You cannot use this command in a CICS program.
- To use this command you must first define an I/O PCB for your program.
- You cannot reestablish position in the midst of nonunique keys or nonkeyed segments.
- You cannot use this command unless the system log is stored on direct access storage and dynamic backout has been specified. You must also specify BKO=Y in the parm field of your JCL when you execute the program.

---

## Command code reference

Use the following reference information for the command codes.

**Restriction:** Command codes cannot be used by MSDB calls.

**Restrictions:** The following restrictions apply for Fast Path secondary index command code and multiple SSA support:

- The C command code cannot be specified in any SSA other than the first SSA. If specified, it will be rejected with a status code of AJ.
- The V command code for an ISRT call is ignored.
- A, G, and subset pointer related command codes (M, R, S, W, and Z) are not supported. They are rejected with a status code of AJ.

**Restrictions:** The following restrictions apply for any DL/I call for a physical parent segment of the target segment where target segment is not the root segment:

- The P, Q, U, and V command codes are ignored.
- The field name must be the sequence field name for the parent segment if the SSA contains a qualification statement. If any field name other than the sequence field name is specified, it will be rejected with a status code of AK.

*Table 51. Summary of command codes*

<b>Command code</b>	<b>Description</b>
A	Clear positioning and start the call at the beginning of the database.
C	Use the concatenated key of a segment to identify the segment.
D	Retrieve or insert a sequence of segments in a hierarchic path using only one call, instead of using a separate (path) call for each segment.
F	Back up to the first occurrence of a segment under its parent when searching for a particular segment occurrence. Disregarded for a root segment.
G	Prevent randomization or the calling of the HALDB Partition Selection exit routine and search the database sequentially.
L	Retrieve the last occurrence of a segment under its parent.
M	Move a subset pointer to the next segment occurrence after your current position. (Used with DEDBs only.)
N	Designate segments that you do not want replaced when replacing segments after a Get Hold call. Typically used when replacing a path of segments.
O	Either field names or both segment position and lengths can be contained in the SSA qualification for combine field position.
P	Set parentage at a higher level than what it usually is (the lowest-level SSA of the call).
Q	Reserve a segment so that other programs cannot update it until you have finished processing and updating it.
R	Retrieve the first segment occurrence in a subset. (Used with DEDBs only.)
S	Unconditionally set a subset pointer to the current position. (Used with DEDBs only.)
U	Limit the search for a segment to the dependents of the segment occurrence on which position is established.
V	Use the hierarchic level at the current position and higher as qualification for the segment.
W	Set a subset pointer to your current position, if the subset pointer is not already set. (Used with DEDBs only.)
Z	Set a subset pointer to 0, so it can be reused. (Used with DEDBs only.)
-	Null. Use an SSA in command code format without specifying the command code. Can be replaced during execution with the command codes that you want.

The following table shows the list of command codes with applicable calls.

Table 52. Command codes and related calls

Command Code	GU	GHU	GN	GHN	GNP GHNP	REPL	ISRT	DLET
A				X				
C	X		X		X		X	
D	X		X		X		X	
F	X		X		X		X	
G				X				
L	X		X		X		X	
M	X		X		X	X	X	
N						X		
O	X		X		X		X	
P	X		X		X		X	
Q	X		X		X		X	
R	X		X		X		X	
S	X		X		X	X	X	
U	X		X		X		X	
V	X		X		X		X	
W	X		X		X	X	X	
Z	X		X		X	X	X	X
-	X		X		X	X	X	X

## A command code

You can use the A command code to cause position in the database to be cleared which will result in the call starting at the beginning of the database.

If an application had been traversing through a database and not finding the requested data down a certain path, it could issue a qualified GN or GHN call with command code A to reset position at the beginning of the database and search a different path for the data.

## C command code

You can use the C command code to indicate to IMS that (instead of supplying a qualification statement) you are supplying the segment's concatenated key as a means of identifying it. You can use either the C command code or a qualification statement, but not both.

You can use the C command code for all Get calls and for the ISRT call. When you code the concatenated key, enclose it in parentheses following the \*C, and place it in the same position that would otherwise contain the qualification statement.

For example, suppose you wanted to satisfy this request:

Did Joan Carter visit the clinic on March 3, 2009? Her patient number is 07755.

The PATIENT segment's key field is the patient number, and the ILLNESS segment's key field is the date field, so the concatenated key is 0775520090303. This

number is comprised of four digits for the year, followed by two digits for both the month and the day. You issue a GU call with the following SSA to satisfy the request:

```
GU ILLNESSb*C(0775520090303)
```

Using the C command code is sometimes more convenient than a qualification statement because it is easier to use the concatenated key than to move each part of the qualification statement to the SSA area during program execution. Using the segment's concatenated key is the equivalent of giving all the SSA in the path to the segment qualified on their keys.

For example, suppose that you wanted to answer this request:

What treatment did Joan Carter, patient number 07755, receive on March 3, 2009?

Using qualification statements, you would specify the following SSA with a GU call:

```
GU PATIENTb(PATN0bbbEQ07755)
   ILLNESSb(ILLDATEbEQ20090303)
   TREATMNTb
```

Using a C command code, you can satisfy the previous request by specifying the following SSA on a GU call:

```
GU ILLNESSb*C(0775520090303)
   TREATMNTb
```

If you need to qualify a segment by using a field other than the key field, use a qualification statement instead of the C command code.

Only one SSA with a concatenated key is allowed for each call. To return segments to your program in the path to the segment specified by the concatenated key, you can use unqualified SSA containing the D command code.

For example, if you want to return the PATIENT segment for Joan Carter to your I/O area, in addition to the ILLNESS segment, use the call:

```
GU PATIENTb*Db
   ILLNESSb*C(0775520090303)
```

You can use the C command code with the object segment for a Get call, but not for an ISRT call. The object segment for an ISRT call must be unqualified.

## D command code

You can use the D command code to retrieve or insert a sequence of segments in a hierarchic path with one call rather than retrieving or inserting each segment with a separate call. A call that uses the D command code is called a *path call*.

For your program to use the D command code, the P processing option must be specified in the PCB, unless your program uses command code D when processing DEDBs.

**Related reading:** For more information on using the P processing option, see the description of PSB generation in *IMS Version 13 System Utilities*.

## Retrieving a sequence of segments

When you use the D command code with retrieval calls, IMS places the segments in your I/O area. The segments in the I/O area are placed one after the other, left to right, starting with the first SSA you supplied. To have IMS return each segment in the path, you must include the D command code in each SSA. You can, however, include an intervening SSA without the D command code. You do not need to include the D command code on the last segment in the path, because IMS always returns the last segment in the path to your I/O area.

The D command code has no effect on the IMS retrieval logic. The only thing it does is cause each segment to be moved to your I/O area. The segment name in the PCB is the lowest-level segment that is retrieved or the last level that is satisfied in the call in the case of a GE (not-found) status code. Higher-level segments with the D command code are placed in the I/O area.

If IMS is unable to find the lowest segment your program has requested, it returns a GE (not-found) status code, just as it does if your program does not use the D command code and IMS is unable to find the segment your program has requested. This is true even if IMS reaches the end of the database before finding the lowest segment your program requested. If IMS reaches the end of the database without satisfying any levels of a path call, it returns a GB (end of database) status code. However, if IMS returns one or more segments to your I/O area (new segments for which there was no current position at the start of the current call), and if IMS is unable to find the lowest requested segment, IMS returns a GE status code, even if it has reached the end of the database.

The advantages of using the D command code are significant even if your program is not sure that it will need the dependent segment returned by D. For example, suppose that after examining the dependent segment, your program still needs to use it. Using the D command, your program has the segment if you need it, and your program is not required to issue another call for the segment.

For an example of the D command code, suppose your program has this request:

Compute the balance due for each of the clinic's patients by subtracting the payments received from the amount billed; print bills to be mailed to each patient.

To process this request for each patient, your program needs to know the patient's name and address, what the charges are for the patient, and the amount of payment the patient has made. Issue this call until your program receives a GE status code indicating that no more patient segments exist:

```
GN  PATIENTb*Db
    BILLINGb*Db
    PAYMENTbb
```

Each time you issue this call, your I/O area contains the patient segment, the billing segment, and the payment segment for a particular person.

## Inserting a sequence of segments

With ISRT calls, your program can use the D command code to insert a path of segments simultaneously. Your program need not include D for each SSA in the path. Your program just specifies D on the first segment that you want IMS to insert. IMS inserts the segments in the path that follow.

For example, suppose your program has this request:

Judy Jennison visited the clinic for the first time. Add a record that includes PATIENT, ILLNESS, and TREATMNT segments.

After building the segments in your I/O area, issue an ISRT call with the following SSA:

```
ISRT  PATIENTb*Db
      ILLNESSbb
      TREATMNTb
```

Not only is the PATIENT segment added, but the segments following the PATIENT segment, ILLNESS and TREATMNT, are also added to the database.

You cannot use the D command code to insert segments if a logical child segment in the path exists.

## F command code

You can use the F command code to start the search with the first occurrence of a certain segment type or to insert a new segment as the first occurrence in a chain of segments.

### Retrieving a segment as the first occurrence

You can use the F command code for GN and GNP calls. Using it with GU calls is redundant (and is disregarded) because GU calls can already back up in the database. When you use F, you indicate that you want the search to start with the first occurrence of the segment type you indicate under its parent in attempting to satisfy this level of the call.

You can use the F command code for GN and GNP calls to back up in the database. You can back up to the first occurrence of the segment type that has current position, or you can back up to a segment type that is before the current position in the hierarchy.

**Restriction:** The parent of the segment that you are backing up from must be in the same hierarchic path as the segment you are backing up to. IMS disregards F when you supply it at the root level or with a GU or GHU.

The search must start with the first occurrence of the segment type that you indicate under the parent. When the search at that level is satisfied, that level is treated as though a new occurrence of a segment has satisfied the search. This is true even when the segment that satisfies an SSA where F command code is specified as the same segment occurrence on which DL/I was positioned before the call was processed.

When a new segment occurrence satisfies an SSA, the position of all dependent segments is reset. New searches for dependent segments then start with the first occurrence of that segment type under its parent.

### Inserting a segment as the first occurrence

When you use F with an ISRT call, you are indicating that you want IMS to insert the segment you have supplied as the first segment occurrence of its segment type. Use F with segments that have either no key at all or a non unique key, and that have HERE specified on the RULES operand of the SEGM statement in the DBD. If

you specify HERE in the DBD, the F command code overrides this, and IMS inserts the new segment occurrence as the first occurrence of that segment type.

Using the F command code to override the RULES specification on the DBD applies only to the path (either logical or physical) that you are using to access the segment for the ISRT call. For example, if you are using the physical path to access the segment, the command code applies to the physical path but not to the logical path. For clarification of using command codes with the RULES specification, ask the database administrator at your installation.

For example, suppose that you specified RULES=HERE in the DBD for the TREATMNT segment. You want to satisfy this request:

Mary Martin visited the clinic today and visited a number of different doctors. Add the TREATMNT segment for Dr. Smith as the first TREATMNT segment for the most recent illness.

First you build a TREATMNT segment in your I/O area:

```
19930302ESEDRIXbbb0040SMITHbbbb
```

Then you issue an ISRT call with the following SSA. This adds a new occurrence of the TREATMNT segment as the first occurrence of the TREATMNT segment type among those with equal keys.

```
ISRT  PATIENTb(PATN0bbb=b06439)
      ILLNESSb*L
      TREATMNT*F
```

This example applies to HDAM or PHDAM root segments and to dependent segments for any type of database.

**Related reference:**

“GU/GHU call” on page 19

## G command code

You can use the G command code to indicate to IMS to skip randomization or the calling of the partition selection exit and search the database sequentially. While this command code can be used with other database types, it will affect the access of only HDAM/PHDAM, DEDB, and PHIDAM databases.

When accessing an HDAM/PHDAM, DEDB, or PHIDAM database that is accessed using a HALDB Partition Selection exit routine, and the records are not in sequence across partition boundaries, all keys in the requested range of a multiple qualification SSA might not be returned. If the first call to the database or command A is used, command code G can be used to sequentially read through the database until the SSA is satisfied.

## L command code

You can use the L command code to retrieve the last occurrence of a particular segment type or to insert a segment as the last occurrence of a segment type.

### Retrieving a segment as the last occurrence

The L command code indicates that you want to retrieve the last segment occurrence that satisfies the SSA, or that you want to insert the segment occurrence you are supplying as the last occurrence of that segment type. Like F, L simplifies



your programming because you can go directly to the last occurrence of a segment type without having to examine the previous occurrences with program logic, if you know that it is the last segment occurrence that you want. L can be used with GU or GHU, because IMS normally returns the first occurrence when you use a GU call. IMS disregards L at the root level.

Using L with GU, GN, and GNP indicates to IMS that you want the last occurrence of the segment type that satisfies the qualification you have provided. The qualification is the segment type or the qualification statement of the SSA. If you have supplied just the segment type (an unqualified SSA), IMS retrieves the last occurrence of this segment type under its parent.

For example, suppose you have this request using the medical hierarchy:

What was the illness that brought Jennifer Thompson, patient number 10345, to the clinic most recently?

In this example, assume that RULES=LAST is specified in the DBD for the database on ILLNESS. Issue this call to retrieve this information:

```
GU    PATIENTb(PATNObbb=b10345)
      ILLNESSb*L
```

The first SSA gives IMS the number of the particular patient. The second SSA asks for the last occurrence (in this case, the first occurrence chronologically) of the ILLNESS segment for this patient.

## Inserting a segment as the last occurrence

Use L with ISRT only when the segment has no key or a non-unique key, and the insert rule for the segment is either FIRST or HERE. Using the L command code overrides both FIRST and HERE for HDAM or PHDAM root segments and dependent segments in any type of database.

Using the L command code to override the RULES specification on the DBD applies only to the path (either logical or physical) that you are using to access the segment for the ISRT call. For example, if you are using the physical path to access the segment, the command code applies to the physical path but not to the logical path. For clarification of using command codes with the RULES specification, ask your database administrator.

## N command code

The N command code prevents you from replacing a segment on a path call. In conjunction with the D command code, it lets the application program to process multiple segments using one call. Alone, the D command code retrieves a path of segments in your I/O area. With the N command code, the D command code lets you distinguish which segments you want to replace.

For example, the following code only replaces the TREATMNT segment.

```
GHU    PATIENT*D(PATNObbb=b06439)
      ILLNESSb*D(ILLDATEb=19930301)
      TREATMNT
REPL   PATIENT*N(PATNObbb=b06439)
      ILLNESSb*N(ILLDATEb=19930301)
      TREATMNT
```

**Restriction:** If you use D and N command codes together, IMS retrieves the segment but does not replace it.

The N command code applies only to REPL calls, and IMS ignores it if you include the code in any other call.

## O command code

You can use the O command code to specify a SSA qualification with the position and length of the target data instead of a DBD-defined field name.

This command code is valid for full function database types (HDAM, HIDAM, PHIDAM, and PHDAM) and Fast Path DEDBs.

This command code is supported for the following DL/I calls:

- GU SSA
- GHU SSA
- GN SSA
- GNP SSA
- GHNP SSA
- ISRT SSA

When command code O is specified, the SSA qualification can contain either normal field names or the starting offset and length of the data that you want to retrieve.

You must specify the offset and length as two 4-byte binary values in place of the usual 8-byte character value that is used to specify a field name. The starting position for the offset is 1 and the offset is relative to the physical start of the segment definition. The maximum length that is supported is the maximum segment size for the database type. The minimum length is 1.

For example, a segment might have several fields defined in the DBD with the following offsets and lengths:

Field	Offset	Length
Labname	1	5
Street	10	20
State	30	2

The application program has a COBOL copy book with the following map:

Field	Offset	Length
Labname	1	5
Type	6	3
Street	10	20
State	30	2

The database contains two records with the following data:

```
I          111111112222222233
I 12345678901234567901235678901
I
Segment #1 I SVL  DEV 555 BAILEY AVE  CA
Segment #2 I ARC  RSC 650 HARRY RD    CA
I
```

You can specify a GU call with the O command code in the following format to retrieve data without needing the fields to be specified in the DBD. The following

example demonstrates how to specify the offset and length values in a DFSDDLTO test application using hexadecimal edit mode:

```
00000000
GU IBMLABS*0 ('00010005'x=SVL )
00000000
GU IBMLABS*0 ('00010005'x=ARC )
00030000
GU IBMLABS*0 ('00000002'x=CA)
00000000
GU IBMLABS*0 ('000060003'x=DEV)
```

In the first GU call, the offset is 1 and the length of the target data is 5.

## P command code

Ordinarily, IMS sets parentage at the level of the lowest segment that is accessed during a call. To set parentage at a higher level, you can use the P command code in a GU, GN, or GNP call.

The parentage that you set with P works just like the parentage that IMS sets: it remains in effect for subsequent GNP calls, and is not affected by ISRT, DLET, or REPL calls. It is only affected by GNP if you use the P command code in the GNP call. Parentage is canceled by a subsequent GU, GHU, GN, or GHN.

Use the P command code at only one level of the call. If you mistakenly use P in multiple levels of a call, IMS sets parentage at the lowest level of the call that includes P.

If IMS cannot fully satisfy the call that uses P (for example, IMS returns a GE status code), but the level that includes P is satisfied, P is still valid. If IMS cannot fully satisfy the call including the level that contains P, IMS does not set any parentage. You would receive a GP (no parentage established) if you then issued a GNP.

If you use P with a GNP call, IMS processes the GNP call with the parentage that was already set by preceding calls. IMS then resets parentage with the parentage you specified using P after processing the GNP call.

For example, if you want to send a current bill to all of the patients seen during the month, the determining value is in the ILLNESS segment. You want to look at only patients whose ILLNESS segments have dates after the first of the month. For patients who have been to the clinic during the month, you need to look at their addresses and the amount of charges in the BILLING segment so that you can print a bill. For this example, assume the date is March 31, 1993. Issue these two calls to process this information:

```
GN  PATIENTb*PD
    ILLNESSb(ILLDATEb>=19930301)
GNP BILLINGbb
```

After you locate a patient who has been to the clinic during the month, you issue the GNP call to retrieve that patient's BILLING segment. Then you repeat the GN call to find each patient who has been to the clinic during the month, until IMS returns a GB status code.

## Q command code

Use the Q command code if you want to prevent another program from updating a segment until your program reaches a commit point. The Q command code tells

IMS that your application program needs to work with a segment and that no other tasks can be allowed to modify the segment until the program has finished.

This means that you can retrieve segments using the Q command code, then retrieve them again later, knowing that they have not been altered by another program. (You should be aware, however, that reserving segments for the exclusive use of your program can affect system performance.)

You can use the Q command code in batch programs in a data-sharing environment and in CICS and IMS online programs. IMS ignores Q in non-data sharing batch programs.

## Limiting the number of database calls

For full function, before you use the Q command code in your program, you must specify a MAXQ value during PSBGEN. This establishes the maximum number of database calls (with Q command codes) that you can make between sync points.

**Related reading:** For information on PSBGEN, see *IMS Version 13 System Utilities*.

Fast Path does not support the MAXQ parameter. Consequently in Fast Path, you can issue as many database calls with Q command codes as you want.

## Using segment lock class

For full function, when you use the Q command code to retrieve a segment, you specify the letter Q followed by a letter (A-J), designating the lock class of that segment (for example, QA). If the lock class is not a letter (A-J), IMS returns the status code GL.

Fast Path supports the Q command code alone, without a letter designating the lock class. However, for consistency between Fast Path and full function, Fast Path treats the Q command code as a 2-byte string, where the second byte must be a letter (A-J). If the second byte is not a letter (A-J), IMS returns the status code AJ.

For example, suppose a customer wants to place an order for items 1, 2, and 3, but only if 50 item 1's, 75 item 2's, and 100 item 3's are available. Before placing this order, the program must examine all three item segments to determine whether an adequate number of each item is available. You do not want other application programs to change any of the segments until your program has determined this and, if possible, placed the order.

To process this request for full function, your program uses the Q command code when it issues the Get calls for the item segments. When you use the Q command code in the SSA, you assign a lock class immediately following the command code in the SSA.

```
GU  PART X
    ITEM 1   *QA
GU  PART X
    ITEM 2   *QA
GU  PART X
    ITEM 3   *QA
```

**Exception:** For Fast Path, the second byte of the lock class is not interpreted as lock class 'A'.

After retrieving the item segments, your program can examine them to determine whether an adequate number of each item are on hand to place the order. Assume 100 of each item are on hand. Your program then places the order and updates the database accordingly. To update the segment, your program issues a GHU call for each segment and follows it immediately with a REPL call:

```
GHU  ITEM 1
REPL  ITEM 1 with the value 50
GHU  ITEM 2
REPL  ITEM 2 with the value 25
GHU  ITEM 3
REPL  ITEM 3 with the value 0
```

## Using the DEQ call with the Q command code

When you use the Q command code and the DEQ call, you reserve and release segments.

For full function, to issue a DEQ call against an I/O PCB to release a segment, you place the letter designating the segment's lock class in the first byte of an I/O area. Then, you issue the DEQ call with the name of the I/O area that contains the letter.

A DEDB DEQ call is issued against a DEDB PCB. Because Fast Path does not support lock class, a DEDB DEQ call does not require that a lock class be specified in the I/O area.

**Restriction:** The EXEC DL/I interface does not support DEDB DEQ calls, because EXEC DL/I disallows a PCB for DEQ calls.

## Retrieving segments with full-function DEQ calls

The DEQ call releases all segments that are retrieved using the Q command code, except:

- Segments modified by your program, until your program reaches a commit point
- Segments required to keep your position in the hierarchy, until your program moves to another database record
- A class of segments that has been locked again as another class

If your program only reads segments, it can release them by issuing a DEQ call. If your program does not issue a DEQ call, IMS releases the reserved segments when your program reaches a commit point. By releasing them with a DEQ call before your program reaches a commit point, you make them available to other programs more quickly.

## Retrieving buffers with Fast Path DEQ calls

DEQ calls cause Fast Path to release a buffer that satisfies one of the conditions:

- The buffer has not been modified, or the buffer does not protect a valid root position.
- The buffer has been protected by a Q command code.

Fast Path returns an FW status code when no buffers can be released for a DEQ call.

Any CI locking or segment-level locking performed with a Q command code is protected from other application programs until a DEQ call is issued or a commit point is reached.

### **Considerations for root and dependent segments (full function only)**

If you use the Q command code on a root segment, other programs in which the PCB does not have update capability can access the database record. Programs in which the PCB has update capability cannot access any of the segments in that database record. If you use the Q command code on a dependent segment, other programs can read the segment using one of the Get calls without the hold. If your program accesses shared databases, and if any of the segments in that block are reserved with the Q command code, application programs in other IMS systems cannot update anything in that block. The Q command code does not hold segments from one step of a conversation to another.

**Related Reading:** For more information on the relationship between the Q command code and the DEQ call, see the topic "Reserving Segments for the Exclusive Use of Your Program" in *IMS Version 13 Application Programming*.

## **U command code**

As IMS satisfies each level in a retrieval or ISRT call, a position on the segment occurrence that satisfies that level is established. The U command code prevents position from being moved from a segment during a search of its hierarchic dependents.

If the segment has a unique sequence field, using this code is equivalent to qualifying the SSA so that it is equal to the current value of the key field. When a call is being satisfied, if the position is moved above the level that the U code was issued at, the code has no effect for the segment type whose parent changed position.

U is especially useful when unkeyed dependents or non-unique keyed segments are being processed. The position on a specific occurrence of an unkeyed or non-unique keyed segment can be held by using this code.

**Example:** Suppose you want to find out about the illness that brought a patient named Mary Warren to the clinic most recently, and about the treatments she received for that illness. The following figure shows the PATIENT, ILLNESS, and TREATMNT segments for Mary Warren.

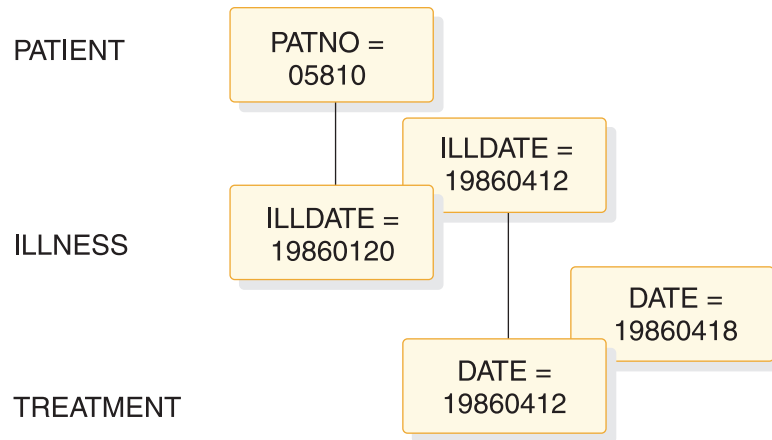


Figure 3. U command code example

To retrieve this information, retrieve the first ILLNESS segment and the TREATMNT segments associated with that ILLNESS segment. To retrieve the most recent ILLNESS segment, you can issue the following GU call:

```
GU  PATIENTb(PATNObbb=b05810
    ILLNESSb*L
```

After this call, IMS establishes a position at the root level on the PATIENT segment with the key 05810 and on the last ILLNESS segment. Because other ILLNESS segments with the key 19860412 may exist, you can think of this one as the most recent ILLNESS segment. You might want to retrieve the TREATMNT segment occurrences that are associated with that ILLNESS segment. You can do this by issuing the GN call below with the U command code:

```
GN  PATIENTb*U
    ILLNESSb*U
    TREATMNT
```

In this example, the U command code indicates to IMS that you want only TREATMNT segments that are dependents of the ILLNESS and PATIENT segments on which IMS has established position. Issuing the above GN call the first time retrieves the TREATMNT segment with the key of 19860412. Issuing the GN call the second time retrieves the TREATMNT segment with the key 19860418. If you issue the call a third time, IMS returns a not-found status code. The U command code tells IMS that, if it does not find a segment that satisfies the lower qualification under this parent, it cannot continue looking under other parents. If the U command code was not in the PATIENT SSA, the third GN call causes IMS to move forward at the root level in an attempt to satisfy the call. If you supply a U command code for a qualified SSA, IMS ignores the U.

If used in conjunction with command code F or L, the U command code is disregarded at the level and all lower levels of SSA for that call.

## V command code

Using the V command code on an SSA is similar to using a U command code in that SSA and all preceding SSA. Specifying the V command code for a segment level tells IMS that you want to use the position that is established at that level and above as a qualification for the call.

Using the V command code is analogous to qualifying your request with a qualified SSA that specifies the current IMS position.

For example, suppose that you wanted to answer this request:

Did Joan Carter, patient number 07755, receive any treatment on March 3, 2009?

Using a qualified SSA, specify the following call:

```
GU    PATIENTb(PATNObbb=b07755)
      ILLNESSb(ILLDATEb=20090303)
      TREATMNT
```

If you have position established on the PATIENT segment for patient number 07755 and on the ILLNESS segment for March 3, 2009, you can use your position to retrieve the TREATMNT segments in which you are interested. You do this by specifying the V command code as follows:

```
GN    PATIENTbb
      ILLNESSbb*V
      TREATMNT
```

Using the V command code for a call is like establishing parentage and issuing a subsequent GNP call, except that the V command code sets the parentage for the call it is used with, not for subsequent calls. For example, to satisfy the previous request, you could have set parentage at the ILLNESS segment level and issued a GNP to retrieve any TREATMNT segments under that parent. With the V command code, you specify that you want the ILLNESS segment to be used as parentage for that call.

You can specify the V command code for any parent segment. If you use the V command code with a qualified SSA, it is ignored for that level and for any higher level that contains a qualified SSA.

## NULL command code

The null command code (-) enables you to reserve one or more positions in a SSA in which a program can store command codes, if they are needed during program execution.

For example, reserve position for two command codes as follows:

```
GU    PATIENTb*--(PATNObbb=b07755)
      ILLNESSbILLDATEb=19930303)
      TREATMNT
```

Using the null command code lets you use the same set of SSAs for more than one purpose. However, dynamically modifying the SSA makes debugging more difficult.

## DEDB command codes for DL/I

The M, R, S, W, and Z command codes are only used with a DEDB.

### Sample application program

The following examples are based on one sample application program—the recording of banking transactions for a passbook (savings account) account. The transactions are written to a database as either posted or unposted, depending on whether they were posted to the customer's passbook.



For example, when Bob Emery does business with the bank but forgets to bring in his passbook, an application program writes the transactions to the database as unposted. The application program sets a subset pointer to the first unposted transaction, so it can be easily accessed later. The next time Bob remembers to bring in his passbook, a program posts the transactions.

The program can directly retrieve the first unposted transaction using the subset pointer that was previously set. After the program has posted the transactions, it sets the subset pointer to 0. An application program that updates the database later will be able to tell that no unposted transactions exist. The following figure summarizes the processing that is performed when the passbook is unavailable and when it is available.

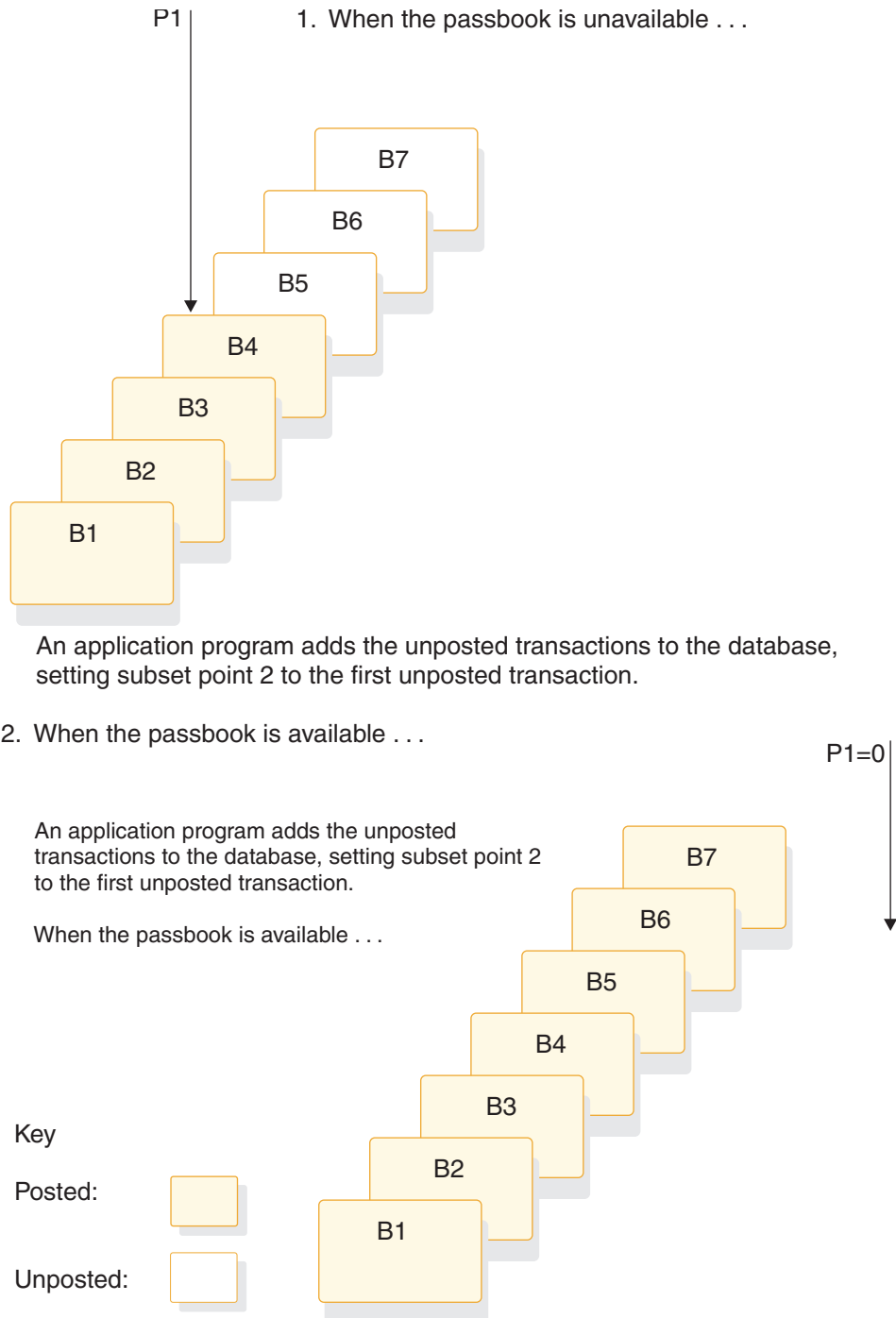


Figure 4. Processing for the passbook example

### M command code

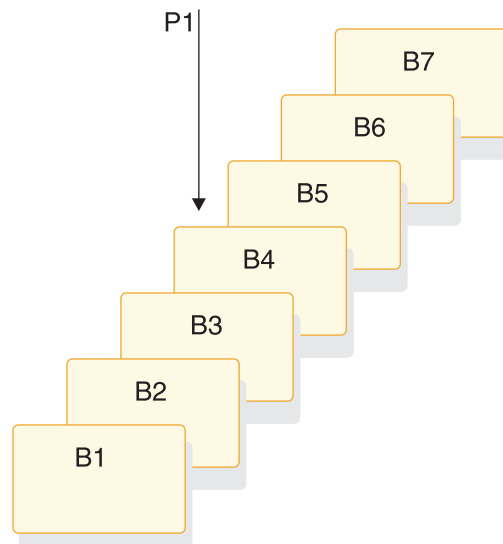
To move the subset pointer forward to the next segment after your current position, your program issues a call with the M command code.

Using the passbook account example, suppose that you want to post some, but not all, of the transactions, and that you want the subset pointer to be set to the first unposted transaction. The following command sets subset pointer 1 to segment B6, as shown in the figure below.

```
GU      Abbbbbbb(AKEYbbb  
        Bbbbbbbb*R1M1
```

If the current segment is the last in the chain, and you use an M command code, IMS sets the pointer to 0.

Before the call:



After the call:

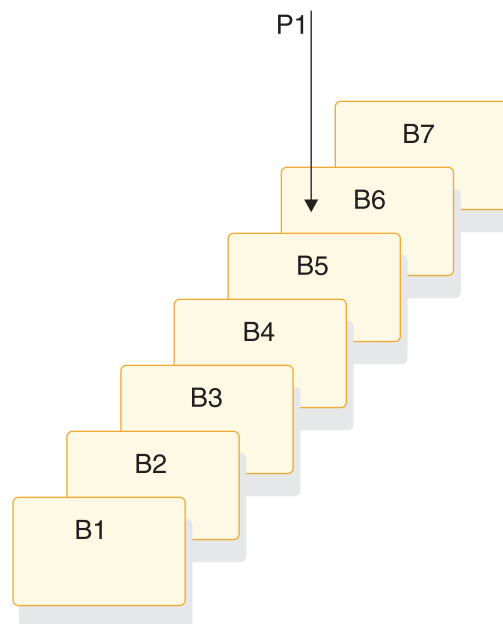


Figure 5. Moving the subset pointer to the next segment after your current position

### R command code

To retrieve the first segment occurrence in the subset, your program issues a Get call with the R command code. The R command code does not set or move the pointer. It indicates to IMS that you want to establish position on the first segment

occurrence in the subset. The R command code is like the F command code, except that the R command code applies to the subset instead of to the entire segment chain.

Using the passbook account example, suppose that Bob Emery visits the bank and brings his passbook; you want to post all of the unposted transactions. Because subset pointer 1 was previously set to the first unposted transaction, your program uses the following call to retrieve that transaction:

```
GU  Abbbbbbb(AKEYbbbb=bA1)
    Bbbbbbbb*R1
```

As shown by the following figure, this call retrieves segment B5. To continue processing segments in the chain, you can issue GN calls as you would if you were not using subset pointers.

If the subset does not exist (subset pointer 1 has been set to 0), IMS returns a GE status code, and your position in the database will be immediately following the last segment in the chain. Using the passbook example, the GE status code tells you that no unposted transactions exist.

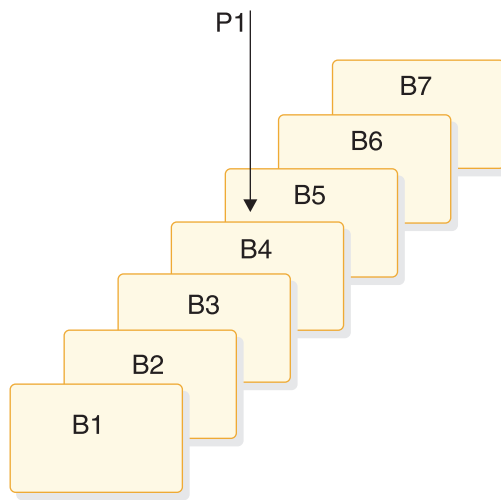


Figure 6. Retrieving the first segment in a chain of segments

You can specify only one R command code for each SSA. If you use more than one R in a SSA, IMS returns an AJ status code to your program.

You can use R with other command codes, except F and Q. Other command codes in a SSA take effect after the R command code has been processed, and after position has been successfully established on the first segment in the subset. If you use the L and R command codes together, the last segment in the segment chain is retrieved. (If the subset pointer that was specified with the R command code, IMS returns a GE status code instead of the last segment in the segment chain.) Do not use the R and F command codes together. If you do, you will receive an AJ status code. The R command code overrides all insert rules, including LAST.

### S command code

To set a subset pointer unconditionally, regardless of whether it is already set, your program issues a call with the S command code.

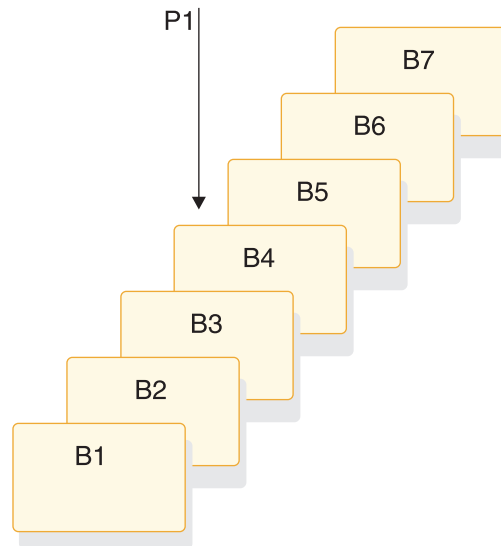
When your program issues a call that includes the S command code, IMS sets the pointer to your current position.

For example, to retrieve the first B segment occurrence in the subset defined by subset pointer 1 and to reset pointer 1 at the next B segment occurrence, you would issue the following commands:

```
GU  Abbbbbbb(AKEYbbb=bB1)
    Bbbbbbbb*R1
GN  Bbbbbbbb*S1
```

After you issue this call, instead of pointing to segment B5, subset pointer 1 points to segment B6, as shown in the following figure.

Before the call:



After the call:

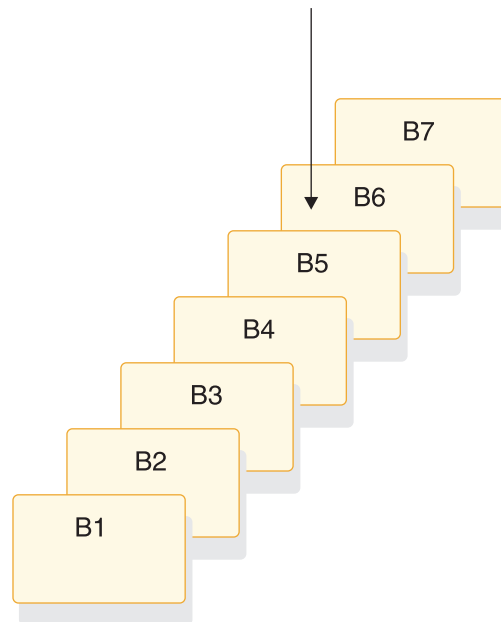


Figure 7. Unconditionally setting the subset pointer to your current position

## W command code

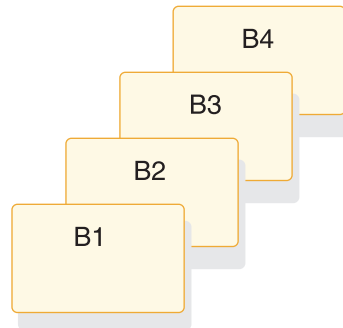
Like the S command code, the W command code sets the subset pointer conditionally. Unlike the S command code, the W command code updates the subset pointer only if the subset pointer is not already set to a segment.

For example, using the passbook example, suppose that Bob Emery visits the bank and forgets to bring his passbook. You add the unposted transactions to the database. You want to set the pointer to the first unposted transaction, so that later, when you post the transactions, you can immediately access the first one. The following call sets the subset pointer to the transaction you are inserting if it is the first unposted one.

```
ISRT      Abbbbbbb(AKEYbbbb=bA1)
          Bbbbbbbb*W1
```

As shown by the following figure, this call sets subset pointer 1 to segment B5. If unposted transactions already exist, the subset pointer is not changed.

Before the call:



After the call:

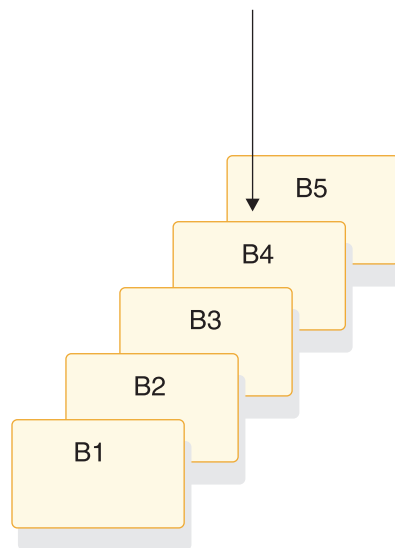


Figure 8. Conditionally setting the subset pointer to your current position

### Z command code

The Z command code sets the value of the subset pointer to 0. After your program issues a call with the Z command code, the pointer is no longer set to a segment, and the subset defined by that pointer no longer exists.

IMS returns a status code of GE to your program if you try to use a subset pointer having a value of 0.

For example, using the passbook example, suppose that you used the R command code to retrieve the first unposted transaction. You then process the chain of segments, posting the transactions. After posting the transactions and inserting any new ones into the chain, use the Z command code to set the subset pointer to 0 as shown in the following call:

ISRT      Abbbbbbb(AKEYbbbb=bA1)  
 Bbbbbbbb\*Z1

After this call, subset pointer 1 is set to 0, which indicates to a program that subsequently updates the database that no unposted transactions exist.

## Relationship between calls, AIBs, and PCBs

The following table shows the relationship of calls to full function (FF), main storage database (MSDB), data entry database (DEDB), I/O, and general sequential access method (GSAM) PCBs.

Table 53. Call relationship to PCBs

CALL	AIB	FF PCBs	MSDB PCBs	DEDB PCBs	I/O PCBs	GSAM PCBs
APSB	X					
CHKP	X				X	
CIMS	X					
CLSE	X					X
DEQ	X			X	X	
DLET	X	X	X	X		
DPSB	X					
FLD	X		X	X		
GHN	X	X	X	X		
GHNP	X	X	X	X		
GHU	X	X	X	X		
GMSG	X					
GN	X	X	X	X	X	X
GNP	X	X	X	X		
GSCD <sup>1</sup>	X	X	X	X	X	
GU	X	X	X	X	X	X
ICMD	X					
INIT	X				X	
INQY	X					
ISRT	X	X	X	X	X	X
LOG	X				X	
OPEN	X					X
PCB <sup>2</sup>						
POS	X			X		
RCMD	X					
REPL	X	X	X	X		
ROLB	X				X	
ROLL <sup>2</sup>						
ROLS	X	X			X	
SETS/SETU	X				X	
SNAP <sup>3</sup>	X	X	X	X	X	



Table 53. Call relationship to PCBs (continued)

CALL	AIB	FF PCBs	MSDB PCBs	DEDB PCBs	I/O PCBs	GSAM PCBs
STAT <sup>4</sup>	X	X				
SYNC	X				X	
TERM <sup>2</sup>						
XRST	X				X	

**Note:**

1. GSCD is a Product-sensitive Programming Interface.
2. The PCB, ROLL, and TERM calls do not have an associated PCB.
3. SNAP is a Product-sensitive Programming Interface.
4. STAT is a Product-sensitive Programming Interface.

## DL/I test program (DFSDDLTO) reference

DFSDDLTO 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 DFSDDLTO using any PSB, including those that use an IMS-supported language. You can also use DFSDDLTO as a general-purpose database utility program.

The functions that DFSDDLTO provides include:

- Issuing any valid DL/I call against any database using:
  - Any segment search argument (SSA) or PCB, or both

**Important:** Calls that use a PCB must have specified LIST=YES in the PSB.

- 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 DFSDDLTO control statements.
- Sending messages to the z/OS system console (with or without a reply).
- Repeating each call up to 9,999 times.

## Control statements

DFSDDLTO processes control statements to control the test environment. DFSDDLTO can issue calls to IMS full-function databases and Fast Path databases, as well as DC calls.

When you are coding the DFSDDLTO control statements, keep these items in mind:

- You must fill in column 1 of each control statement. If column 1 is blank, the statement type defaults to the prior statement type. DFSDDLTO 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 DFSDDLTO functions.
- All codes and fields in the DFSDDLTO statements must be left justified followed by blanks, unless otherwise specified.

## Control statement guidelines

The order of control statements is critical in constructing a successful call. To avoid unpredictable results, follow these guidelines:

- If you are using STATUS and OPTION statements, place them somewhere before the calls that are to use them.
- Both types of COMMENT statements are optional but, if present, must appear before the call they document.
- You must code CALL FUNCTION statements and any required SSAs consecutively without interruption.
- CALL DATA statements must immediately follow the last continuation, if any, of the CALL FUNCTION statements.
- COMPARE statements are optional but must follow the last CALL (FUNCTION or DATA) statement.
- 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 DFSDDLTO control statements.

**Exception:** IGNORE statements are the only exception to this rule.

- Use IGNORE statements (N or period (.)) to override any statement, regardless of its position in the input stream. You can use IGNORE statements in either SYSIN or SYSIN2 input streams.

### Related reference:

“SYSIN DD statement” on page 274

“SYSIN2 DD statement” on page 274

“PUNCH CTL statement” on page 267

## ABEND statement

The ABEND statement causes IMS to issue an abend and terminate DFSDDLTO.

The following table shows the format of the ABEND statement.

Table 54. 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-----<
0 DUMP                                                    22100010
```

No dump will be produced; OPTION statement provided requests NODUMP.

```
|-----1-----2-----3-----4-----5-----6-----7-----<
0 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, z/OS commands, checkpoint IDs) required by the DL/I call specified in the CALL FUNCTION statement.

## Examples of DFSDDLTO 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-----<
0 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. DFSDDLTO 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-----<
0 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
/*

```

### CALL FUNCTION statement

The following table gives the format for CALL FUNCTION statements, including the column number, function, code, and description.

This is the preferred format when you are not working with column-specific SSAs.

Table 55. 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 hexadecimal characters allowed is 1-F.
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, with or without leading zeros.
9	Reserved	b	
10-13	Identifies DL/I call function	b	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	b	

Table 55. CALL FUNCTION statement (continued)

Column	Function	Code	Description
16-23 or	SSA name	xxxxxxx	Must be left-justified.
16-23 or	Token	xxxxxxx	Token name (SETS/ROLS).
16-23 or	MOD name	xxxxxxx	Modname (PURG+ISRT).
16-23 or	Subfunction	xxxxxxx	nulls, DBQUERY, FIND, ENVIRON, PROGRAM (INQY).
16-19 and	Statistics type	xxxx	DBAS/DBES-OSAM or VBAS/VBES-VSAM (STAT). <sup>2</sup>
20 or	Statistics format	x	F - Formatted U- Unformatted S - Summary.
16-19	SETO ID <sup>1</sup>	SETx	Where x is 1, 2, or 3. Specified on SETO and CHNG calls as defined in Note.
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.

Table 55. CALL FUNCTION statement (continued)

Column	Function	Code	Description
73-80	Sequence indication	nnnnnnnn	For SYSIN2 statement override.
25-32	OTMA descriptor name	xxxxxxx	8-byte character field (ICAL).
34-39	The wait time for the synchronous call to be processed	nnnnnn	6-byte character field with a range from 1 to 999999 (ICAL).
41-45	The input message length	nnnnn	The length of the input data in the request area (ICAL).
47-51	The response area length	nnnnn	The length of the response area for the output message (ICAL).

**Note:**

1. SETO CALL:

The SETO ID (SET1, SET2, or SET3) is required on the SETO call if DFSDDL0 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, DFSDDL0 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 DFSDDL0 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, DFSDDL0 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 DFSDDL0.

2. STAT is a Product-sensitive Programming Interface.

This 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 a 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 a 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 reference:**

“CALL FUNCTION statement with column-specific SSAs”

**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. The following table gives the format for the CALL FUNCTION statement with column-specific SSAs.

*Table 56. CALL FUNCTION statement (column-specific SSAs)*

Column	Function	Code	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 56. CALL FUNCTION statement (column-specific SSAs) (continued)

Column	Function	Code	Description
37	Reserved	b	Separator field.
38- <i>nn</i>	Field value	<b>nnnnn</b>	Required if segment is qualified. <b>Note:</b> Do not use '5D' or ')' in field value.
<i>nn</i> +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	<b>nnnnnnnn</b>	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 a SSA is 290 bytes, and the maximum number of SSAs for this program is 15, which is the same as the IMS limit.

**Related reference:**

“CALL FUNCTION statement” on page 238

**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 through 13, and code the necessary data in columns 16 through 71. You can continue data by entering a nonblank character in column 72. On the continuation statement, columns 1 through 15 are blank and the data resumes in column 16. The following table shows the format for a CALL DATA statement.

Table 57. 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 through 65) to be propagated through remaining I/O area. <b>Note:</b> This must be the last data statement and cannot be continued.
4	Format options	<b>b</b>	Not a variable-length segment.
		<b>V</b>	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.



Table 57. CALL DATA statement (continued)

Column	Function	Code	Description
		<b>M</b>	For statements describing the second through the last variable-length segments, LL field is added before the segment data.
		<b>P</b>	For the first statement describing a fixed-length segment in a path call.
		<b>Z</b>	For message segment, LLZZ field is added before the data.
		<b>U</b>	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	<b>nnnn</b>	This value must be right justified but need not contain leading zeros. If you do not specify a length, DFSDDL0 will use the number of DATA statements read multiplied by 56 to derive the length.
9	Reserved	<b>b</b>	
10-13	Identifies CALL DATA statement	<b>DATA</b>	Identifies this as a DATA statement.
14-15	Reserved	<b>b</b>	
16-71	Data area	<b>xxxx</b>	Data that goes in the I/O area.
	or		
16-23	Checkpoint ID		Checkpoint ID (SYNC).
	or		
16-23	Destination name		Destination name (CHNG).
	or		
16	DEQ option		DEQ options (A,B,C,D,E,F,G,H,I, or J).
72	Continuation column	<b>b</b>	If no more continuations for this segment.
		<b>x</b>	If more data for this segment or more segments.
73-80	Sequence indication	<b>nnnnnnnn</b>	For SYSIN2 statement override.

When inserting variable-length segments or including variable-length data for a CHPK 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, concatenation of logical child and 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.

The following table shows the format for an OPTION DATA statement, including the column number, function, code, and description.

*Table 58. 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.
72	Continuation column	b	If no more continuations for options.
		x	If more option data exists in following statement.

Table 58. OPTION DATA statement (continued)

Column	Function	Code	Description
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.

The following table shows the format for a FEEDBACK DATA statement, including the column number, function, code, and description.

Table 59. 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, DFSDDL0 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.

### DL/I call functions

The following table shows the DL/I call functions supported in DFSDDL0 and which ones require data statements.

Table 60. DL/I call functions.

Call	AIB Support	PCB Support	Data Stmt <sup>1</sup>	Description
CHKP	yes	yes	R	Checkpoint.
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.

Table 60. DL/I call functions (continued).

Call	AIB Support	PCB Support	Data Stmt <sup>1</sup>	Description
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 <sup>2</sup>	Get Hold Next.
GHNP	yes	yes	O <sup>2</sup>	Get Hold Next in Parent.
GHU	yes	yes	O <sup>2</sup>	Get Hold Unique.
GMSG <sup>3</sup>	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 <sup>2</sup>	Get Next segment.
GNP	yes	yes	O <sup>2</sup>	Get Next in Parent.
GU	yes	yes	O <sup>2</sup>	Get Unique segment.
GUR	yes	no	R	Get Unique Record from the IMS catalog database. <b>Tip:</b> Specify LCASE=C on the OPTION statement to make the records, which are returned as XML instance documents, more readable.
ICAL	yes	no	R	IMS Call enables an application program that runs in the IMS TM environment to send a synchronous request for data or services to a non-IMS application program or service that runs in a z/OS or distributed environment.
ICMD <sup>3</sup>	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.

Table 60. DL/I call functions (continued).

Call	AIB Support	PCB Support	Data Stmt <sup>1</sup>	Description
INQY <sup>3</sup>	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.
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 <sup>3</sup>	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.
RLSE	yes	yes	N	Release all locks held by an application that are for unmodified data.
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 (DFSDDLTO 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.

Table 60. DL/I call functions (continued).

Call	AIB Support	PCB Support	Data Stmt <sup>1</sup>	Description
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.
SNAP <sup>4</sup>	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 <sup>5</sup>	yes	yes	O	The STAT call retrieves statistics on the IMS system. This call must reference only full-function DB PCBs. Statistics type is coded in columns 16-19 of the CALL FUNCTION statement.  <b>DBAS</b> For OSAM database buffer pool statistics.  <b>VBAS</b> For VSAM database subpool statistics. Statistics format is coded in column 20 of the CALL FUNCTION statement.  <b>F</b> For the full statistics to be formatted if F is specified, the I/O area must be at least 360 bytes.  <b>U</b> For the full statistics to be unformatted if U is specified, the I/O area must be at least 72 bytes.  <b>S</b> 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

The following examples show how to use the 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,XTTU=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.





**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 > G131G131G131G131G131G131G131G131G131G131G131G*
      CONT 131G131G131G131G131G131G131G131G131G131G131G131G131G131G*
      CONT 00G400G400G400G400G400G400G400G400G400G400G400G400G400G400G400G400 *
      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)
```

**GUR Call:** Use a CALL FUNCTION statement to contain the GUR function and a DATA statement to specify the maximum size of the output area for the returned XML document.

```
|---+---1---+---2---+---3---+---4---+---5---+---6---+---7---+---<
0 LCASE=C
S1111 1 1 1 1DFSCAT00 AIB
L U0001 GUR HEADER (RHDRSEQ EQDBD DBOHIDK5)
L Z9999 DATA
```

The following table shows the key lines and elements in the example of the GUR call:

*Table 61. Explanation of the example*

Line in the example	Explanation
0 LCASE=C	Specifies that DFSDDLTO uses character representation and not hexadecimal representation for the XML output. Without character representation, you cannot read the returned XML document.
S1111 1 1 1 1DFSCAT00 AIB	Specifies that DFSDDLTO uses the AIB interface and the DB PCB name is DFSCAT00, which is the system-defined catalog.
L U0001 GUR HEADER	Specifies that IMS is to issue one GUR call. The SSA contains the key field RHDRSEQ, which is used to find a DBD that is named DBOHIDK5.
L Z9999 DATA	Specifies that DFSDDLTO is to use the maximum data output area, which is 9999 bytes.

If the GUR call returns an XML document that is too large to fit into the output area that is specified by the DATA statement, you must modify the GUR call so that it is repeated. You can repeat the GUR call in one of two ways:

- Set the repeat count on the GUR call (columns 5-8) to the number of times to repeat the call, which is the recommended way. In the following example, U0002 specifies that IMS is to issue two GUR calls:

```
|
| |---+-----1-----+-----2-----+-----3-----+-----4-----+-----5-----+-----6-----+-----7-----+-----<
| L U0002 GUR  HEADER (RHDRSEQ ==PSB      BMP255  )
| L Z9999 DATA
```

- Use multiple GUR calls:

```
|
| |---+-----1-----+-----2-----+-----3-----+-----4-----+-----5-----+-----6-----+-----7-----+-----<
| L U0001 GUR  HEADER (RHDRSEQ ==PSB      BMP255  )
| L Z9999 DATA
| L U0001 GUR  HEADER (RHDRSEQ ==PSB      BMP255  )
| L Z9999 DATA
```

Either method produces the same results.

**ICAL Call:** Use a CALL FUNCTION statement to contain the ICAL function. Use a CALL DATA statement to contain the message to pass from the IMS application to the program that is specified in the IMS OTMA descriptor.

The following example demonstrates how to send a synchronous callout request message to a destination named DESCRPTR with 45 bytes of request data and expect 100 bytes of response data to be returned in a timeout value of 500 (or 5 seconds).

```
|---+-----1-----+-----2-----+-----3-----+-----4-----+-----5-----+-----6-----+-----7-----+-----<
L          ICAL  SENDRCV DESCRPTR 000500 00045 00100
L          DATA HELLO OUT THERE. THIS IS A MESSAGE FROM IMS.
```

**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 30022222222222222      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 through 8 blank and put data in columns 16 through 71. To continue data, put a nonblank character in column 72, and the continued data in columns 16 through 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 through 8 must contain the length of the segment data. The length in columns 5 through 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 through 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 through 8. If the segment is longer than 56 bytes, put a nonblank character in column 72, and continue data on the next statement. 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 through 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 through 8
- The keyword DATA in columns 10 through 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 DFSDDLTO 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 through 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
```

**RLSE Call:** Use a CALL FUNCTION statement to contain the RLSE function.

```
|-----1-----2-----3-----4-----5
L          RLSE
```

**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   PRT0=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   PRT0=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   PRT0=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 topic 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 2222222                10103212
```

**This topic 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 Version 13 Application Programming 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

```

**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

```

## DFSDDLTO call functions

The DFSDDLTO call functions were created for DFSDDLTO. They do not represent "valid" IMS calls and are not punched as output if DFSDDLTO encounters them while a CTL (PUNCH) statement is active.

The following table shows the special call functions of the CALL FUNCTION statement. Descriptions and examples of these special functions follow.

*Table 62. CALL FUNCTION statement with DFSDDLTO 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.

Table 62. CALL FUNCTION statement with DFSDDLTO call functions (continued)

Column	Function	Code	Description
		<b>nnnn</b>	'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	<b>STAKb</b>	Stack control statements for later execution.
		<b>ENDb</b>	Stop stacking and begin execution.
		<b>SKIPb</b>	Skip statements until START function is encountered.
		<b>START</b>	Start processing statements again.
73-80	Sequence indication	<b>nnnnnnnn</b>	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 DFSDDLTO 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 DFSDDLTO to process. These functions are normally read from SYSIN2 and provide a temporary override to an established SYSIN input stream. DFSDDLTO reads all control statements occurring between the SKIP and START statements, but takes no action. When DFSDDLTO encounters the START statement, it reads and processes the next statement normally.

#### Related reference:

"PUNCH CTL statement" on page 267

## COMMENT statement

Use the COMMENT statement to print comments in the output data.

The two types of COMMENT statements, conditional and unconditional are described. The following table shows the format of the COMMENT statement.

Table 63. COMMENT statement

Column	Function	Code	Description
1	Identifies control statement	<b>T</b>	Conditional comment statement.
		<b>U</b>	Unconditional comment statement.
2-72	Comment data		Any relevant comment.
73-80	Sequence indication	<b>nnnnnnnn</b>	For SYSIN2 statement override.



## 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 DFSDDLTO 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. DFSDDLTO 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 DFSDDLTO stream before the call they are to document. The time and date are printed with each unconditional COMMENT statement. The previous table lists the column number, function, code, and description

## 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---+---<
0 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 DFSDDLTO 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, DFSDDLTO uses the comparison length to perform comparisons if you provide a length. The length comparison option (column 3) is not applicable.

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.

### COMPARE AIB statement

The COMPARE AIB statement is optional. You can use it to compare values returned to the AIB by IMS.

The following table shows the format of the COMPARE AIB statement.

Table 64. COMPARE AIB statement

Column	Function	Code	Description
1	Identifies control statement	<b>E</b>	COMPARE statement.
2	Hold compare option	<b>H</b>	Hold COMPARE statement. See note for COMPARE AIB Statement.
		<b>b</b>	Reset hold condition for a single COMPARE statement.
3	Reserved	<b>b</b>	
4-6	AIB compare	<b>AIB</b>	Identifies an AIB compare.
7	Reserved	<b>b</b>	
8-11	Return code	<b>xxxx</b>	Allow specified return code only.
12	Reserved		
13-16	Reason code	<b>xxxx</b>	Allow specified reason code only.
17-72	Reserved	<b>b</b>	<b>b</b>
73-80	Sequence indication	<b>nnnnnnnn</b>	For SYSIN2 statement override.

**Note for COMPARE AIB Statement:** 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 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.

The following table gives the format of the COMPARE DATA statement.

Table 65. COMPARE DATA statement.

Column	Function	Code	Description
1	Identifies control statement	<b>E</b>	COMPARE statement.
2	Reserved	<b>b</b>	
3	Length comparison option	<b>b</b>	For fixed-length segments or if the LL field of the segment is not included in the comparison; only the data is compared.
		<b>L</b>	The length in columns 5-8 is converted to binary and compared against the LL field of the segment.

Table 65. COMPARE DATA statement (continued).

Column	Function	Code	Description
4	Segment length option	<b>b</b>	
		<b>V</b>	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.
		<b>M</b>	For the second or subsequent variable-length segment of a path call, or for a concatenated logical-child-logical-parent segment.
		<b>P</b>	For fixed-length segments in path calls.
5-8	Comparison length	<b>Z</b>	For message segment.
		<b>nnnn</b>	Length to be used for comparison. (Required for length options V, M, and P if L is coded in column 3.)
9	Reserved	<b>b</b>	
10-13	Identifies type of statement	<b>DATA</b>	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>b</b>	
16-71	String of data		Data against which the segment in the I/O area is to be compared.
72	Continuation column	<b>b</b>	If blank, data is NOT continued.
		<b>x</b>	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	<b>nnnnnnnn</b>	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, DFSDDL0 uses the shorter value for the length of the comparison and display of:
  - The length of data supplied in the I/O area by IMS
  - The number of DATA statements read times 56

**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.

The following table shows the format of the COMPARE PCB statement.

Table 66. COMPARE PCB statement.

Column	Function	Code	Description
1	Identifies control statement	<b>E</b>	COMPARE statement.
2	Hold compare option	<b>H</b>	Hold compare statement.
		<b>b</b>	Reset hold condition for a single COMPARE statement.
3	Snap dump options (if compare was unequal)	<b>b</b>	Use default value. (You can change the default value or turn off the option by coding the value in an OPTION statement.)
		<b>1</b>	The complete I/O buffer pool.
		<b>2</b>	The entire region (batch regions only).
		<b>4</b>	The DL/I blocks.
		<b>8</b>	Terminate the job step on miscompare of DATA or PCB.
4	Extended SNAP <sup>1</sup> options	<b>S</b>	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.
		<b>b</b>	Ignore extended option.
		<b>P</b>	SNAP the complete buffer pool (batch).
		<b>S</b>	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	<b>nn</b>	'nn' is the segment level for COMPARE PCB. A leading zero is required.
7	Reserved	<b>b</b>	
8-9	Status code	<b>b</b>	Allow blank status code only.
		<b>xx</b>	Allow specified status code only.
		<b>XX</b>	Do not check status code.
		<b>OK</b>	blank, GA, GC, or GK allowed.
10	Reserved	<b>b</b>	
11-18	Segment name User Identification	<b>xxxxxxx</b>	Segment name for DB PCB compare.
			Logical terminal for I/O.
			Destination for ALT PCB.
19	Reserved	<b>b</b>	
20-23	Length of key	<b>nxxx</b>	'nxxx' 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>b</b>	If blank, key feedback is not continued.
		<b>x</b>	If not blank, key feedback is continued, starting in columns 16-71 of subsequent statements.
73-80	Sequence indication	<b>nnnnnnnn</b>	For SYSIN2 statement override.

Table 66. COMPARE PCB statement (continued).

Column	Function	Code	Description
<b>Note:</b>			
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.

**Related reference:**

“OPTION statement” on page 265

**Examples of COMPARE DATA and COMPARE PCB statements**

The following examples show how COMPARE DATA and COMPARE PCB statements are used.

**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 A100A100A100A100A100A100A100A100A100A100A100A100A100A100X10102200
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 A100A100A100A100A100A100A100A100A100A100A100A100A100A100X10101700
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
L      DJ      (DJSS     = DJSS01)                               X38006100
L      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
L      DJ    (DJSS     = DJSS01)                                X38006100
L      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
L      DJ    (DJSS     = DJSS01)                                X38006100
L      QAJAXQAJ (QAJASS  = QAJASS97)                            38006200
E LV0036 DATA QSS02QASS02QAJSS01QAJASS97*IQAJA**                *38006300
M0016 DATA QAJSS01*2QAJ**                                38006350

```

## IGNORE statement

DFSDDLTO 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 N or . (period) in a SYSIN2 input stream causes the SYSIN input stream to be ignored as well. The following table gives the format of the IGNORE statement. An example of the statement follows.

Table 67. 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 statement using 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

```

#### Related reference:

“SYSIN2 DD statement” on page 274

## 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. The following table shows the format of the OPTION statement. An example follows.

Table 68. 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=	<ul style="list-style-type: none"> <li>• 0</li> <li>• 1 to 9999</li> </ul>	<ul style="list-style-type: none"> <li>• Turns the ABORT parameter off.</li> <li>• Number of unequal compares before aborting job. Initial default is 5.</li> </ul>
	LINECNT=	10 to 99	Number of lines printed per page. Must be filled with zeros. Initial default 54.
	SNAP <sup>1</sup>	x	SNAP option default, when results of compare are unequal. To turn the SNAP option off, code 'SNAP='. 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.
	LCASE=	<ul style="list-style-type: none"> <li>• H</li> <li>• C</li> </ul>	<ul style="list-style-type: none"> <li>• Hexadecimal representation for lower case characters. This is the initial default.</li> <li>• Character representation for lower case characters.</li> </ul>
	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 DFSDDL0 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---+---<
0 ABORT=5,DUMP,LINECNT=54,SPA=4096,SNAP=5                               67101010
```

**Related reference:**

“COMPARE PCB statement” on page 261



## PUNCH CTL 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, or combinations of these statements.

The following table shows the format and keyword parameters for the PUNCH CTL statement.

Table 69. PUNCH CTL statement

Column	Function	Code	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:		
	OTHER		Reproduces all input control statements except: <ul style="list-style-type: none"> <li>• CTL (PUNCH) statements.</li> <li>• N or . (IGNORE) statements.</li> <li>• COMPARE statements.</li> <li>• CALL statements with functions of SKIP and START. Any control statements that appear between SKIP and START CALLs are not punched.</li> <li>• 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.</li> </ul>
	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.

Table 69. PUNCH CTL statement (continued)

Column	Function	Code	Description
	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.
	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	<b>nnnnnnnn</b>	For SYSIN2 statement override.

To change the punch control options while processing a single DFSDDL0 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 DFSDDL0 merge 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 DFSDDL0 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 the previous table, 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, DFSDDL0 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
```

#### Related reference:

“DFSDDL0 call functions” on page 257

“Control statements” on page 235

## STATUS statement

With the STATUS statement, you establish print options and name the PCB that you want subsequent calls to be issued against.

The following table shows the format of the STATUS statement.

Table 70. STATUS statement

Column	Function	Code	Description
1	Identifies control statement	<b>S</b>	STATUS statement.
2	Output device option	<b>b</b>	Use PRINTDD when in a DL/I region; use I/O PCB in MPP region.
		<b>1</b>	Use PRINTDD in MPP region if DD statement is provided; otherwise, use I/O PCB.
		<b>A</b>	Same as if 1, and disregard all other fields in this STATUS statement.
3	Print comment option	<b>b</b>	Do not print.
		<b>1</b>	Print for each call.
		<b>2</b>	Print only if compare done and unequal.
4	Print AIB option	<b>b</b>	Do not print.
		<b>1</b>	Print for each call.
		<b>2</b>	Print only if compare done and unequal.
5	Print call option	<b>b</b>	Do not print.
		<b>1</b>	Print for each call.
		<b>2</b>	Print only if compare done and unequal.
6	Reserved	<b>b</b>	
7	Print compare option	<b>b</b>	Do not print.
		<b>1</b>	Print for each call.
		<b>2</b>	Print only if compare done and unequal.
8	Reserved	<b>b</b>	

Table 70. STATUS statement (continued)

Column	Function	Code	Description
9	Print PCB option	<b>b</b>	Do not print.
		<b>1</b>	Print for each call.
		<b>2</b>	Print only if compare done and unequal.
10	Reserved	<b>b</b>	
11	Print segment option	<b>b</b>	Do not print.
		<b>1</b>	Print for each call.
		<b>2</b>	Print only if compare done and unequal.
12	Set task and real time	<b>b</b>	Do not time
		<b>1</b>	Time each call.
		<b>2</b>	Time each call if compare done and unequal.
13-14	Reserved	<b>b</b>	
15	PCB selection option	<b>1</b>	PCB name passed in columns 16-23 (use option 1).
		<b>2</b>	DBD name passed in columns 16-23 (use option 2).
		<b>3</b>	Relative DB PCB passed in columns 16-23 (use option 3).
		<b>4</b>	Relative PCB passed in columns 16-23 (use option 4).
		<b>5</b>	\$LISTALL passed in columns 16-23 (use option 5).
		<b>b</b>	If column 15 is blank, DFSDDLTO selects options 2 through 5 based on content of columns 16-23.
Opt. 1 16-23	PCB selection PCB name	<b>alpha</b>	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>b</b> <b>alpha</b>	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>b</b> <b>numeric</b>	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	<b>b</b> <b>numeric</b>	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	<b>\$LISTALL</b>	Prints out all PCBs in the PSB for test script.

Table 70. STATUS statement (continued)

Column	Function	Code	Description
24	Print status option	<b>b</b>	Use print options to print this STATUS statement.
		<b>1</b>	Do not use print options in this statement; print this STATUS statement.
		<b>2</b>	Do not print this STATUS statement but use print options in this statement.
		<b>3</b>	Do not print this STATUS statement and do not use print options in this statement.
25-28	PCB processing option	<b>xxxx</b>	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>b</b>	
30-32	AIB interface	<b>AIB</b>	Indicates that the AIB interface is used and the AIB is passed rather than passing the PCB. (Passing the PCB is the default.) <b>Note:</b> 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. DFSDDLTO 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	<b>nnnnnnnn</b>	For SYSIN2 statement override.

If DFSDDLTO 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 DFSDDLTO 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 DFSDDLTO 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 1 PCBNAME
```

```
|---+---1---+---2---+---3---+---4---+---5---+---6---+---7---+---<
S 1 1 1 1 1 1 PCBNAME 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 2 DBDNAME
```

```
|---+---1---+---2---+---3---+---4---+---5---+---6---+---7---+---<
S 1 1 1 1 1 2 DBDNAME 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 z/OS console without waiting for a reply.

The following table shows the format for the WTO statement.

Table 71. 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 z/OS 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 z/OS system console and waits for a reply.

The following table shows the format of the WTOR statement.

Table 72. 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 hold until DFSDDLTO receives a response from the z/OS 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 for the DL/I test program (DFSDDLTO)

DFSDDLTO uses these DD statements.

Execution JCL depends on the installation data set naming standards as well as the IMS environment (batch or online).

```
//SAMPLE JOB ACCOUNTING,NAME,MSGLEVEL=(1,1),MSGCLASS=3,PRTY=8          33001100
//GET EXEC PGM=DFSRRCO0,PARM='DLI,DFSDDLTO,PSBNAME'                    33001200
//STEPLIB DD DSN=IMS.SDFSRESL,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
/*
```

The following code example shows how to code JCL for DFSDDLTO in a BMP. Use of a procedure is optional and is only shown here as an example.

### Example JCL code for DFSDDLTO in a BMP

```
//SAMPLE JOB ACCOUNTING,NAME,MSGLEVEL=(1,1),MSGCLASS=A                00010047
//*****
/* BATCH DL/I JOB *
//*****
//BMP EXEC IMSBATCH,MBR=DFSDDLTO,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
/*

```

### **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.

#### **Related reference:**

“SYSIN2 DD statement”

“Control statements” on page 235

### **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 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.

#### **Related reference:**

“SYSIN DD statement”

“Control statements” on page 235

“IGNORE statement” on page 265





```

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 2222222222222222222222222222222222222222222222222222222222222222222211 01000360
L      ISRT                              01000370
L Z0080 DATA -SYNC INTERVAL 2 SEG 2 -END EOM 1 X01000380
L P      DATA 222222222222222222222222222222222222222222222222222222222222222222211 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 3333333333333333333333333333333333333333333333333333333333333311 01000510
L      ISRT                              01000520
L Z0080 DATA -SYNC INTERVAL 3 SEG 2 -END EOM 1 X01000530
L P      DATA 3333333333333333333333333333333333333333333333333333333333333311 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:**

1. The PREINIT= parameter coded in the EXEC statement invokes the restart process.
2. When DFSDDLTO 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. DFSDDLTO 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 DFSDDLTO is rescheduled, it examines the checkpoint field and finds 01000240. DFSDDLTO 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, DFSDDLTO would restart from statement 01000000 when it was rescheduled.

**Execution of DFSDDLTO in IMS regions**

DFSDDLTO 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 DFSDDLTO program in an MPP region, you must give DFSDDLTO 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, DFSDDLTO normally gets input by issuing a GU and GNs to the I/O PCB. DFSDDLTO 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, DFSDDLTO 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 DFSDDLTO 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 DFSDDLTO return codes

A non-zero return code from DFSDDLTO indicates the number of unequal comparisons that occurred during that time.

A return code of 0 (zero) from DFSDDLTO does not necessarily mean that DFSDDLTO executed without errors. There are several messages issued by DFSDDLTO 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 DFSDDLTO output. You must examine the output to ensure DFSDDLTO executed as expected.

## DFSDDLTO operations

You can use DFSDDLTO to load a database, print, retrieve, replace, and delete segments; perform regression testing; as a debugging aid; and to verify how a call is executed.

### Load a database

Use DFSDDLTO 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---+---<
0 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

```

## 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 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 DFSDDL0 control statements when a GB status code is returned or the GN has been repeated 9999 times.

## 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

```

## 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

```

## Do regression testing

DFSDDLTO is ideal for doing regression testing. By using a known database, DFSDDLTO 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.

## 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.

## Verify how a call is executed

Because it is very easy to execute a particular call, you can use DFSDDLTO 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.



---

## Chapter 2. DRDA DDM command architecture reference

IMS supports the distributed data management architecture (DDM) of the Distributed Relational Database Architecture™ (DRDA). You can develop your own source DDM server that communicates with the IMS target DDM server to provide access to databases managed by IMS DB in DBCTL and DB/TM IMS systems.

The IMS documentation for the DDM architecture includes only the DDM structures that are required to connect to and communicate with IMS and the DDM structures that have been changed or defined by IMS.

For the complete documentation of the DDM, see *DRDA, Version 4, Volume 3: Distributed Data Management (DDM) Architecture*, which is available from The Open Group at [www.opengroup.org](http://www.opengroup.org).

The DDM architecture includes the following elements or *terms*:


- Commands
- Command objects
- Reply objects
- Reply messages

Each term, whether it is a command, command object, reply object, parameter, or message, is represented by a *codepoint*, a hexadecimal value that represents and identifies the component in communication between a source server and the target server. For example, the EXCSAT command is represented by X'1041', the EXCSATRD reply object is represented by X'1443', the SRVNAM parameter is represented by X'116D', and so on.

As an open standard, the DRDA specification requires that products that use the specification must conform to the conventions, protocols, standards, and so on, of its architecture. However, the DDM architecture that is a part of the DRDA specification allows products to create *product-unique extensions*, in which a product, such as IMS, uses a subset of the existing DDM-defined commands, parameters, and messages, as well as product-unique structures that are defined by the product. When creating a product-unique extension that has product-unique structures, the product must conform to the DDM architecture.

The product-unique extension for IMS conforms to both the DDM architecture and the DRDA specification. IMS uses a subset of the existing DDM-defined commands, parameters, and messages, as well as a variety of IMS-defined structures that conform to the DDM architecture, but are unique to IMS.

### Related concepts:

 [Programming with the IMS support for DRDA \(Application Programming\)](#)

---

## Overview of the syntax for DDM terms supported by IMS

IMS supports the general syntax of terms defined by the distributed data management (DDM) architecture.

All DDM commands, reply messages, and chained objects begin with a 6-byte data stream structure header (DSSHDR), followed in order by a 2-byte binary integer

that defines the length of the term (LL), and a 2-byte hexadecimal codepoint (CP) that uniquely identifies the DDM term, and data, if any.

Parameters of commands, messages, and objects start with LL, followed in order by CP and the data. Parameters, which are also known as instance variables, do not include a DSSHDR.

Some data structures, such as the IMS product-unique data structures aibStream, dbpcbStream, and iopcbStream, do not include DSSHDR, LL, or CP.

**Related reference:**

“DEALLOCDB command (X'C801')” on page 289

## DSSHDR syntax

DSSHDR is the 6-byte header that contains information about the data stream structure (DSS) of terms defined by the distributed data management (DDM) architecture.

DSSHDR has the following format:

**LL** A 2-byte specification of the length of the whole command, reply, or object, including the 6-byte DSS HEADER. The minimum possible value is 6, and the maximum is 32,767.

**DDMID**

A 1-byte Systems Network Architecture (SNA) registered General Data Stream (GDS) identifier. The DDMID field is always D0 for a DDM command.

**FORMAT ID**

A 1-byte indicator of whether the DSS is chained to the next DSS and what to do when errors occur. The byte contains the following bits, from 0 to 7, left to right:

**Bit 0** Unused.

**Bit 1** A flag. 1 indicates that the DSS structure is chained to the next structure. 0 indicates no chaining.

**Bit 2** A flag. 1 indicates to continue when errors occur, and 0, otherwise.

**Bit 3** A flag. 1 indicates that the next DSS has the same request correlator, and 0, otherwise. If bit 1 is 0, bit 3 is also 0.

**Bits 4 through 7**

Indicate the DSS type:

- 1: a Request DSS.
- 2: a Reply DSS.
- 3: an Object DSS.
- 4: an Encrypted Object DSS.

**RQSDRR**

A generated 2-byte field that associates a request with its request data, the replies to the request, and the data that is returned for the request.

---

## DDM commit and rollback processing

The IMS implementation of the distributed data management (DDM) architecture includes support for commit and rollback processing.

XA support and the processing of global transactions is controlled by the DDM commands SYNCCTL and SYNCCRD.



The processing of local transactions is controlled by the DDM commands RDBCMM and RDBRLLBCK.

IMS does not extend these DDM commands beyond their original specification by DRDA.

Documentation for these commands can be found in *DRDA, Version 4, Volume 3: Distributed Data Management (DDM) Architecture*.

---

## DDM commands and command objects

IMS supports a subset of the distributed data management (DDM) architecture commands and command objects and defines other IMS product-unique DDM commands.

### ACCRDB command (X'2001')

The distributed data management (DDM) architecture ACCRDB command allocates a program specification block (PSB) on behalf of the source server. The PSB represents a connection between the DDM source server and an IMS database.

The PSB remains allocated until the database connection is closed and the communications conversation is terminated.

#### Format

►►—DSSHDR—LL—CP—RDBNAM—RDBACCCL—PRDID—PRDDTA—TYPDEFNAM—►►

#### Parameters

##### DSSHDR

The 6-byte header that contains information about the data stream structure (DSS).

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'2001', the 2-byte codepoint of the ACCRDB command.

##### RDBNAM

A required parameter (X'2110') that contains the IMS PSB name that identifies the target database. The PSB name is a character string up to 8 bytes long. RDBNAM can optionally include the alias name of the IMS data store.

##### RDBACCCL

A required parameter that specifies the application manager that accesses the database. The codepoint for RDBACCCL is X'210F'. The value of RDBACCCL is reserved and must be X'2407'.

##### PRDID

A required parameter that specifies the release level of the source DDM server. The codepoint for PRDID is X'112E'.

##### PRDDTA

An optional parameter that specifies product-specific information that is passed to the target if the SRVCLSNM of the target server is not known when the ACCRDB command is issued. The codepoint for PRDDTA is X'2104'. This parameter can be ignored by the target server.

## TYPDEFNAM

A required parameter (X'002F') that specifies the name of the data type definition. TYPDEFNAM consists of a 2-byte specification of length (LL), a 2-byte codepoint (CP), and the VALUE. The VALUE is reserved and must be QTDSQL370, which is the general EBCDIC SQL type definition for machines that use EBCDIC strings, IEEE floating-point numbers, and non-byte-reversed floating-point and integer numbers.

## Usage

If no errors occur during the processing of the ACCRDB command, the IMS target server returns the ACCRDBRM reply message to indicate that the database has been allocated.

## Chained command objects

No command objects can be chained to the ACCRDB command.

## Positive reply messages

In response to the ACCRDB command, the IMS target DDM server returns to the source server the following positive reply messages:

### ACCRDBRM

Access to database completed.

Codepoint: X'2201'

Specifies that the named database in the previous ACCRDB command is now available to the client for processing.

## Error reply messages

In response to the ACCRDB command, the IMS target DDM server can return to the source DDM server the following error reply messages that are unique to the ACCRDB command:

*Table 73. Possible error reply messages unique to the ACCRDB command*

Codepoint of reply message	Name of reply message	Meaning of reply message
X'2203'	RDBATHRM	Not authorized to database.
X'2211'	RDBNFNRM	Database not found.
X'221A'	RDBAFLRM	RDB access failed reply message.  If the RDBNAM parameter was specified on the ACCRDB command, the RDBAFLRM reply message indicates that the database (RDB) failed the attempted connection.

### Related reference:

“ACCRDBRM reply message (X'2201’)” on page 327

“RDBNAM parameter (X'2110’)” on page 357

“RDBAFLRM reply message (X'221A’)” on page 342

“RDBATHRM reply message (X'2203’)” on page 343

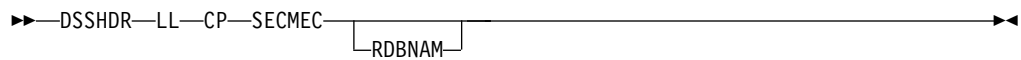
“RDBNACRM reply message (X'2204’)” on page 344

## ACCSEC command (X'106D')

The ACCSEC DDM command is used to determine the type of security checking that is performed when an application program on the source server connects to a database on the IMS target server.

The source server uses the ACCSEC command to negotiate with the IMS target server which type of security mechanism, as defined by the DDM architecture, is used for identification and authentication. IMS supports only the user ID and Password Security Mechanism (USRIDPWD) of the DDM architecture. The ACCSEC command must always precede the SECCHK command when any of the valid security mechanisms are active.

### Format



### Parameters

#### DSSHDR

The 6-byte header that contains information about the data stream structure (DSS).

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'106D', the 2-byte codepoint of the ACCSEC command.

#### SECMEC

A required parameter that specifies the security mechanism that the source server uses when interacting with the IMS target server. IMS supports only the USRIDPWD security mechanism of the DDM architecture. To specify USRIDPWD enter a 2-byte binary number 3 in the SECMEC parameter.

#### RDBNAM

An optional parameter (X'2110') that contains the IMS PSB name that identifies the target database. The PSB name is a character string up to 8 bytes long. RDBNAM can optionally include the alias name of the IMS data store.

### Usage

During the initial handshaking between the source and target DRDA servers, the source server must issue the EXCSAT command chained to the ACCSEC command.

In a successful exchange, the IMS target server returns the ACCSECRD reply data object in response to the ACCSEC command. The ACCSECRD reply object identifies the security mechanism that is used by the IMS target server to the source server. In a successful exchange, the value returned in the ACCSECRD reply object is the same as the value of the SECMEC parameter of the ACCSEC command.

If the IMS target server detects an error while processing the ACCSEC command, the ACCSECRD reply object contains the SECCHKCD parameter. In the

ACCSECRD reply object, the SECCHKCD parameter has an implied severity code of ERROR. After an error, the ACCSEC command must be sent again before a SECCHK command can be sent to authenticate the connection.

### Chained command objects

No command objects can be chained to the ACCSEC command.

### Reply data objects

In response to the ACCSEC command, the IMS target DDM server can return to the source DDM server the following reply data objects:

#### ACCSECRD (X'14AC')

Access security reply data.

### Error reply messages

In response to the ACCSEC command, the IMS target DDM server can return to the source DDM server the following reply messages:

*Table 74. Possible reply messages for the ACCSEC command*

Codepoint of reply message	Name of reply message	Meaning of reply message
X'121C'	CMDATHRM	Not Authorized to Command
X'1232'	AGNPRMRM	Permanent agent error
X'1233'	RSCLMTRM	Resource limits reached
X'123C'	INVRQSRM	Invalid request
X'124C'	SYNTAXRM	Data stream syntax error
X'1250'	CMDNSPRM	Command not supported
X'1251'	PRMNSPRM	Parameter not supported
X'1252'	VALNSPRM	Parameter value not supported
X'1254'	CMDCHKRM	Command check reply message
X'125F'	TRGNSPRM	Target not supported

#### Related reference:

“ACCSECRD reply object (X'14AC')” on page 329

“RDBNAM parameter (X'2110')” on page 357

## CLSQR command (X'2005')

The distributed data management (DDM) Architecture CLSQR command closes a query that was opened previously by an OPNQRY call.

### Format

▶▶—DSSHDR—LL—CP—PCBNAME—◀◀

## Parameters

### DSSHDR

The 6-byte header that contains information about the data stream structure (DSS).

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'2005', the 2-byte codepoint of the CLSQRY command.

### PCBNAME

A required parameter that specifies the PCB name that uniquely identifies the query made by a DL/I call. The PCB name is specified as a character string. The value is initially sent with the original OPNQRY command. The same value must subsequently be sent in commands such as CNTQRY, CLSQRY, and RLSE for proper correlation with the original OPNQRY call. The codepoint for the PCBNAME parameter is X'C907'.

## Usage

Use the DDM command CLSQRY (close a query) to close a query that was opened previously by an OPNQRY call.

## Chained command objects

No command objects can be chained to the CLSQRY command.

## Error reply messages

If errors occur during the processing of the CLSQRY command, the IMS target DDM server can return to the source DDM server the following error reply messages:

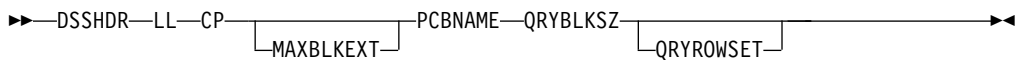
*Table 75. Possible error reply messages for the CLSQRY command*

Codepoint of reply message	Name of reply message	Meaning of reply message
X'121C'	CMDATHRM	Not Authorized to Command
X'1232'	AGNPRMRM	Permanent agent error
X'1233'	RSCLMTRM	Resource limits reached
X'1245'	PRCCNVRM	Conversational protocol error
X'124C'	SYNTAXRM	Data stream syntax error
X'1250'	CMDNSPRM	Command not supported
X'1251'	PRMNSPRM	Parameter not supported
X'1252'	VALNSPRM	Parameter value not supported
X'1254'	CMDCHKRM	Command check reply message
X'125F'	TRGNSPRM	Target not supported

## CNTQRY command (X'2006')

The distributed data management (DDM) architecture CNTQRY command continues a query by resuming the return of the result set data that was generated by a previous OPNQRY call.

## Format



## Parameters

### DSSHDR

The 6-byte header that contains information about the data stream structure (DSS).

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'2006', the 2-byte codepoint of the CNTQRY command.

### MAXBLKEXT

An optional parameter that specifies the maximum number of extra blocks per result set that the requester is capable of receiving as reply data in the response to the CNTQRY command. The number is specified as a 2-byte binary number. A value of 0 indicates that the requester is not capable of receiving extra query blocks of answer set data. A value of -1 indicates that the requester is capable of receiving the entire result set. The codepoint for MAXBLKEXT is X'2141'.

### PCBNAME

A required parameter that specifies the PCB name that uniquely identifies the query made by a DL/I call. The PCB name is specified as a character string. The value is initially sent with the original OPNQRY command. The same value must subsequently be sent in commands such as CNTQRY, CLSQRY, and RLSE for proper correlation with the original OPNQRY call. The codepoint for the PCBNAME parameter is X'C907'.

### QRYBLKSZ

A required parameter that specifies the size of query blocks that is ideal for the source application program. Query blocks are used by the target server to return answer set data. The target server can override this parameter as needed. The query block size is specified as a 4-byte unsigned binary number. The minimum size for a query block is 0.5 KB. The maximum size is 10 MB. The codepoint for QRYBLKSIZ is X'2114'.

### QRYROWSET

An optional parameter that specifies the number of rows of data to return in one network reply. The number of rows is specified as a 4-byte binary number. The minimum value for QRYROWSET is 0. The maximum possible value is 32 767. The codepoint for QRYROWSET is X'2156'.

## Usage

The DDM command CNTQRY (continue a query) to resume the return of result set data generated by a previous OPNQRY call.

## Chained command objects

No command objects are chained to the CNTQRY command.

## Reply data objects

The following reply data objects can be returned in response to the CNTQRY command:

## QRYDTA (X'241B')

Query answer set data.

### Error reply messages

If errors occur during the processing of the CNTQRY command, the IMS target DDM server can return to the source DDM server the following error reply messages:

Table 76. Possible error reply messages for the CNTQRY command

Codepoint of reply message	Name of reply message	Meaning of reply message
X'121C'	CMDATHRM	Not authorized to command
X'1232'	AGNPRMRM	Permanent agent error
X'1233'	RSCLMTRM	Resource limits reached
X'1245'	PRCCNVRM	Conversational protocol error
X'124C'	SYNTAXRM	Data stream syntax error
X'1250'	CMDNSPRM	Command not supported
X'1251'	PRMNSPRM	Parameter not supported
X'1252'	VALNSPRM	Parameter value not supported
X'1254'	CMDCHKRM	Command check reply message
X'125F'	TRGNSPRM	Target not supported
X'2204'	RDBNACRM	Database not accessed
X'220B'	ENDQRYRM	End of query
X'220D'	ABNUOWRM	Abnormal end of unit of work condition
X'2213'	SQLERRRM	SQL error condition
X'2218'	RDBUPDRM	Database update reply message.

## DEALLOCDB command (X'C801')

The distributed data management (DDM) DEALLOCDB command terminates all resources that are associated with a PSB by deallocating the PSB named in the RDBNAM parameter.

### Format

►►—DSSHDR—LL—CP—RDBNAM—►►

### Parameters

#### DSSHDR

The 6-byte header that contains information about the data stream structure (DSS).

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'C801', the 2-byte codepoint of the DEALLOCDB command.

#### RDBNAM

A required parameter (X'2110') that contains the IMS PSB name that identifies the target database. The PSB name is a character string up to 8 bytes long. RDBNAM can optionally include the alias name of the IMS data store.

## Usage

If no errors occur during the processing of the DEALLOCDB command, the IMS target server returns the DEALLOCDBRM reply message to indicate that the database has been successfully deallocated.

## Chained command objects

No command objects can be chained to the DEALLOCDB command.

## Positive reply messages

In response to the DEALLOCDB command, the IMS target DDM server returns to the source server the following positive reply messages:

### DEALLOCDBRM (X'CA01')

Deallocation of database complete.

Specifies that the named PSB is now deallocated.

## Error reply messages

In response to the DEALLOCDB command, the IMS target DDM server can return to the source DDM server the following error reply messages:

*Table 77. Possible error reply messages for the DEALLOCDB command*

Codepoint of reply message	Name of reply message	Meaning of reply message
X'1232'	AGNPRMRM	Permanent agent error
X'124C'	SYNTAXRM	Data stream syntax error

### Related reference:

“RDBNAM parameter (X'2110’)” on page 357

“DEALLOCDBRM reply message (X'CA01’)” on page 331

“Overview of the syntax for DDM terms supported by IMS” on page 281

## DLIFUNC command object (X'CC05')

Use the distributed data management (DDM) architecture DLIFUNC (DL/I function) command object to specify the DL/I function that is being called.

## Format

►►—DSSHDR—LL—CP—BYTSTRDR—————►►

## Parameters

### DSSHDR

The 6-byte header that contains information about the data stream structure (DSS).

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.



**CP** X'CC05', the 2-byte codepoint of the DLIFUNC command object.

**BYTSTRDR**

Byte String Data Representation, a required character string that contains the DL/I call to run on the database. The following character string values can be specified in the DLIFUNC command object:

**ISRT**

Insert call

**DLET**

Delete call

**REPL**

Replace call

**GHU**

Get Hold Unique call

**GU** Get Unique call

**GHN**

Get Hold Next call

**GN** Get Next call

**GHNP**

Get Hold Next Within Parent call

**GNP**

Get Next Within Parent call

**DELETE**

Batch Delete call

**UPDATE**

Batch Replace call

**RETRIEVE**

Batch Retrieve call

**Related reference:**

“EXCSQLIMM command (X'200A’)” on page 294

“OPNQRY command (X'200C’)” on page 304

## **DLIFUNCFLG command object (X'CC09')**

Use the distributed data management (DDM) architecture DLIFUNCFLG (DL/I function flag) command object to specify whether a DL/I batch processing operation starts with a GU or a GN call and which SSA list is associated with each call.

### **Format**

►►—DSSHDR—LL—CP—FFFF—◄◄

### **Parameters**

**DSSHDR**

The 6-byte header that contains information about the data stream structure (DSS).

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'CC09', the 2-byte codepoint of the DLIFUNCFLG command object.

**FFFF**

A required 4-byte flag value. Each byte in the flag specifies a different DL/I batch processing option:

**First byte**

X'00' Begin batch processing with a GHN call.

X'80' Begin batch processing with a GHU call.

**Second byte**

The first four bits of the second byte indicate which SSAList is associated with the get position call. The second four bits indicate which SSAList is associated with an optional REPL call that follows the get position call:

B'0000'

No SSA

B'1000'

First SSA in list

B'0100'

Second SSA in list

B'0010'

Third SSA in list

B'0001'

Fourth SSA in list

**Third byte**

The third byte is specified in the same format as the second byte, but is used for subsequent GHN and optional REPL calls that follow the initial get position call.

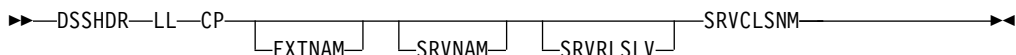
**Fourth byte**

Reserved.

## EXCSAT command (X'1041')

The distributed data management (DDM) architecture EXCSAT command initiates the exchange of attributes between a source application server and an IMS target server to identify the server class names and levels of DDM support of each server. The EXCSAT command must always be the first command sent from a source server to the IMS target server.

### Format



### Parameters

**DSSHDR**

The 6-byte header that contains information about the data stream structure (DSS).

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'1041', the 2-byte codepoint of the EXCSAT command.

### **EXTNAM**

Optional. The variable-length name of the process or thread that is requesting access to an IMS database. The specified name identifies the application thread for tracing and problem determination. If the job name includes embedded blanks, the name must be enclosed in quotation marks. The maximum length of EXTNAM is 255 bytes. The codepoint is X'115E'.

### **SRVNAM**

Optional. The variable-length name of the source DDM server. The specified name identifies for tracing and problem determination purposes the hostname of the computer that the source application program is running on. If the server name includes embedded blanks, the name must be enclosed in quotation marks. The maximum length is 255 bytes. The codepoint is X'116D'.

### **SRVCLSNM**

Specifies the DDM server class name used by IMS: DFS. DFS is currently the only class name supported by IMS. The SRVCLSNM enables the DRDA product-unique extension used by IMS.

The codepoint of SRVCLSNM is X'1147'. The variable-length DDM server class name is specified as a character string.

## **Usage**

The EXCSAT DDM command is used to initiate a request to access an IMS database and identify the requestor, a DDM source server to the DDM target server of IMS.

During the initial handshaking between the source and target DRDA servers, the source server must issue the EXCSAT command chained to the ACCSEC command.

In a successful exchange, the IMS target server returns the EXCSATRD reply data object in response to the EXCSAT command. The EXCSATRD reply object identifies the IMS target server to the source server.

## **Chained command objects**

No command objects are chained to the EXCSAT command.

## **Reply data objects**

In response to the EXCSAT command, the IMS target DDM server can return to the source DDM server the following reply data objects:

### **EXCSATRD (X'1443')**

Exchange server attributes.

## **Error reply messages**

In response to the EXCSAT command, the IMS target DDM server can return to the source DDM server the following error reply messages:

*Table 78. Possible error reply messages for the EXCSAT command*

<b>Codepoint of reply message</b>	<b>Name of reply message</b>	<b>Meaning of reply message</b>
X'1210'	MGRLVLRM	Manager-level conflict
X'124C'	SYNTAXRM	Data stream syntax error

Table 78. Possible error reply messages for the EXCSAT command (continued)

Codepoint of reply message	Name of reply message	Meaning of reply message
X'1250'	CMDNSPRM	Command not supported
X'1251'	PRMNSPRM	Parameter not supported
X'1252'	VALNSPRM	Parameter value not supported
X'1254'	CMDCHKRM	Command check reply message
X'125F'	TRGNSPRM	Target not supported

**Related reference:**

“EXCSATRD reply object (X'1443’)” on page 334

## EXCSQLIMM command (X'200A')

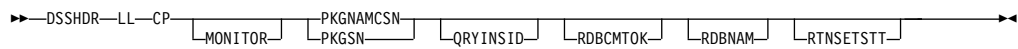
The distributed data management (DDM) architecture EXCSQLIMM command executes an insert, update, or delete operation on an IMS database.

### Format

DLI Flow:



SQL Flow:



### Parameters

#### DSSHDR

The 6-byte header that contains information about the data stream structure (DSS).

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'200A', the 2-byte code point of the EXCSQLIMM command.

#### QRYINSID

An 8-byte query instance identifier.

**Restriction:** This parameter is required if the EXCSQLIMM command is operating on a positioned delete/update SQL statement and more than one query instance exists for the section associated with the query.

#### PCBNAME

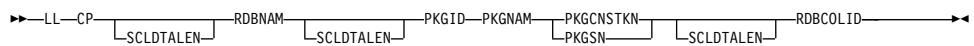
A required parameter that specifies the PCB name that uniquely identifies the query made by a DL/I call. The PCB name is specified as a character string. The value is initially sent with the original OPNQRY command. The same value must subsequently be sent in commands such as CNTQRY, CLSQRY, and RLSE for proper correlation with the original OPNQRY call. The codepoint for the PCBNAME parameter is X'C907'.

## PKGNAMCSN(X'2113')

Specifies the fully qualified package name, consistency token, and section number within the package that is used to execute the SQL. The PKGNAMCSN can have one of the following formats, depending on the length of the RDBNAM, RDBCOLID, and PKGID contained therein:

- RDBNAM, RDBCOLID, and PKGID each have a length of 18. This format of the PKGNAMCSN is identical to the sole format used before DDM Level 7, where the length is fixed at 68. The use of the SCLDTALEN is disallowed with this format.
- At least one of RDBNAM, RDBCOLID, or PKGID has a length > 18. This format of the PKGNAMCSN requires the SCLDTALEN precedes each of the RDBNAM, RDBCOLID, and PKGID. With this format, the PKGNAMCSN has a minimum length of 75 and a maximum length of 785.

### Format:



### Parameters:

#### RDBNAM

An 18- to 255-byte character field that represents the relational database name.

#### PKGID

An 18- to 255-byte character field that represents the relational database package identifier.

#### PKGNAM

An 18- to 255-byte character field that specifies the fully qualified name of a relational database package.

#### PKGCNSTKN

An 8-byte character field that verifies that the requester's application and the relational database package are synchronized. Mutually exclusive with PKGSN.

#### PKGSN

A 2-byte short field that represents the Section Number. Mutually exclusive with PKGCNSTKN.

#### SCLDTALEN

Specifies the length of the instance variable that immediately follows:

- RDB collection identifier (RDBCOLID)
- Relational database name (RDBNAM)
- RDB package identifier (PKGID)

This token is allowed only when the length of one or more of the parameters listed is greater than 18 bytes.

#### RDBCOLID

An 18- to 255-byte character field that identifies a unique collection of objects that are contained in a relational database. It is used for user-defined grouping.

**Note:** If the length of any one of RDBNAM, RDBCOLID, or PKGID exceeds 18 bytes, the SCLDTALEN is mandatory and must precede the RDBCOLID. Otherwise, the SCLDTALEN is disallowed.

### **RDBCMTOK**

An optional parameter (X'2105') that specifies whether the database allows the processing of commit and rollback operations. Set the value to X'F1' (TRUE), which indicates that the database does allow commit and rollback processing.

**Note:** IMS Universal drivers always send a value of TRUE.

### **RTNSETSTT(X'210E')**

If any special register setting was modified during command execution, the return SET statement controls whether the target server must return one or more SQLSTT reply data objects upon successful command processing. Each SQLSTT reply data object contains an SQL SET statement for a special register whose setting was modified on the current connection.

If no special register setting was modified, no SQLSTT reply data object is returned, regardless of the RTNSETSTT setting.

#### **Format:**

▶▶—LL—CP—VALUE—————▶▶

#### **Parameters:**

##### **VALUE**

X'00' Target server must not return any SQL SET statements.

X'01' Target server must return one or more SQL SET statements for special registers whose settings were modified.

—

**Note:** IMS always sends X'01' from the IMS Universal drivers.

### **MONITOR(X'1900')**

▶▶—LL—CP—FLAGS—————▶▶

##### **FLAGS**

A 4-byte flag value.

## **Usage**

The DDM command EXCSQLIMM (execute immediate SQL) executes a replace, insert, or delete operation on an IMS database.

If no errors occur during the processing of the EXCSQLIMM command, the IMS target server returns the database update reply message RDBUPDRM (X'2218').

## **Chained command objects**

The following command objects can be chained to the EXCSQLIMM command:

### **INAIB (X'CC01')**

Contains AIB data. If the DLIFUNC value is either DELETE or UPDATE, the AIB parameter is required.

### **DLIFUNC (X'CC05')**

The DL/I call to execute on the database. The DL/I call is specified as a

character string and defines the action to perform on the database. For a description of the possible values for DLIFUNC, see the description of DLIFUNC.

#### **FLDENTRY (X'CC03')**

If DLIFUNC is set to ISRT, REPL, or UPDATE, the FLDENTRY parameter is required.

#### **SSALIST (X'CC06')**

Lists the segment search arguments. If DLIFUNC is set to UPDATE or DELETE, the SSALIST parameter is required. If DLIFUNC is set to DLET, ISRT, or REPL, the SSALIST parameter is optional.

### **Positive reply messages**

In response to the EXCSQLIMM command, the IMS target DDM server returns to the source server the following positive reply message:

#### **RDBUPDRM (X'2218')**

Database update reply message.

### **Reply data objects**

No reply data objects are returned in response to the EXCSQLIMM command.

### **Error reply messages**

In response to the EXCSQLIMM command, the IMS target DDM server can return to the source DDM server the following error reply messages:

*Table 79. Possible error reply messages for the EXCSQLIMM command*

<b>Code point of reply message</b>	<b>Name of reply message</b>	<b>Meaning of reply message</b>
X'121C'	CMDATHRM	Not Authorized to Command
X'1232'	AGNPRMRM	Permanent agent error
X'1233'	RSCLMTRM	Resource limits reached
X'1245'	PRCCNVRM	Conversational protocol error
X'124C'	SYNTAXRM	Data stream syntax error
X'1250'	CMDNSPRM	Command not supported
X'1251'	PRMNSPRM	Parameter not supported
X'1252'	VALNSPRM	Parameter value not supported
X'1253'	OBJNSPRM	Object not supported
X'1254'	CMDCHKRM	Command check reply message
X'125F'	TRGNSPRM	Target not supported
X'2204'	RDBNACRM	Database not accessed
X'220D'	ABNUOWRM	Abnormal end of unit of work condition
X'220E'	DTAMCHRM	Data Descriptor Mismatch
X'2213'	SQLERRRM	SQL error condition
X'2225'	CMMRQSRM	Commitment request

## EXCSQLIMM examples

The following example shows EXCSQLIMM that is part of the request to an OPNQRY call.

	SEND BUFFER: EXCSQLIMM	(ASCII)	(EBCDIC)
[ibm][ims][drda][t4] 0000	0060D0510002005A 200A00442113E2C1	.^.Q...Z ..D!...	.-}.....!.....SA
[ibm][ims][drda][t4] 0010	D4D7D3C540404040 4040404040404040	...@@@@@@@@@@@@	MPLE
[ibm][ims][drda][t4] 0020	D5E4D3D3C9C44040 4040404040404040	.....@@@@@@@@	NULLID
[ibm][ims][drda][t4] 0030	4040E2E8E2E2C8F2 F0F0404040404040	@@.....@@@@@@	SYSSH200
[ibm][ims][drda][t4] 0040	404040405359534C 564C303100410005	@@@SYSLVL01.A..	...<.<.....
[ibm][ims][drda][t4] 0050	2105F10005210E01 0008190080000000	!.....!.....	..1.....

### Related reference:

“DLIFUNC command object (X'CC05’)” on page 290

“FLDENTRY command object (X'CC03’)” on page 300

“SSALIST command object (X'CC06’)” on page 325

“INAIB command object (X'CC01’)” on page 302

“RDBUPDRM reply message (X'2218’)” on page 346

## EXCSQLSET command (X'2014')

The distributed data management (DDM) architecture Execute SQL SET command (EXCSQLSET) executes one or more SET statements to establish the application environment.

### Format

►►—DSSHDR—LL—CP—PKGNAMCSN—RTNSETSTT—MONITOR—◄◄

### Parameters

#### DSSHDR

The 6-byte header field containing information about the DSS.

**LL** A 2-byte field that has the length of the EXCSQLSET command.

#### CP(X'2014')

The 2-byte codepoint of the EXCSQLSET command.

#### PKGNAMCSN(X'2113')

Specifies the fully qualified package name, consistency token, and section number within the package that is used to execute the SQL. The PKGNAMCSN can have one of the following formats, depending on the length of the RDBNAM, RDBCOLID, and PKGID contained therein:

- RDBNAM, RDBCOLID, and PKGID each have a length of 18. This format of the PKGNAMCSN is identical to the sole format used before DDM Level 7, where the length is fixed at 68. The use of the SCLDTALEN is disallowed with this format.
- At least one of RDBNAM, RDBCOLID, or PKGID has a length > 18. This format of the PKGNAMCSN requires the SCLDTALEN precedes each of the RDBNAM, RDBCOLID, and PKGID. With this format, the PKGNAMCSN has a minimum length of 75 and a maximum length of 785.

#### Format:

►►—LL—CP—      —RDBNAM—      —PKGID—PKGNAME—      —      —RDBCOLID—◄◄

SCLDTALEN
SCLDTALEN
PKGCNSTKN
PKGSN
SCLDTALEN



**Parameters:**

**RDBNAM**

An 18- to 255-byte character field that represents the relational database name.

**PKGID**

An 18- to 255-byte character field that represents the relational database package identifier.

**PKGNAME**

An 18- to 255-byte character field that specifies the fully qualified name of a relational database package.

**PKGCNSTKN**

An 8-byte character field that verifies that the requester's application and the relational database package are synchronized. Mutually exclusive with PKGSN.

**PKGSN**

A 2-byte short field that represents the Section Number. Mutually exclusive with PKGCNSTKN.

**SCLDTALEN**

Specifies the length of the instance variable that immediately follows:

- RDB collection identifier (RDBCOLID)
- Relational database name (RDBNAM)
- RDB package identifier (PKGID)

This token is allowed only when the length of one or more of the parameters listed is greater than 18 bytes.

**RDBCOLID**

An 18- to 255-byte character field that identifies a unique collection of objects that are contained in a relational database. It is used for user-defined grouping.

**Note:** If the length of any one of RDBNAM, RDBCOLID, or PKGID exceeds 18 bytes, the SCLDTALEN is mandatory and must precede the RDBCOLID. Otherwise, the SCLDTALEN is disallowed.

**RTNSETSTT(X'210E')**

Return SET statement controls whether the target server must return one or more SQLSTT reply data objects, each containing an SQL SET statement for a special register whose setting has been modified on the current connection, upon successful processing of the command, if any special register had its setting modified during execution of the command. NO SQLSTT reply data object is returned if no special register has had its setting modified, regardless of RTNSETSTT setting.

**Format:**

▶▶—LL—CP—VALUE—▶▶

**Parameters:**

**VALUE**

X'00' – Target server must not return any SQL SET statements.

X'01' – Target server must return one or more SQL SET statements for special registers whose settings have been modified.

Note: IMS will always send a 0x'01' from the Universal Driver.

### MONITOR(X'1900')



#### FLAGS

A 4-byte flag value.

### EXCSQLSET examples

The following example shows EXCSQLSET that is part of the request to an OPNQRY call.

[ibm][ims][drda][t4]	SEND BUFFER: EXCSQLSET	(ASCII)	(EBCDIC)
[ibm][ims][drda][t4]	0 1 2 3 4 5 6 7 8 9 A B C D E F	0123456789ABCDEF	0123456789ABCDEF
[ibm][ims][drda][t4] 0000	004ED05100010048 2014004421134BC9	.N.Q...H ..D!.K.	.+}.....I
[ibm][ims][drda][t4] 0010	D4E2F14040404040 4040404040404040	...@@@@@@@@@@@@	MS1
[ibm][ims][drda][t4] 0020	D5E4D3D3C9C44040 4040404040404040	.....@@@@@@@@	NULLID
[ibm][ims][drda][t4] 0030	4040E2E8E2E2D5F2 F0F0404040404040	@@.....@@@@@@	SYSSN200
[ibm][ims][drda][t4] 0040	404040405359534C 564C30310041	@@@SYSLVL01.A	...<.<.....

Note: RTNSETSTT & MONITOR are not in the example.

### FLDENTRY command object (X'CC03')

Use the distributed data management (DDM) architecture FLDENTRY (field entry) command object to specify the field to insert or update.

#### Format



#### Parameters

##### DSSHDR

The 6-byte header that contains information about the data stream structure (DSS).

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'CC03', the 2-byte codepoint of the FLDENTRY command object.

##### RECOFF

A required, 4-byte signed integer that contains the offset of the field within the hierarchic path I/O area.

##### FLDVAL

A required string that contains the byte array to place into the I/O area for the ISRT or REPL DL/I call starting at position RECOFF.

#### Usage

Multiple FLDENTRY command objects might be chained to the EXCSQLIMM command.

#### Related reference:

“EXCSQLIMM command (X'200A)” on page 294

## FLDENTRYREL command object (X'CC0C')

Use the distributed data management (DDM) architecture FLDENTRYREL (relative field entry) command object to specify which field to insert or update.

**Restriction:** The FLDENTRYREL command object is supported only with an ODBM DDM level of 1, 2, 3 or 1, 3.

### Format

►►—DSSHDR—LL—CP—SEGMOFF—SEGMID—FLDVAL—————►►

### Parameters

#### DSSHDR

The 6-byte header that contains information about the data stream structure (DSS).

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'CC0C', the 2-byte codepoint of the FLDENTRYREL command object.

#### SEGMOFF

A required, 4-byte, signed integer that specifies the relative offset of the target field from the start of the parent segment.

#### SEGMID

A required, 1-byte, signed integer that specifies which segment in the SEGMLIST the field is referenced from. This value is relative to 1 rather than 0.

#### FLDVAL

The value for the field that is being updated or inserted.

## IMSCALL command (X'C803')

Use the distributed data management (DDM) architecture IMSCALL command to issue DL/I calls for IMS DB system services.

### Format

►►—DSSHDR—LL—CP—CALLNAME—IOAREA—————►►

### Parameters

#### DSSHDR

The 6-byte header that contains information about the data stream structure (DSS).

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'C803', the 2-byte codepoint of the IMSCALL command.

#### CALLNAME

A required character string (codepoint is X'C90C') that represents the type of the DL/I call that is made.

#### IOAREA

An optional parameter in byte array (codepoint is X'C90B') that specifies the input and output area.

## Usage

The IMSCALL command issues DL/I calls for IMS DB system services in the following format:

►► *call\_name*—INAIB—IOAREA—◄◄

## Chained command objects

### INAIB (X'CC01')

The AIB data to send from the source to the target server.

## Positive reply messages

In response to the IMSCALL command, the IMS target DDM server returns to the source server the following reply message:

### IMSCALLRM (X'CA04')

Contains the results of the IMSCALL command. The results can indicate the success or failure of the DL/I call for IMS DB system services.

## Error reply messages

In response to the OPNQRY command, the IMS target DDM server can return to the source DDM server the following error reply messages:

*Table 80. Possible reply messages for the OPNQRY command*

Codepoint of reply message	Name of reply message	Meaning of reply message
X'1232'	AGNPRMRM	Permanent agent error
X'124C'	SYNTAXRM	Data stream syntax error
X'1251'	PRMNSPRM	Parameter not supported
X'1252'	VALNSPRM	Parameter value not supported

### Related reference:

“IMSCALLRM reply message (X'CA04)’” on page 336

“INAIB command object (X'CC01)’”

“DL/I calls for IMS DB system services” on page 35

## INAIB command object (X'CC01')

Use the distributed data management (DDM) architecture INAIB (input AIB) command object to contain the AIB data to send from the source to the target server.

### Format

►► DSSHDR—LL—CP—AIBRSNM1—AIBRSNM2—AIBSFUNC—AIBOALEN—◄◄

## Parameters

### DSSHDR

The 6-byte header that contains information about the data stream structure (DSS).

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'CC01', the 2-byte codepoint of the INAIB command object.

### AIBRSNM1

A required String that contains the resource (PCB) name. The string must be left-aligned and padded with blanks, to a total of 8 bytes. The codepoint is X'C901'.

### AIBRSNM2

An optional String that contains a 4-character ID of ODBA startup table DFSxxx0, where xxx is the 4-character ID. The codepoint is X'C902'.

### AIBSFUNC

An optional String that contains the sub-function code. The String must be left justified and padded with blanks to a total of 8 bytes. The codepoint is X'C903'.

### AIBOALEN

An optional, 4-byte integer that specifies the maximum output length. This field is used for all calls that return data. The codepoint is X'C904'.

## Usage

This AIB command object contains only the AIB data to send from the source to the target server. The AIB and DBPCB data to send from the target to the source server is contained in the aibStream and dbpcbStream data structures inside the OUTAIBDBPCB objects.

### Related reference:

“AIBOALEN parameter (X'C904’)” on page 351

“AIBRSNM1 parameter (X'C901’)” on page 351

“AIBRSNM2 parameter (X'C902’)” on page 351

“AIBSFUNC parameter (X'C903’)” on page 352

“EXCSQLIMM command (X'200A’)” on page 294

“IMSCALL command (X'C803’)” on page 301

“OPNQRY command (X'200C’)” on page 304

## MONITORRD command (X'1C00')

The distributed data management (DDM) architecture MONITORRD allows the target agent to return monitoring data to the source agent. The value returned is used to determine the elapsed CPU time for a database call.

### Format

►►—DSSHDR—LL—CP—ETIME—►►

## Parameters

### DSSHDR

The 6-byte header field containing information about the DSS.

**LL** A 2-byte field that has the length of the MONITORRD command.

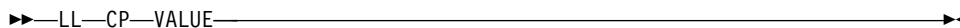
| **CP(X'1C00')**

| The 2-byte codepoint of the MONITORRD command.

| **ETIME(X'1901')**

| The elapsed time is a 64-bit binary number that measures time in  
| microseconds. Consists of two bytes of length field (LL), and two bytes of the  
| code point, followed by the data. The length is 12 bytes.

| **Format:**



| **Parameters:**

| **VALUE**

| An 8-byte field representing the elapsed time.

| **MONITORRD example**

| In the following example, the server time is calculated in the trace by aggregated  
| all of the MONITORRD ETIME values for a communication exchange.

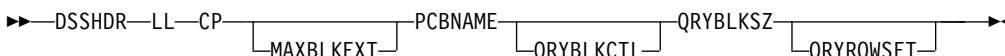
```
| [ibm][ims][drda][t4] RECEIVE BUFFER: MONITORRD (ASCII) (EBCDIC)
| [ibm][ims][drda][t4] 0000 0016D04300020010 1C00000C19010000 ...C..... ..}.....
| [ibm][ims][drda][t4] 0010 000000036B39 ....k9 .....
| [ibm][ims][drda][SystemMonitor:stop] core: 283.09152ms | network: 256.137805ms | server: 254.816ms
```

| **OPNQRY command (X'200C')**

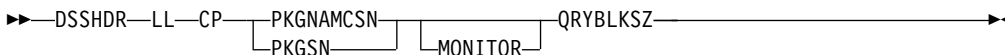
| The distributed data management (DDM) architecture OPNQRY command opens a  
| query to a database for a read request.

| **Format**

| DLI Flow:



| SQL Flow:



| **Parameters**

| **DSSHDR**

| The 6-byte header that contains information about the data stream structure (DSS).

| **LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

| **CP** X'200C', the 2-byte code point of the OPNQRY command.

| **MAXBLKEXT**

| An optional parameter that specifies the maximum number of extra blocks per  
| result set that the requester is capable of receiving as reply data in the response  
| to an OPNQRY or CNTQRY command. The number is specified as a 2-byte  
| binary number. A value of 0 indicates that the requester is not capable of

receiving extra query blocks of answer set data. A value of -1 indicates that the requester is capable of receiving the entire result set. The code point for MAXBLKEXT is X'2141'.

#### **MONITOR(X'1900')**

▶—LL—CP—FLAGS—▶

##### **FLAGS**

A 4-byte flag value.

##### **PCBNAME**

A required parameter that specifies the PCB name that uniquely identifies the query made by a DL/I call. The PCB name is specified as a character string. The value is initially sent with the original OPNQRY command. The same value must subsequently be sent in commands such as CNTQRY, CLSQRY, and RLSE for proper correlation with the original OPNQRY call. The codepoint for the PCBNAME parameter is X'C907'.

##### **PKGNAMCSN(X'2113')**

Specifies the fully qualified package name, consistency token, and section number within the package that is used to execute the SQL. Mutually exclusive with PKGSN.

##### **PKGSN**

A 2-byte short field that represents the Section Number. Mutually exclusive with PKGCNSTKN.

##### **QRYBLKCTL**

An optional parameter that specifies the type of query block protocol that is used when a query is opened. IMS supports only the limited block query protocol of the DDM architecture. If the QRYBLKCTL parameter is specified on the OPNQRY command, the 2-byte data portion of the QRYBLKCTL parameter must specify the hexadecimal value of X'2417', the code point for the limited block query protocol (LMTBLKPRC). If the QRYBLKCTL parameter is omitted from the OPNQRY command, the IMS target server still uses the limited block query protocol. The code point for the QRYBLKCTL parameter is X'2132'.

##### **QRYBLKSZ**

A required parameter that specifies the size of query blocks that is ideal for the source application program. Query blocks are used by the target server to return answer set data. The target server can override this parameter as needed. The query block size is specified as a 4-byte unsigned binary number. The minimum size for a query block is 0.5 KB. The maximum size is 10 MB. The code point for the QRYBLKSIZ parameter is X'2114'.

##### **QRYROWSET**

An optional parameter that specifies the number of rows of data to return in one network reply. The number of rows is specified as a 4-byte binary number. The minimum value for QRYROWSET is 0. The maximum value is 32 767. The code point for the QRYROWSET parameter is X'2156'.

## **Usage**

If no errors occur during processing of the OPNQRY, the IMS target server returns the OPNQRYRM reply message to indicate that the query was successfully opened.

## Command objects

The following command objects can be chained to the OPNQRY command:

### INAIB (X'CC01')

A required command object that contains AIB data.

**Note:** The INAIB object in OPNQRY is not used in DRDA DDM command support for native SQL implementations.

### DLIFUNC (X'CC05')

A required command object that specifies the action to take on the database. The data field of DLIFUNC is a database function that is specified as a character string. The valid values for DLIFUNC, when it is chained to the OPNQRY command, are: RETRIEVE, GHU, GU, GHN, GN, GNP, or GHNP.

**Note:** The DLIFUNC object in OPNQRY is not used in DRDA DDM command support for native SQL implementations.

### RTRVFLD (X'CC04')

An optional scalar data object representing a field that the client wants to retrieve. Multiple RTRVFLD objects can be chained to the OPNQRY command. If an RTRVFLD object is not included on the OPNQRY command, all fields in the retrieved segment are returned.

**Note:** The RTRVFLD object in OPNQRY is not used in DRDA DDM command support for native SQL implementations.

### SSALIST (X'CC06')

An optional chained object that lists the segment search arguments. If the SSALIST is not included on the OPNQRY command, the IMS target server ignores any RTRVFLD chained objects and the query results in an unqualified step through the IMS database.

**Note:** The SSALIST object in OPNQRY is not used in DRDA DDM command support for native SQL implementations.

## Positive reply messages

In response to the OPNQRY command, the IMS target DDM server returns to the source server the following positive reply messages:

### OPNQRYRM (X'2205')

Open query reply message.

## Reply data objects

The following reply data objects can be returned in response to the CNTQRY command:

### QRYDSC (X'241A')

Query answer set description.

### QRYDTA (X'241B')

Query answer set data.



## Error reply messages

In response to the OPNQRY command, the IMS target DDM server can return to the source DDM server the following error reply messages:

Table 81. Possible reply messages for the OPNQRY command

Code point of reply message	Name of reply message	Meaning of reply message
X'121C'	CMDATHRM	Not authorized to command
X'1232'	AGNPRMRM	Permanent agent error
X'1233'	RSCLMTRM	Resource limits reached
X'1245'	PRCCNVRM	Conversational protocol error
X'124C'	SYNTAXRM	Data stream syntax error
X'1250'	CMDNSPRM	Command not supported
X'1251'	PRMNSPRM	Parameter not supported
X'1252'	VALNSPRM	Parameter value not supported
X'1253'	OBJNSPRM	Object not supported
X'1254'	CMDCHKRM	Command check reply message
X'125F'	TRGNSPRM	Target not supported
X'2204'	RDBNACRM	Database not accessed
X'220A'	DSCINVRM	Invalid description
X'220B'	ENDQRYRM	End of query
X'220D'	ABNUOWRM	Abnormal end of unit of work condition
X'220E'	DTAMCHRM	Data descriptor mismatch
X'220F'	QRYPOPRM	Query previously opened
X'2212'	OPNQFLRM	Open query failure
X'2218'	RDBUPDRM	Database update reply message

## OPNQRY examples

```

| OPNQRY only example:
| [ibm][ims][drda][t4] SEND BUFFER: OPNQRY (ASCII) (EBCDIC)
| [ibm][ims][drda][t4] 0000 005BD00100030055 200C004421134BC9 .[.....U ..D!.K. .$.}.....I
| [ibm][ims][drda][t4] 0010 D4E2F14040404040 4040404040404040 ...@@@@@@@@@@@@ MS1
| [ibm][ims][drda][t4] 0020 D5E4D3D3C9C44040 4040404040404040 .....@@@@@@@@ NULLID
| [ibm][ims][drda][t4] 0030 4040E2E8E2E2D5F2 F0F0404040404040 @@.....@@@@@ SYSSN200
| [ibm][ims][drda][t4] 0040 404040405359534C 564C303100010008 @@@@SYSLVL01.... ...<.<.....
| [ibm][ims][drda][t4] 0050 2114000000000005 215D01 !.....!]. .....).

| OPNQRY complete chained request example for SQL SELECT:
| [ibm][ims][drda][t4] SEND BUFFER: EXCSQLSET (ASCII) (EBCDIC)
| [ibm][ims][drda][t4] 0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF 0123456789ABCDEF
| [ibm][ims][drda][t4] 0000 004ED05100010048 2014004421134BC9 .N.Q...H ..D!.K. .+}.....I
| [ibm][ims][drda][t4] 0010 D4E2F14040404040 4040404040404040 ...@@@@@@@@@@@@ MS1
| [ibm][ims][drda][t4] 0020 D5E4D3D3C9C44040 4040404040404040 .....@@@@@@@@ NULLID
| [ibm][ims][drda][t4] 0030 4040E2E8E2E2D5F2 F0F0404040404040 @@.....@@@@@ SYSSN200
| [ibm][ims][drda][t4] 0040 404040405359534C 564C30310041 @@@@SYSLVL01.A ...<.<....

| [ibm][ims][drda][t4] SEND BUFFER: SQLSTT (ASCII) (EBCDIC)
| [ibm][ims][drda][t4] 0000 0031D0430001002B 2414002353455420 .1.C...+$.#SET ..}.....
| [ibm][ims][drda][t4] 0010 434C49454E542057 524B53544E4E414D CLIENT WRKSTNNAM .<..+.....+.(

```

```

| [ibm][ims][drda][t4] 0020 452027392E36352E 3137342E32352700 E '9.65.174.25'. .....
| [ibm][ims][drda][t4] 0030 00 . .
| [ibm][ims][drda][t4]
| [ibm][ims][drda][t4] SEND BUFFER: PRPSQLSTT (ASCII) (EBCDIC)
| [ibm][ims][drda][t4] 0000 0058D05100020052 200D004421134BC9 .X.Q...R ..D!.K. ..}.....I
| [ibm][ims][drda][t4] 0010 D4E2F14040404040 4040404040404040 ...@@@@@@@@@@@@ MS1
| [ibm][ims][drda][t4] 0020 D5E4D3D3C9C44040 4040404040404040 .....@@@@@@@@ NULLID
| [ibm][ims][drda][t4] 0030 4040E2E8E2E2D5F2 F0F0404040404040 @@.....@@@@@ SYSSN200
| [ibm][ims][drda][t4] 0040 404040405359534C 564C303100010005 @@@@SYSLVL01.... ...<.<.....
| [ibm][ims][drda][t4] 0050 2116F10005214604 !.....!F. ..1.....
| [ibm][ims][drda][t4]
| [ibm][ims][drda][t4] SEND BUFFER: SQLATTR (ASCII) (EBCDIC)
| [ibm][ims][drda][t4] 0000 001CD05300020016 2450000E464F5220 ...S....$P..FOR ..}.....&...|..
| [ibm][ims][drda][t4] 0010 52454144204F4E4C 59200000 READ ONLY .. .....|+<....
| [ibm][ims][drda][t4]
| [ibm][ims][drda][t4] SEND BUFFER: SQLSTT (ASCII) (EBCDIC)
| [ibm][ims][drda][t4] 0000 0029D04300020023 2414001B53656C65 .).C...#$...Sele ..}.....%.
| [ibm][ims][drda][t4] 0010 6374202A2066726F 6D20504844414D56 ct * from PHDAMV .....?_&...(
| [ibm][ims][drda][t4] 0020 41522E7761726400 00 AR.ward.. ..../.....
| [ibm][ims][drda][t4]
| [ibm][ims][drda][t4] SEND BUFFER: OPNQRY (ASCII) (EBCDIC)
| [ibm][ims][drda][t4] 0000 005BD00100030055 200C004421134BC9 .[.....U ..D!.K. .$.}.....I
| [ibm][ims][drda][t4] 0010 D4E2F14040404040 4040404040404040 ...@@@@@@@@@@@@ MS1
| [ibm][ims][drda][t4] 0020 D5E4D3D3C9C44040 4040404040404040 .....@@@@@@@@ NULLID
| [ibm][ims][drda][t4] 0030 4040E2E8E2E2D5F2 F0F0404040404040 @@.....@@@@@ SYSSN200
| [ibm][ims][drda][t4] 0040 404040405359534C 564C303100010008 @@@@SYSLVL01.... ...<.<.....
| [ibm][ims][drda][t4] 0050 2114000000000005 215D01 !.....!]. .....).

```

**Related reference:**

- “OPNQFLRM reply message (X'2212’)” on page 337
- “DLIFUNC command object (X'CC05’)” on page 290
- “INAIIB command object (X'CC01’)” on page 302
- “RTRVFLD command object (X'CC04’)” on page 312
- “SSALIST command object (X'CC06’)” on page 325
- “OPNQRYRM reply message (X'2205’)” on page 338
- “QRYPOPRM reply message (X'220F’)” on page 341

## PRPSQLSTT command (X'200D')

The distributed data management (DDM) architecture Prepare SQL Statement command (PRPSQLSTT) dynamically binds a SQL statement to a section in an existing database (RDB) package.

### Format

►►—DSSHDR—LL—CP—SQLSTTGRP—►►

### Parameters

**DSSHDR**

The 6-byte header field containing information about the DSS.

**LL** A 2-byte field that has the length of the PRPSQLSTT command.

**CP(X'200D')**

The 2-byte codepoint of the PRPSQLSTT command.

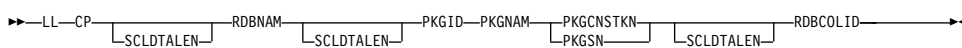
**PKGNAMCSN(X'2113')**

Specifies the fully qualified package name, consistency token, and section number within the package that is used to execute the SQL. The

PKGNAMECSN can have one of the following formats, depending on the length of the RDBNAM, RDBCOLID, and PKGID contained therein:

- RDBNAM, RDBCOLID, and PKGID each have a length of 18. This format of the PKGNAMECSN is identical to the sole format used before DDM Level 7, where the length is fixed at 68. The use of the SCLDTALEN is disallowed with this format.
- At least one of RDBNAM, RDBCOLID, or PKGID has a length > 18. This format of the PKGNAMECSN requires the SCLDTALEN precedes each of the RDBNAM, RDBCOLID, and PKGID. With this format, the PKGNAMECSN has a minimum length of 75 and a maximum length of 785.

**Format:**



**Parameters:**

**RDBNAM**

An 18- to 255-byte character field that represents the relational database name.

**PKGID**

An 18- to 255-byte character field that represents the relational database package identifier.

**PKGNAME**

An 18- to 255-byte character field that specifies the fully qualified name of a relational database package.

**PKGCNSTKN**

An 8-byte character field that verifies that the requester's application and the relational database package are synchronized. Mutually exclusive with PKGSN.

**PKGSN**

A 2-byte short field that represents the Section Number. Mutually exclusive with PKGCNSTKN.

**SCLDTALEN**

Specifies the length of the instance variable that immediately follows:

- RDB collection identifier (RDBCOLID)
- Relational database name (RDBNAM)
- RDB package identifier (PKGID)

This token is allowed only when the length of one or more of the parameters listed is greater than 18 bytes.

**Note:** If the length of any one of RDBNAM, RDBCOLID, or PKGID exceeds 18 bytes, the SCLDTALEN is mandatory and must precede each of the three parameters RDBNAM, RDBCOLID, and PKGID. Otherwise, the SCLDTALEN is disallowed.

**RDBCOLID**

An 18- to 255-byte character field that identifies a unique collection of objects that are contained in a relational database. It is used for user-defined grouping.

**RTNSQLDA(X'2116')**

Return SQL Descriptor Area controls whether to return an SQL descriptor area

that applies to the SQL statement this command identifies. The target SQLAM obtains the SQL descriptor area by performing an SQL DESCRIBE function on the statement after the statement has been prepared.

▶▶—LL—CP—VALUE—————▶▶

**Parameters:**

**VALUE**

TRUE (X'F1') – Indicates an SQLIMSDA is returned.

FALSE (X'F0') – Indicates an SQLIMSDA is not returned.

**Note:** IMS will always send a 0x'01' from the Universal Driver.

**TYPSQLDA(X'2146')**

Type of SQL Descriptor Area.

▶▶—LL—CP—TYPE—————▶▶

**Parameters:**

**TYPE**

A single-byte signed number that specifies the type of SQLIMSDA to return for the command.

**0** Standard output SQLIMSDA. This type is supported for ODBM.

**1** Standard input SQLIMSDA. This type is supported for ODBM.

**2** Light output SQLIMSDA

**3** Light input SQLIMSDA

**4** Extended output SQLIMSDA

**5** Extended input SQLIMSDA

**MONITOR(X'1900')**

▶▶—LL—CP—FLAGS—————▶▶

**FLAGS**

A 4-byte flag value.

**PRPSQLSTT examples**

The following example shows PRPSQLSTT that is part of the request to an OPNQRY call.

	SEND BUFFER: PRPSQLSTT	(ASCII)	(EBCDIC)
[ibm][ims][drda][t4] 0000	0058D05100020052 200D004421134BC9	.X.Q...R ..D!.K. ..}	.....I
[ibm][ims][drda][t4] 0010	D4E2F14040404040 4040404040404040	...@@@	MS1
[ibm][ims][drda][t4] 0020	D5E4D3D3C9C44040 4040404040404040	.....@@@	NULLID
[ibm][ims][drda][t4] 0030	4040E2E8E2E2D5F2 F0F0404040404040	@@.....@@@	SYSSN200
[ibm][ims][drda][t4] 0040	404040405359534C 564C303100010005	@@@SYSLVL01....	...<.<.....
[ibm][ims][drda][t4] 0050	2116F10005214604	!....!F.	..1.....

**RLSE command (X'C802')**

Use the distributed data management (DDM) architecture RLSE command to release any database locks that are held by the application.

## Format

►—DSSHDR—LL—CP—PCBNAME—►

## Parameters

### DSSHDR

The 6-byte header that contains information about the data stream structure (DSS).

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'C802', the 2-byte codepoint of the RLSE command.

### PCBNAME

A required parameter that specifies the PCB name that uniquely identifies the query made by a DL/I call. The PCB name is specified as a character string. The value is initially sent with the original OPNQRY command. The same value must subsequently be sent in commands such as CNTQRY, CLSQRY, and RLSE for proper correlation with the original OPNQRY call. The codepoint for the PCBNAME parameter is X'C907'.

## Chained command objects

No command objects are chained to the RLSE command.

## Positive reply messages

In response to the RLSE command, the IMS target DDM server returns to the source server the following positive reply message:

### RLSERM (X'CA03')

The Release Locks Reply Message indicates to the requester that an RLSE command has completed normally.

## Chained reply data objects

No reply data objects are returned in response to the RLSE command.

## Error reply messages

In response to the RLSE command, the IMS target DDM server can return to the source DDM server the following error reply messages:

*Table 82. Possible error reply messages for the RLSE command*

Codepoint of reply message	Name of reply message	Meaning of reply message
X'121C'	CMDATHRM	Not authorized to command
X'1232'	AGNPRMRM	Permanent agent error
X'1233'	RSCLMTRM	Resource limits reached
X'124C'	SYNTAXRM	Data stream syntax error
X'1250'	CMDNSPRM	Command not supported
X'1251'	PRMNSPRM	Parameter not supported
X'1252'	VALNSPRM	Parameter value not supported

Table 82. Possible error reply messages for the RLSE command (continued)

Codepoint of reply message	Name of reply message	Meaning of reply message
X'1254'	CMDCHKRM	Command check reply message
X'125F'	TRGNSPRM	Target not supported

**Related reference:**

“RLSERM reply message (X'CA03)’” on page 347

## RTRVFLD command object (X'CC04')

Use the distributed data management (DDM) architecture RTRVFLD command object to specify the field that the client wants to retrieve data from.

### Format

►►—DSSHDR—LL—CP—RECOFF—FLDLEN—◄◄

### Parameters

**DSSHDR**

The 6-byte header that contains information about the data stream structure (DSS).

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'CC04', the 2-byte codepoint of the RTRVFLD command object.

**RECOFF**

A 4-byte signed integer that contains the offset of the field within the hierarchic path I/O area that is to be returned from the DL/I call.

**FLDLEN**

A 4-byte signed integer that contains the length of the field.

**Related reference:**

“OPNQRY command (X'200C)’” on page 304

## RTRVFLDREL command object (X'CC0B')

Use the distributed data management (DDM) architecture FLDENTRYREL (relative retrieve field) command object to specify which field the client wants to retrieve data from.

**Restriction:** The RTRVFLDREL command object is supported only with an ODBM DDM level of 1, 2, 3 or 1, 3.

### Format

►►—DSSHDR—LL—CP—SEGMOFF—FLDLEN—SEGMID—◄◄

### Parameters

**DSSHDR**

The 6-byte header that contains information about the data stream structure (DSS).

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'CC0B', the 2-byte codepoint of the RTRVFLDREL command object.

**SEGMOFF**

A required, 4-byte, signed integer that specifies the relative offset of the target field from the start of the parent segment.

**SEGMID**

A required, 1-byte, signed integer that specifies which segment in the SEGMLIST the field is referenced from. This value is relative to 1 rather than 0.

**FLDLEN**

The length of the target field.

## SECCHK command (X'106E')

The distributed data management (DDM) architecture SECCHK command passes the user information from the source server to the target security manager of the IMS target server to authenticate the user with RACF or another security product.

When security checking is active for the IMS target server, the SECCHK command must be preceded by the ACCSEC command.

### Format

►►—DSSHDR—LL—CP—PASSWORD—SECMEC—USRID—◄◄

### Parameters

**DSSHDR**

The 6-byte header that contains information about the data stream structure (DSS).

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'106E', the 2-byte codepoint of the SECCHK command.

**SECMEC**

A required parameter that specifies the security mechanism agreed upon by the source server and the target server. For IMS, specify USRIDPWD.

The security mechanism is negotiated between the source server and the target server by using the ACCSEC command and the ACCSECRD reply object.

**USRID**

A required, variable-length parameter that specifies the user ID of the source application program as a character string. The length can be from 1 to 255 characters.

**PASSWORD**

A required, variable-length parameter that specifies the password of the source application program as a character string. The length can be from 1 to 255 characters.

### Usage

IMS uses a user ID and password to check security; therefore the value of the SECMEC parameter specifies the DDM USRIDPWD security mechanism.

If no errors occur during the processing of the SECCHK command, the IMS target server returns the SECCHKRM reply message to indicate the acceptability of the security information.

The SECCHK command must be preceded by the ACCSEC command.

## Chained command objects

No command objects can be chained to the SECCHK command.

## Positive reply messages

In response to the SECCHK command, the IMS target DDM server returns to the source server the following positive reply message:

### SECCHKRM (X'1219')

Security check reply message.

## Error reply messages

In response to the SECCHK command, the IMS target DDM server can return to the source DDM server the following error reply messages:

*Table 83. Possible error reply messages for the SECCHK command*

Codepoint of reply message	Name of reply message	Meaning of reply message
X'1218'	MGRDEPRM	Manager dependency error
X'121C'	CMDATHRM	Not authorized to command
X'1232'	AGNPRMRM	Permanent agent error
X'1233'	RSCLMTRM	Resource limits reached
X'123C'	INVRQSRM	Invalid request
X'1245'	PRCCNVRM	Conversational protocol error
X'124C'	SYNTAXRM	Data stream syntax error
X'1250'	CMDNSPRM	Command not supported
X'1251'	PRMNSPRM	Parameter not supported
X'1252'	VALNSPRM	Parameter value not supported
X'1253'	OBJNSPRM	Object not supported
X'1254'	CMDCHKRM	Command check reply message
X'125F'	TRGNSPRM	Target not supported

## SEGMLIST command object (X'CC0A')

Use the distributed data management (DDM) architecture SEGMLIST (Segment List) command object to specify the minimum and maximum length of each segment being retrieved or updated.

**Restriction:** The SEGMLIST command object is supported only with an ODBM DDM level of 1, 3 or 1, 2, 3.



## Format



## Parameters

### DSSHDR

The 6-byte header that contains information about the data stream structure (DSS).

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'CC0A', the 2-byte codepoint of the SEGMLIST command object.

### COUNT

A 1-byte, signed value that counts the number of segments in a record that is being retrieved or updated. The total number of segments in a record is limited to 15. The value of the COUNT parameter corresponds to the number of instances of the MINMAX parameter included in the command object. This value is required.

### MINMAX

An 8-byte field that is divided into two 4-byte signed integers. The first integer is the minimum number of bytes in a segment and the second integer is the maximum number of bytes. If these integers are equal, the segment is fixed length.

## SQLATTR command (X'2450')

The distributed data management (DDM) architecture SQL Statement Attributes command (SQLATTR) specifies the SQL statement attributes being prepared.

## Format



## Parameters

### DSSHDR

The six byte header field containing information about the DSS.

**LL** A two byte field that has the length of the SQLATTR command.

### CP(X'2450')

The 2-byte codepoint of the SQLATTR command.

### SQLSTGRP

SQL Statement Group Description.

#### Format:



#### Parameters:

### SQLSTATEMENT\_m

A variable length string containing the SQL statement.

## SQLSTATEMENT\_s

A variable length string containing the SQL statement.

## SQLATTR examples

The following example shows SQLATTR that is part of the request to an OPNQRY call.

[ibm] [ims] [drda] [t4]	SEND BUFFER: SQLATTR	(ASCII)	(EBCDIC)
[ibm] [ims] [drda] [t4] 0000	001CD05300020016 2450000E464F5220	...S...\$P..FOR	..}.....&... ..
[ibm] [ims] [drda] [t4] 0010	52454144204F4E4C 59200000	READ ONLY ..	..... +<....

## SQLCARD command (X'2408')

The distributed data management (DDM) architecture SQL Descriptor Area Row Description with SQL Communications Area command (SQLCARD) provides metadata information about the columns being retrieved along with the communications area.

### Format

►►—DSSHDR—LL—CP—SQLCAGRP—————◄◄

### Parameters

#### DSSHDR

The 6-byte header field containing information about the DSS.

**LL** A 2-byte field that has the length of the SQLCARD command.

#### CP(X'2408')

The 2-byte code point of the SQLCARD command.

#### SQLCAGRP

SQL Communications Area Group Description.

#### Format:

►►—FLAG—SQLCODE—SQLSTATE—SQLERRPROC—SQLCAXGRP—SQLDIAGGRP—————◄◄

#### Parameters:

##### FLAG

A 1-byte field that determines if the value for the SQLCAGRP is null. A null indicator is denoted with the value X'FF'.

##### SQLCODE

A 4-byte integer field that contains the return code that is sent by the database manager after completion of each SQL statement.

##### SQLSTATE

A 5-byte character field that contains the outcome of the most recently executed SQL statement.

##### SQLERRPROC

An 8-byte character field that contains the name of the CSECT that detected the error reported by the SQLIMSCODE.

##### SQLCAXGRP

SQL Communications Area Exceptions Group Description.

**Format:**

►►—FLAG—SQLERRD—SQLWARN—SQLRDBNAME—SQLERRMSG\_m—SQLERRMSG\_s—►►

**Parameters:**

**FLAG**

A 1-byte field that determines if the value for the SQLCAXGRP is null. A null indicator is denoted with the value X'FF'.

**SQLERRD**

Six 1-byte integer fields whose values are used to diagnose error conditions

**SQLWARN**

Eleven 1-byte character fields that represents SQLIMSWARN0 to SQLIMSWARNA.

**SQLRDBNAME**

A variable character string that shows the name of the remote database.

**SQLERRMSG\_m**

A variable character string that contains one or more tokens, separated by X'FF', that are substituted for variables in the descriptions of error conditions. It may contain truncated tokens. A message length of 70 bytes indicates a possible truncation.

**SQLERRMSG\_s**

A variable character string that contains one or more tokens, separated by X'FF', that are substituted for variables in the descriptions of error conditions. It may contain truncated tokens. A message length of 70 bytes indicates a possible truncation.

**SQLDIAGGRP**

SQL Descriptor Optional Group Description

**Format:**

►►—FLAG—SQLDIAGSTT—SQLDIAGCI—SQLDIAGCN—►►

**Parameters:**

**FLAG**

A 1-byte field that determines if the value for the SQLCAXGRP is null. A null indicator is denoted with the value X'FF'.

**SQLDIAGSTT**

SQL Diagnostics Statement Group Description.

**SQLDIAGCI**

SQL Diagnostics Condition Information Array.

**SQLDIAGCN**

SQL Diagnostics Connection Array.

**SQLCARD examples**

The following example shows SQLCARD that is part of the request to an OPNQRY call.

	[ibm]	[ims]	[drda]	[t4]	RECEIVE BUFFER: SQLCARD	(ASCII)	(EBCDIC)
	[ibm]	[ims]	[drda]	[t4]	0000 0059D05300030053 2408006400000030	.Y.S...S\$.d...0	..}.....
	[ibm]	[ims]	[drda]	[t4]	0010 3230303053514C52 4930314600010004	2000SQLRI01F....	.....<.....
	[ibm]	[ims]	[drda]	[t4]	0020 8001000000000000 0000000000000000	.....	.....
	[ibm]	[ims]	[drda]	[t4]	0030 0000000000202020 2020202020202020	.....	.....
	[ibm]	[ims]	[drda]	[t4]	0040 001253414D504C45 2020202020202020	..SAMPLE	....(&<.....
	[ibm]	[ims]	[drda]	[t4]	0050 2020202000000000 FF		

## SQLDARD command (X'2411')

The distributed data management (DDM) architecture SQL Descriptor Area Row Description with SQL Communications Area command (SQLDARD) provides metadata information about the columns being retrieved along with the communications area.

### Format

►►—DSSHDR—LL—CP—SQLCARD—SQLDHGRP—SQLNUMGRP—SQLDAGRP—►►

### Parameters

#### DSSHDR

The 6-byte header field containing information about the DSS.

**LL** A 2-byte field that has the length of the SQLDARD command.

#### CP(X'2411')

The 2-byte codepoint of the SQLDARD command.

#### SQLCARD

SQL Communications Area Row Description

#### SQLDHGRP

SQL Descriptor Header Group Description (column metadata that applies to all fields in the result set).

#### Format:

►►—FLAG—SQLDHOLD—SQLDRETURN—SQLDSCROLL—SQLDSENSITIVE—SQLDFCODE—SQLDKEYTYPE—SQLDRDBNAM—►►

►—SQLDSHEMA\_m—SQLDSHEMA\_s—►►

#### Parameters:

##### FLAG

A 1-byte field that determines if the value for the SQLDHGRP is null. A null indicator is denoted with the value X'FF'.

##### SQLDHOLD

A 2 byte short field. This field can have a value of 0 or 1. A value of 1 indicates this statement is related to a cursor which is defined using the WITH HOLD clause. Otherwise, the value is 0.

##### SQLDRETURN

A 2-byte short field.

##### SQLDSCROLL

A 2-byte short field.

##### SQLDSENSITIVE

A 2-byte short field.

```

|
|      SQLDFCODE
|      A 2-byte short field.
|
|      SQLDKEYTYPE
|      A 2-byte short field.
|
|      SQLDRDBNAM
|      A variable character string that shows the name of the remote database.
|
|      SQLDSHEMA_m
|      A variable character string that shows the name of the schema.
|
|      SQLDSHEMA_s
|      A variable character string that shows the name of the schema.
|
| SQLNUMBRP
| SQL Number of Elements Group Descriptions.
|
| Format:
|
|      ►►—Number of Columns—————►►
|
|
| Parameters:
|
| Number of Columns
|      A 2-byte short that represents the number of columns being returned by
|      the query.
|
| SQLDAGRP
| SQL Data Area Group Description (column metadata specific to each column).
|
| Format:
|
|      ►►—SQLPRECISION—SQLSCALE—SQLLENGTH—SQLTYPE—SQLCCSID—SQLDOPTGRP—————►►
|
|
| Parameters:
|
| SQLPRECISION
|      A 2-byte short field representing the precision for the column.
|
| SQLSCALE
|      A 2-byte short field representing the scale for the column.
|
| SQLLENGTH
|      An 8-byte field representing the length of the column in bytes.
|
| SQLTYPE
|      A 2-byte short field representing the data type of the column.
|
| SQLCCSID
|      A 2-byte short field representing the CCSID of the column.
|
| SQLDOPTGRP
|      SQL Descriptor Optional Group Description
|
| Format:
|
|      ►►—FLAG—SQLUNNAMED—SQLNAME_m—SQLNAME_s—SQLLABEL_m—SQLLABEL_s—SQLCOMMENTS_m—SQLCOMMENTS_s—————►►
|
|      ►—SQLUDTGRP—SQLDXGRP—————►►
|
|
| Parameters:

```

**FLAG**  
 A 1-byte field that determines if the value for the SQLDHGRP is null.  
 A null indicator is denoted with the value X'FF'.

**SQLUNNAMED**  
 A 2-byte short field.

**SQLNAME\_m**  
 A variable character string that shows the column name.

**SQLNAME\_s**  
 A variable character string that shows the column name.

**SQLLABEL\_m**  
 A variable character string

**SQLLABEL\_s**  
 A variable character string

**SQLCOMMENTS\_m**  
 A variable character string

**SQLCOMMENTS\_s**  
 A variable character string

**SQLUDTGRP:**  
 SQL User-Defined Data Group Description

**Format:**

►►—FLAG—SQLUDTXTYPE—SQLUDTRDB—SQLUDTSHEMA\_m—SQLUDTSHEMA\_s—SQLUDTNAME\_m—SQLUDTNAME\_s————►►

**Parameters:**

**FLAG**  
 A 1-byte field that determines if the value for the SQLDHGRP is null. A null indicator is denoted with the value X'FF'.

**SQLUDTXTYPE**  
 A 4-byte integer field.

**SQLUDTRDB**  
 A variable character string.

**SQLUDTSHEMA\_m**  
 A variable character string.

**SQLUDTSHEMA\_s**  
 A variable character string.

**SQLUDTNAME\_m**  
 A variable character string.

**SQLUDTNAME\_s**  
 A variable character string.

**SQLDXGRP**  
 SQL Descriptor Extended Group Description.

**Format:**

►►—FLAG—SQLXKEYMEM—SQLXUPDATEABLE—SQLXGENERATED—SQLXPARMMODE—SQLXRDBNAM—SQLXCORNAME\_m————►►

►SQLXCORNAME\_s—SQLXBASENAME\_m—SQLXBASENAME\_s—SQLXSCHEMA\_m—SQLXSCHEMA\_s—SQLXNAME\_m—SQLXNAME\_s—►

**Parameters:**

**FLAG**

A 1-byte field that determines if the value for the SQLDHGRP is null. A null indicator is denoted with the value X'FF'.

**SQLXKEYMEM**

A 2-byte short field.

**SQLXUPDATEABLE**

A 2-byte short field.

**SQLXGENERATED**

A 2-byte short field.

**SQLXPARMODE**

A 2-byte short field.

**SQLXRDBNAM**

A variable character string that shows the name of the remote database.

**SQLXCORNAME\_m**

A variable character string that shows the name of the table.

**SQLXCORNAME\_s**

A variable character string that shows the name of the table.

**SQLXBASENAME\_m**

A variable character string that shows the name of the table.

**SQLXBASENAME\_s**

A variable character string that shows the name of the table.

**SQLXSCHEMA\_m**

A variable character string that shows the name of the schema.

**SQLXSCHEMA\_s**

A variable character string that shows the name of the schema.

**SQLXNAME\_m**

A variable character string that shows the name of the column.

**SQLXNAME\_s**

A variable character string that shows the name of the column.

**SQLDARD examples**

The following example shows SQLDARD that is part of the request to an OPNQRY call.

	RECEIVE BUFFER: SQLDARD	(ASCII)	(EBCDIC)
[ibm][ims][drda][t4] 0000	0185D0530002017F 241100000000000030	...S....\$......0	.e)....".....
[ibm][ims][drda][t4] 0010	3030303053514C30 3930373000000000	0000SQL09070....	.....<.....
[ibm][ims][drda][t4] 0020	0000000000010000 0040010000000000	.....@.....	.....
[ibm][ims][drda][t4] 0030	0000000000202020 2020202020202020	.....	.....
[ibm][ims][drda][t4] 0040	001253414D504C45 2020202020202020	..SAMPLE	....(&<.....
[ibm][ims][drda][t4] 0050	2020202000000000 FF00010000000000	.....	.....
[ibm][ims][drda][t4] 0060	0000550000000000 0000000003000000	..U.....	.....
[ibm][ims][drda][t4] 0070	0000040000000000 0000F10100000000	.....	.....1.....
[ibm][ims][drda][t4] 0080	000005454D504E4F 0000000000000000	..EMPNO.....	....(&+ .....
[ibm][ims][drda][t4] 0090	0000FF0000000000 0000000000065341	.....SA	.....
[ibm][ims][drda][t4] 00A0	4D504C450006454D 504D444300000006	MPLE..EMPMDC....	(&<....(&(.....

```

| [ibm][ims][drda][t4] 00B0 454D504D44430000 0008524943485452 EMPMDC....RICHTR .(&(.....
| [ibm][ims][drda][t4] 00C0 414E00000005454D 504E4F0000000000 AN....EMPNO.... .+.....(&+|.....
| [ibm][ims][drda][t4] 00D0 0004000000000000 00F1010000000000 .....1.....
| [ibm][ims][drda][t4] 00E0 0004444550540000 0000000000000000 ..DEPT..... &.....
| [ibm][ims][drda][t4] 00F0 FF00000000000000 0000000653414D50 .....SAMP .....(&
| [ibm][ims][drda][t4] 0100 4C450006454D504D 444300000006454D LE..EMPMDC....EM <....(&(.....(
| [ibm][ims][drda][t4] 0110 504D444300000008 524943485452414E PMDC....RICHTRAN &(.....+
| [ibm][ims][drda][t4] 0120 0000000444455054 000000000000400 ....DEPT..... &.....
| [ibm][ims][drda][t4] 0130 000000000000F101 000000000000344 .....D .....1.....
| [ibm][ims][drda][t4] 0140 4956000000000000 00000000FF000000 IV..... .....
| [ibm][ims][drda][t4] 0150 0000000000000006 53414D504C450006 .....SAMPLE.. .....(&<...
| [ibm][ims][drda][t4] 0160 454D504D44430000 0006454D504D4443 EMPMDC....EMPMDC .(&(.....(&(..
| [ibm][ims][drda][t4] 0170 0000000852494348 5452414E00000003 ....RICHTRAN.... .....+....
| [ibm][ims][drda][t4] 0180 4449560000 DIV.. .....

```

## SQLDTA command (X'2412')

SQL Program Variable Data (SQLDTA) consists of input data to a SQL statement that a relational database (RDB) is running. It also includes a description of the data.

### Format

```

▶▶—DSSHDR—LL—CP—┬──FDOEXT──┬──FDODSC—FDODTA──┬──FDOOFF──▶▶
                    └──┬──┘           └──┬──┘

```

### Parameters

#### DSSHDR

The 6-byte header field that contains information about the DSS.

**LL** A 2-byte field that contains the length of the SQLDTA command.

#### CP(X'2412')

The 2-byte code point of the SQLDTA command.

#### FDOEXT(X'147B')

A scalar object that contains extent data for each SDA that a Formatted Data Object Architecture descriptor (FDODSC) describes. There is an FDOEXT entry for each field definition in the SQLDTAGRP in the FDODSC. The FDOEXT specification that corresponds to a field definition in the FDODSC defines the number of times that field is repeated in the FDODTA object.

#### Format:

```

▶▶—LL—CP—BYTSTRDR—▶▶

```

#### Parameters:

#### BYTSTRDR

A required byte-string data representation, whose length is an even number, such as X'01010101' or X'3D2B11'.

#### FDODSC(X'0010')

A string that is a DDM scalar whose value is a Formatted Data Object Content Architecture (FD:OCA) descriptor or a segment of an FD:OCA descriptor. An FDODSC consists of one or more FD:OCA triplets that describe the data fields that are contained in another scalar object.

#### Format:



►►—LL—CP—BYTSTRDR—————►►

**Parameters:**

**BYTSTRDR**

A required byte-string data representation, whose length is an even number, such as X'01010101' or X'3D2B11'.

**FDODTA(X'147A')**

A scalar object that contains data that a Formatted Data Object Architecture descriptor (FDODSC) describes. The FDODSC might be present with the Formatted Data Object Data (FDODTA), or it might be implicitly defined based on the context of the command in which the FDODTA is used.

**Format:**

►►—LL—CP—BYTSTRDR—————►►

**Parameters:**

**BYTSTRDR**

A required byte-string data representation, whose length is an even number, such as X'01010101' or X'3D2B11'.

**FDOOFF(X'147D')**

A scalar object that contains offset data for each SDA that a Formatted Data Object Architecture descriptor (FDODSC) describes. There is an FDOOFF entry for each field definition in the SQLDTAGRP in the FDODSC. The FDOOFF specification corresponding to a field definition in the FDODSC defines the offset to the start of the data entry in the FDODTA. The offset value for the first data array is 0.

**Format:**

►►—LL—CP—BYTSTRDR—————►►

**Parameters:**

**BYTSTRDR**

A required byte-string data representation, whose length is an even number, such as X'01010101' or X'3D2B11'.

**Formatted Data Object Content Architecture (FD:OCA)**

The Formatted Data Object Content Architecture (FD:OCA) is an architecture for describing, organizing, and manipulating a linear stream of data.

DDM uses FD:OCA primarily for the description of data for relational database (RDB) access. A complete description of FD:OCA, including how to build and interpret FD:OCA descriptors, is in the FD:OCA Reference.

The functions of FD:OCA are specified through an FD:OCA descriptor, which consists of data structures called triplets. Attribute triplets describe the representation and layout of data in a data stream. Generator triplets specify how the data is manipulated to produce an output data stream.

An FD:OCA-containing architecture, such as DDM, transmits data streams and FD:OCA descriptors between communicating systems and starts a presentation

process as needed. The presentation process accepts both the data stream and the FD:OCA descriptor as input and produces a presentation stream as output, as shown in Figure 3-39. The FD:OCA-containing architecture processes any further presentation streams. For example, DDM can forward the presentation stream for storage to an RDB, or it can pass the presentation stream to an application requester.

### An Overview of selected FD:OCA triplets

FD:OCA defines many more attribute and generator triplets than the current level of DDM architecture requires. The following is an overview of the triplets.

#### Scalar Data Arrays (SDA)

SDAs are the triplets that FD:OCA uses for describing data values that are either single items or linear or rectangular arrays of single items that have the same format. DDM uses SDAs primarily to associate data representation specifications with DDM and SQL data types.

#### Group Data Array (GDA)

GDAs are triplets that define a group of data items as a referable unit. The elements of a GDA point (by label) to SDAs, other GDAs, or to RLOs to describe the data items of the group. The GDA can override certain attributes of each data representation.

#### Row Layouts (RLO)

RLOs are triplets that describe:

- A row that contains fields of one or more types
- A table that contains rows of one or more types
- Multi-dimensional, mixed data structures

RLOs describe data streams that consist of multiple unrelated structures. DDM uses RLOs primarily to describe the answer data that the SQL statements return.

#### FD:OCA Descriptors

An FD:OCA descriptor consists of one or more triplets that are laid out consecutively in a byte stream. Triplets that are referenced by other triplets must precede the referencing triplets. Unreferenced triplets are ignored.

### SQLDTA examples

The following example shows SQLDTA that is part of the request to an OPNQRY call.

[ibm][db2][jcc][t4]	SEND BUFFER: SQLDTA	(ASCII)	(EBCDIC)
[ibm][db2][jcc][t4] 0000	002CD00300040026 2412001300100976	.,.....&\$.....v	..}.....
[ibm][db2][jcc][t4] 0010	D003000403000406 71E4D00001000F14	.....q.....	}.....U}.....
[ibm][db2][jcc][t4] 0020	7A00000000000200 00000005	z.....	:.....

### SQLSTT command (X'2414')

The distributed data management (DDM) architecture SQL Statement Row Description (SQLSTT) command contains one SQL statement.

#### Format

►►—DSSHDR—LL—CP—SQLSTTGRP—◄◄

## Parameters

### DSSHDR

The 6-byte header field containing information about the DSS.

**LL** A 2-byte field that has the length of the SQLSTT command.

### CP(X'2414')

The 2-byte codepoint of the SQLSTT command.

### SQLSTTGRP

SQL Statement Group Description.

#### Format:

►►—SQLSTATEMENT\_m—SQLSTATEMENT\_s—►►

#### Parameters:

##### SQLSTATEMENT\_m

A variable length string containing the SQL statement.

##### SQLSTATEMENT\_s

A variable length string containing the SQL statement.

## SQLSTT examples

The following example shows SQLSTT that is part of the request to an OPNQRY call.

SQLSTT for Special Registry:

[ibm][ims][drda][t4]	SEND BUFFER: SQLSTT	(ASCII)	(EBCDIC)
[ibm][ims][drda][t4] 0000	0032D0430001002C 2414002453455420	.2.C...,\$...\$SET	..}.....
[ibm][ims][drda][t4] 0010	434C49454E542057 524B53544E4E414D	CLIENT WRKSTNNAM	<..+.....+.(
[ibm][ims][drda][t4] 0020	452027392E36352E 3135302E32313827	E '9.65.150.218'	.....
[ibm][ims][drda][t4] 0030	0000	..	..

SQLSTT for Select Statement:

[ibm][ims][drda][t4]	SEND BUFFER: SQLSTT	(ASCII)	(EBCDIC)
[ibm][ims][drda][t4] 0000	0029D04300020023 2414001B53656C65	.)C...#\$....Sele	..}.....%.
[ibm][ims][drda][t4] 0010	6374202A2066726F 6D20504844414D56	ct * from PHDAMV	.....?_&...(.
[ibm][ims][drda][t4] 0020	41522E7761726400 00	AR.ward..	..../....

## SSALIST command object (X'CC06')

Use the distributed data management (DDM) architecture SSALIST command object to contain the list of segment search argument (SSA) objects to qualify the DL/I call.

### Format

►►—DSSHDR—LL—CP—SSACOUNT—SSA—►►

## Parameters

### DSSHDR

The 6-byte header that contains information about the data stream structure (DSS).

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'CC06', the 2-byte codepoint of the SSALIST command object.

### SSACOUNT

Required. The number of SSAs in the SSAList, which is specified as a 2-byte value. The valid range is 1 to 15.

### SSA

An optional byte string that contains an SSA object to be used in a DL/I database call.

#### Related reference:

“EXCSQLIMM command (X'200A’)” on page 294

“OPNQRY command (X'200C’)” on page 304

“SSA parameter (X'C906’)” on page 357

“SSACOUNT parameter (X'C905’)” on page 358

---

## DDM reply messages and reply objects

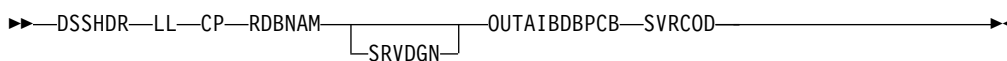
The IMS target server communicates with source server applications by using reply messages defined by the distributed data management (DDM) architecture. Some of the reply messages used by IMS contain parameters, values, or reply objects that are unique to IMS.

### ABNUOWRM reply message (X'220D')

The distributed data management (DDM) architecture ABNUOWRM (abnormal end unit of work condition) reply message indicates that the current unit of work ended abnormally due to some action at the target server.

This reply message can be a result of a deadlock resolution, operator intervention, or other similar situations that cause the database to roll back the current unit of work.

#### Format



#### Parameters

##### DSSHDR

The 6-byte header that contains information about the data stream structure (DSS).

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'220D', the 2-byte codepoint of the ABNUOWRM reply message.

##### RDBNAM

Codepoint X'2110'. The IMS PSB name that identifies the target database.

##### SRVDGN

An optional parameter (X'1153') that contains server diagnostic information. The data portion is a string with a maximum length of 32,767 bytes. The string can be any information the server sends to aid in problem diagnosis. For more information about SRVDGN, see *DRDA, Version 4, Volume 3: Distributed Data Management (DDM) Architecture*, which is available from The Open Group at [www.opengroup.org](http://www.opengroup.org).

##### OUTAIBDBPCB

A required parameter (X'CC02') that contains the AIB and DBPCB data sent from the target to the source.

**SVRCOD**

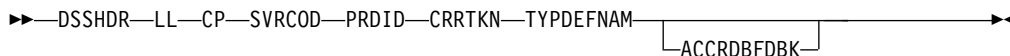
A required parameter (X'1149') that contains the severity code. SVRCOD consists of a 2-byte length field (LL), a 2-byte codepoint (CP), and the data. The data is a 2-byte binary value. The following list describes the possible 2-byte values:

- 0 INFO – Information only severity code
- 4 WARNING – Warning severity code
- 8 ERROR – Error severity code
- 16 SEVERE – Severe error severity code
- 32 ACCDMG – Access damage severity code
- 64 PRMDMG – Permanent damage severity code
- 128 SESDMG – Session damage severity code

**ACCRDBRM reply message (X'2201')**

The distributed data management (DDM) architecture ACCRDBRM (access to database completed) reply message specifies that the named database in the previous ACCRDB command is available to the client for processing.

**Format**



**Parameters**

**DSSHDR**

The 6-byte header that contains information about the data stream structure (DSS).

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'2201', the 2-byte codepoint of the ACCRDBRM reply message.

**ACCRDBFDBK**

An optional parameter (X'CC0D') that contains the **psbfbkStream** and **aliasfbkStream** data sent from the target to the source.



**Parameters**

**LL** The length of ACCRDBFDBK. This length includes the LL and CP.

**CP** X'CC0D' , the 2 byte codepoint of ACCRDBFDBK.

The **psbfbkStream** and **aliasfbkStream** parameters data structure is as follows:

Table 84. The `psbfdbkStream` and `aliasfdbkStream` parameters data structure

Offset	Length	Name	Description
0	1	PSB name null indicator	Binary integer <ul style="list-style-type: none"> <li>• X'00' - indicates that the rest of the data follows.</li> <li>• X'FF' - indicates that there is no data and the total length is one byte.</li> </ul>
1	1	Length	Binary integer <ul style="list-style-type: none"> <li>• Length of the data including the length itself.</li> </ul>
2	8	PSB name	Character string Length of the PSB name.
1 or 10	1	Alias name null indicator	Binary integer <ul style="list-style-type: none"> <li>• X'00' - indicates that the rest of the data follows.</li> <li>• X'FF' - indicates that there is no data and the total length is one byte.</li> </ul>
2 or 11	1	Length	Binary integer <ul style="list-style-type: none"> <li>• Length of the data including the length itself.</li> </ul>
3 or 12	4	Alias name	Character string Length of the alias name.

#### SVRCOD

A required parameter (X'1149') that contains the severity code. SVRCOD consists of a 2-byte length field (LL), a 2-byte codepoint (CP), and the data. The data is a 2-byte binary value. The following list describes the possible 2-byte values:

- 0 INFO – Information only severity code
- 4 WARNING – Warning severity code
- 8 ERROR – Error severity code
- 16 SEVERE – Severe error severity code
- 32 ACCDMG – Access damage severity code
- 64 PRMDMG – Permanent damage severity code
- 128 SESDMG – Session damage severity code

#### PRDID

Product Specific ID. The release level of the DDM server.

This is a required parameter for the ACCRDB command.

**CRRTKN**

A required correlation token that the source and the target servers use to correlate the work for an application.

**TYPDEFNAM**

A required parameter (X'002F') that specifies the name of the data type definition. TYPDEFNAM consists of a 2-byte specification of length (LL), a 2-byte codepoint (CP), and the VALUE. The VALUE is reserved and must be QTDSQL370, which is the general EBCDIC SQL type definition for machines that use EBCDIC strings, IEEE floating-point numbers, and non-byte-reversed floating-point and integer numbers.

**Related reference:**

"ACCRDB command (X'2001')" on page 283

**ACCSECRD reply object (X'14AC')**

The distributed data management (DDM) architecture ACCSECRD (access security reply data) reply object contains the security information from the security manager of the target server. This information is returned in response to the ACCSEC command.

**Format**

►►—DSSHDR—LL—CP—SECMEC—————►►

**Parameters****DSSHDR**

The 6-byte header that contains information about the data stream structure (DSS).

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'14AC', the 2-byte codepoint of the ACCSECRD reply object.

**SECMEC**

Specifies the security mechanism or mechanisms supported by the target server. If the target server supports the DDM security mechanism specified by the source DDM server on the ACCSEC command, the value of SECMEC in the ACCSECRD reply object is the same as the value of SECMEC in the ACCSEC command. If the target server does not support the security mechanism specified by the source server, SECMEC contains one or more values that identify the security mechanisms supported by the target server.

**Usage**

In a successful exchange, the IMS target server confirms the type of security checking requested by the source application program by returning the ACCSECRD reply object with the same value in SECMEC parameter as was submitted in the ACCSEC command.

If an error is detected in processing the ACCSEC command, a SECCHKCD is returned with the ACCSECRD. The SECCHKCD parameter has an implied severity code of ERROR. Before the SECCHK command can be sent to authenticate the connection, the ACCSEC command must be sent again to generate a new set of instance variables on the ACCSECRD.

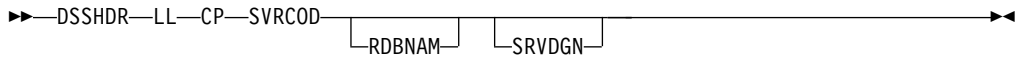
**Related reference:**

“ACCSEC command (X'106D’)” on page 285

## AGNPRMRM reply message (X'1232')

The distributed data management (DDM) architecture AGNPRMRM (permanent agent error) reply message indicates that the requested command could not be completed because the target system detected a permanent error condition.

### Format



### Parameters

#### DSSHDR

The 6-byte header that contains information about the data stream structure (DSS).

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'1232', the 2-byte codepoint of the AGNPRMRM reply message.

#### SVRCOD

A required parameter (X'1149') that contains the severity code. SVRCOD consists of a 2-byte length field (LL), a 2-byte codepoint (CP), and the data. The data is a 2-byte binary value. The following list describes the possible 2-byte values:

0	INFO – Information only severity code
4	WARNING – Warning severity code
8	ERROR – Error severity code
16	SEVERE – Severe error severity code
32	ACCDMG – Access damage severity code
64	PRMDMG – Permanent damage severity code
128	SESDMG – Session damage severity code

#### RDBNAM

Codepoint X'2110'. The IMS PSB name that identifies the target database.

#### SRVDGN

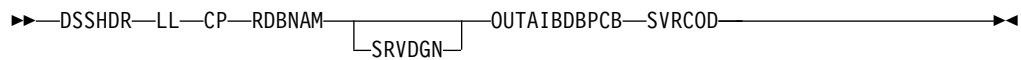
An optional parameter (X'1153') that contains server diagnostic information. The data portion is a string with a maximum length of 32,767 bytes. The string can be any information the server sends to aid in problem diagnosis. For more information about SRVDGN, see *DRDA, Version 4, Volume 3: Distributed Data Management (DDM) Architecture*, which is available from The Open Group at [www.opengroup.org](http://www.opengroup.org).

## CMDVLTRM reply message (X'221D')

The distributed data management (DDM) architecture CMDVLTRM (command violation) reply message indicates that a DDM command violated the processing capabilities of the conversation.



## Format



## Parameters

### DSSHDR

The 6-byte header that contains information about the data stream structure (DSS).

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'221D', the 2-byte codepoint of the CMDVLTRM reply message.

### RDBNAM

Codepoint X'2110'. The IMS PSB name that identifies the target database.

### SRVDGN

An optional parameter (X'1153') that contains server diagnostic information. The data portion is a string with a maximum length of 32,767 bytes. The string can be any information the server sends to aid in problem diagnosis. For more information about SRVDGN, see *DRDA, Version 4, Volume 3: Distributed Data Management (DDM) Architecture*, which is available from The Open Group at [www.opengroup.org](http://www.opengroup.org).

### OUTAIBDBPCB

A required parameter (X'CC02') that contains the AIB and DBPCB data sent from the target to the source.

### SVRCOD

A required parameter (X'1149') that contains the severity code. SVRCOD consists of a 2-byte length field (LL), a 2-byte codepoint (CP), and the data. The data is a 2-byte binary value. The following list describes the possible 2-byte values:

0	INFO – Information only severity code
4	WARNING – Warning severity code
8	ERROR – Error severity code
16	SEVERE – Severe error severity code
32	ACCDMG – Access damage severity code
64	PRMDMG – Permanent damage severity code
128	SESDMG – Session damage severity code

## DEALLOCDBRM reply message (X'CA01')

The distributed data management (DDM) architecture DEALLOCDBRM (deallocate database completed) reply message indicates that the named PSB is deallocated.

## Format



## Parameters

### DSSHDR

The 6-byte header that contains information about the data stream structure (DSS).

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'CA01', the 2-byte codepoint of the DEALLOCDBRM reply message.

### RDBNAM

Codepoint X'2110'. The IMS PSB name that identifies the target database.

### SVRCOD

A required parameter (X'1149') that contains the severity code. SVRCOD consists of a 2-byte length field (LL), a 2-byte codepoint (CP), and the data. The data is a 2-byte binary value. The following list describes the possible 2-byte values:

0	INFO – Information only severity code
4	WARNING – Warning severity code
8	ERROR – Error severity code
16	SEVERE – Severe error severity code
32	ACCDMG – Access damage severity code
64	PRMDMG – Permanent damage severity code
128	SESDMG – Session damage severity code

### SRVDGN

An optional parameter (X'1153') that contains server diagnostic information. The data portion is a string with a maximum length of 32,767 bytes. The string can be any information the server sends to aid in problem diagnosis. For more information about SRVDGN, see *DRDA, Version 4, Volume 3: Distributed Data Management (DDM) Architecture*, which is available from The Open Group at [www.opengroup.org](http://www.opengroup.org).

### OUTAIBDBPCB

A required parameter (X'CC02') that contains the AIB and DBPCB data sent from the target to the source.

#### Related reference:

“DEALLOCDB command (X'C801’)” on page 289

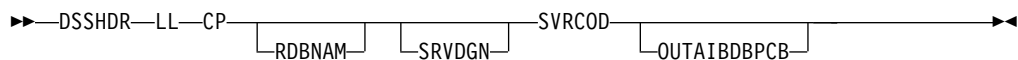
“DEALLOCDBRM reply message (X'CA01’)” on page 331

## ENDQRYRM reply message (X'220B')

The distributed data management (DDM) architecture ENDQRYRM (end of query) reply message indicates that the query processing is terminated and the query or result set is closed.

The query cannot be resumed with the CNTQRY command or closed with the CLSQRY command.

### Format



## Parameters

### DSSHDR

The 6-byte header that contains information about the data stream structure (DSS).

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'220B', the 2-byte codepoint of the ENDQRYRM reply message.

### RDBNAM

Codepoint X'2110'. The IMS PSB name that identifies the target database.

### SRVDGN

An optional parameter (X'1153') that contains server diagnostic information. The data portion is a string with a maximum length of 32,767 bytes. The string can be any information the server sends to aid in problem diagnosis. For more information about SRVDGN, see *DRDA, Version 4, Volume 3: Distributed Data Management (DDM) Architecture*, which is available from The Open Group at [www.opengroup.org](http://www.opengroup.org).

### SVRCOD

A required parameter (X'1149') that contains the severity code. SVRCOD consists of a 2-byte length field (LL), a 2-byte codepoint (CP), and the data. The data is a 2-byte binary value. The following list describes the possible 2-byte values:

0	INFO – Information only severity code
4	WARNING – Warning severity code
8	ERROR – Error severity code
16	SEVERE – Severe error severity code
32	ACCDMG – Access damage severity code
64	PRMDMG – Permanent damage severity code
128	SESDMG – Session damage severity code

### OUTAIBDBPCB

A required parameter (X'CC02') that contains the AIB and DBPCB data sent from the target to the source.

## Usage

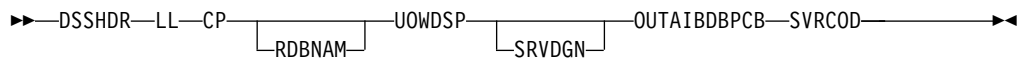
The ENDQRYRM reply message is required to be chained to an OPNQRYRM reply message in the following situations:

- In response to a GU, GN, or batch RETRIEVE call, when no data is returned. The aibdbpcbStream data required by the client is contained in this object.
- In response to a batch RETRIEVE call when the QRYDTA object contains the last row of the data that satisfies the query, which indicates to the source server that ODBM received a GE/GB status code on the last GN call, and no CNTQRY should be sent.

## ENDUOWRM reply message (X'220C')

The distributed data management (DDM) architecture ENDUOWRM (end unit of work) reply message indicates that the unit of work has ended as a result of the last command.

## Format



## Parameters

### DSSHDR

The 6-byte header that contains information about the data stream structure (DSS).

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'220C', the 2-byte codepoint of the ENDUOWRM reply message.

### RDBNAM

Codepoint X'2110'. The IMS PSB name that identifies the target database.

### UOWDSP

A required parameter (X'2115') that specifies the disposition of the last unit of work. If the disposition is committed, all updates in the unit of work are successfully applied. If the disposition is rolled back, all updates in the unit of work are removed.

For more information about UOWDSP, see *DRDA, Version 4, Volume 3: Distributed Data Management (DDM) Architecture*.

### SRVDGN

An optional parameter (X'1153') that contains server diagnostic information. The data portion is a string with a maximum length of 32,767 bytes. The string can be any information the server sends to aid in problem diagnosis. For more information about SRVDGN, see *DRDA, Version 4, Volume 3: Distributed Data Management (DDM) Architecture*, which is available from The Open Group at [www.opengroup.org](http://www.opengroup.org).

### OUTAIBDBPCB

A required parameter (X'CC02') that contains the AIB and DBPCB data sent from the target to the source.

### SVRCOD

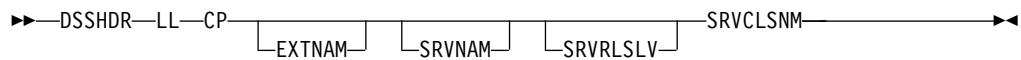
A required parameter (X'1149') that contains the severity code. SVRCOD consists of a 2-byte length field (LL), a 2-byte codepoint (CP), and the data. The data is a 2-byte binary value. The following list describes the possible 2-byte values:

0	INFO – Information only severity code
4	WARNING – Warning severity code
8	ERROR – Error severity code
16	SEVERE – Severe error severity code
32	ACCDMG – Access damage severity code
64	PRMDMG – Permanent damage severity code
128	SESDMG – Session damage severity code

## EXCSATRD reply object (X'1443')

The EXCSATRD reply data object returns information about the IMS target DDM server, such as server name or the product release level, to the source DDM server.

## Format



## Parameters

### DSSHDR

The 6-byte header that contains information about the data stream structure (DSS).

**LL** The 2-byte specification of the length of the EXCSATRD reply object.

**CP** X'1443', the 2-byte codepoint of the EXCSATRD reply object.

### EXTNAM

Optional. The variable-length external name of the target DDM server. For the IMS target DDM server, the external name is the name of the job the IMS system creates or activates to run the DDM server. The EXTNAM parameter is used for tracing and problem determination. If the job name includes embedded blanks, the name must be enclosed in quotation marks within the data field. The maximum length of EXTNAM is 255 bytes. The codepoint is X'115E'.

### SRVNAM

Optional. The variable-length name of the target DDM server specified as a character string. Returned for tracing and problem determination purposes. If the server name includes embedded blanks, the name must be enclosed in quotation marks. The maximum length is 255 bytes. The codepoint is X'116D'.

If the DDM server name includes embedded blanks, the name must be enclosed in quotation marks within the data field.

### SRVRLSLV

Optional. The variable-length name of the product release level of the target DDM server. The SRVRLSLV parameter is used to ensure compatibility between the source server of the application program and the IMS target server. The maximum length of SRVRLSLV is 255 bytes. The codepoint is X'115A'.

### SRVCLSNM

Specifies the DDM server class name, DFS, used by IMS. DFS is currently the only name supported by IMS. The SRVCLSNM enables the DRDA product-unique extension used by IMS.

The codepoint of SRVCLSNM is X'1147'. The variable-length DDM server class name is specified as a character string.

## Usage

The distributed data management (DDM) architecture EXCSATRD reply object is returned by an IMS target server in response to an EXCSAT command. Unless errors occur, the EXCSATRD reply object is always the first reply command from the IMS target DDM server to the source DDM server.

If errors occur during the exchange of server attributes, the IMS target server responds to the EXCSAT command by issuing an error message instead of the EXCSATRD reply object.

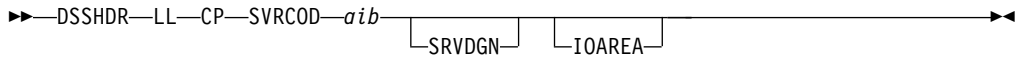
**Related reference:**

“EXCSAT command (X'1041')” on page 292

## IMSCALLRM reply message (X'CA04')

The distributed data management (DDM) architecture IMSCALLRM (IMS call) reply message returns the results of a DL/I call for IMS DB system services submitted by using the IMSCALL command.

### Format



### Parameters

#### DSSHDR

The 6-byte header that contains information about the data stream structure (DSS).

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'CA04', the 2-byte codepoint of the IMSCALLRM reply message.

#### SVRCOD

A required parameter (X'1149') that contains the severity code. SVRCOD consists of a 2-byte length field (LL), a 2-byte codepoint (CP), and the data. The data is a 2-byte binary value. The following list describes the possible 2-byte values:

- |     |   |
|-----|---|
| 0   | INFO – Information only severity code   |
| 4   | WARNING – Warning severity code         |
| 8   | ERROR – Error severity code             |
| 16  | SEVERE – Severe error severity code     |
| 32  | ACCDMG – Access damage severity code    |
| 64  | PRMDMG – Permanent damage severity code |
| 128 | SESDMG – Session damage severity code   |

#### aib

A required parameter and placeholder for exactly one of the following two reply objects:

- OUTAIBDBPCB

A required parameter (X'CC02') that contains the AIB and DBPCB data sent from the target to the source.

- OUTAIBIOPCB

A parameter (X'CC08') that contains the AIB and IOPCB data sent from the target to the source.

#### SRVDGN

An optional parameter (X'1153') that contains server diagnostic information. The data portion is a string with a maximum length of 32,767 bytes. The string can be any information the server sends to aid in problem diagnosis. For more information about SRVDGN, see *DRDA, Version 4, Volume 3: Distributed Data Management (DDM) Architecture*, which is available from The Open Group at [www.opengroup.org](http://www.opengroup.org).

#### IOAREA

An optional parameter in byte array that specifies the input and output area.

#### Related reference:

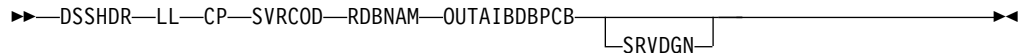
“IMSCALL command (X'C803’)” on page 301

## OPNQFLRM reply message (X'2212')

The distributed data management (DDM) architecture OPNQFLRM (open query failure) reply message indicates that the OPNQRY command failed to open the query.

The reason for the failure is reported in the OUTAIBDBPCB parameter.

### Format



### Parameters

#### DSSHDR

The 6-byte header that contains information about the data stream structure (DSS).

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'2212', the 2-byte codepoint of the OPNQFLRM reply message.

#### SVRCOD

A required parameter (X'1149') that contains the severity code. SVRCOD consists of a 2-byte length field (LL), a 2-byte codepoint (CP), and the data. The data is a 2-byte binary value. The following list describes the possible 2-byte values:

0	INFO – Information only severity code
4	WARNING – Warning severity code
8	ERROR – Error severity code
16	SEVERE – Severe error severity code
32	ACCDMG – Access damage severity code
64	PRMDMG – Permanent damage severity code
128	SESDMG – Session damage severity code

#### RDBNAM

Codepoint X'2110'. The IMS PSB name that identifies the target database.

#### OUTAIBDBPCB

A required parameter (X'CC02') that contains the AIB and DBPCB data sent from the target to the source.

OUTAIBDBPCB contains the reason for the failure.

#### SRVDGN

An optional parameter (X'1153') that contains server diagnostic information. The data portion is a string with a maximum length of 32,767 bytes. The string can be any information the server sends to aid in problem diagnosis. For more information about SRVDGN, see *DRDA, Version 4, Volume 3: Distributed Data Management (DDM) Architecture*, which is available from The Open Group at [www.opengroup.org](http://www.opengroup.org).

#### Related reference:

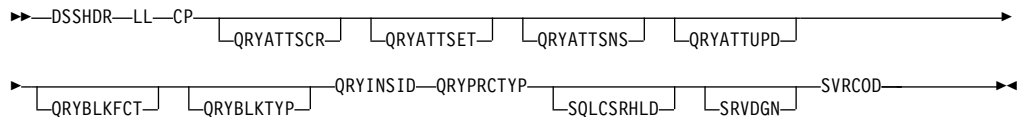
“OPNQRY command (X'200C’)” on page 304

## OPNQRYRM reply message (X'2205')

The distributed data management (DDM) architecture OPNQRYRM (open query) reply message indicates that the open query (OPNQRY) command or execute SQL statement (EXCSQLSTT) command completed normally, and that a query process has been initiated.

This reply message also indicates the type of query protocol and cursor that are used for the query.

### Format



### Parameters

#### DSSHDR

The 6-byte header that contains information about the data stream structure (DSS).

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'2205', the 2-byte codepoint of the OPNQRYRM reply message.

#### QRYATTSCR

An optional parameter that specifies the query attribute for scrollability. The codepoint is X'2149'.

#### QRYATTSET

An optional parameter that indicates whether a cursor is enabled for the a single row or multiple rows to be returned. The codepoint is X'214A'.

#### QRYATTSNS

An optional parameter that specifies the sensitivity of an opened cursor to changes made to the underlying base table. The codepoint is X'2157'.

#### QRYATTUPD

An optional parameter that indicates the updatability of an opened cursor. The codepoint is X'2150'.

#### QRYBLKFCT

An optional parameter that contains the value of the blocking factor, a limit that is imposed by the target server that dictates the number of rows that can be blocked at a time for a query. The codepoint is X'215F'.

#### QRYBLKTYP

An optional parameter that indicates the type of query blocks in which the answer set data will be returned. The codepoint is X'2133'.

#### QRYINSID

A required parameter that uniquely identifies the instance of a query. The codepoint is X'215B'.

This parameter is returned on the OPNQRYRM reply message by the target server when a query is opened. Any subsequent references to this query by the target server must include the QRYINSID value in order to identify the correct instance of the query.



### QRYPRCTYP

A required String parameter that specifies the type of query protocol that is used. The codepoint is X'2102'.

### SQLCSRHLD

An optional Boolean parameter that indicates whether the requester specified the hold cursor position option. The codepoint is X'211F'.

### SRVDGN

An optional parameter (X'1153') that contains server diagnostic information. The data portion is a string with a maximum length of 32,767 bytes. The string can be any information the server sends to aid in problem diagnosis. For more information about SRVDGN, see *DRDA, Version 4, Volume 3: Distributed Data Management (DDM) Architecture*, which is available from The Open Group at [www.opengroup.org](http://www.opengroup.org).

### SVRCOD

A required parameter (X'1149') that contains the severity code. SVRCOD consists of a 2-byte length field (LL), a 2-byte codepoint (CP), and the data. The data is a 2-byte binary value. The following list describes the possible 2-byte values:

0	INFO – Information only severity code
4	WARNING – Warning severity code
8	ERROR – Error severity code
16	SEVERE – Severe error severity code
32	ACCDMG – Access damage severity code
64	PRMDMG – Permanent damage severity code
128	SESDMG – Session damage severity code

For more information on these parameters, see *DRDA, Version 4, Volume 3: Distributed Data Management (DDM) Architecture*, by the Open Group.

## Reply data objects

The following reply data objects can be chained to the OPNQRYRM message in response to the OPNQRY command:

### QRYDSC (X'241A')

Query answer set description.

### QRYDTA (X'241B')

Query answer set data.

#### Related reference:

“OPNQRY command (X'200C’)” on page 304

## QRYDSC reply object (X'241A')

The distributed data management (DDM) architecture QRYDSC (query answer set description) reply object defines the format of the data that is returned in a QRYDTA object.

### Format

►►—DSSHDR—LL—CP—BYTSTRDR—————►►

## Parameters

### DSSHDR

The 6-byte header that contains information about the data stream structure (DSS).

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'241A', the 2-byte codepoint of the QRYDSC reply object.

### BYTSTRDR

A required byte string data representation. This byte string contains the FD:OCA description of the data that the QRYDTA object sends.

## Usage

The format for the data that the QRYDTA object returns never changes. The BYTSTRDR value for the ODBM is always as follows:

```
0676D0260000 0671E0D000010671 F0E00000
```

## QRYDTA reply object (X'241B')

The distributed data management (DDM) architecture QRYDTA (query answer set data) reply object contains some or all of the answer set data that results from a query.

## Format

►►—DSSHDR—LL—CP—*aibdbpcbStream—data*—————►►

## Parameters

### DSSHDR

The 6-byte header that contains information about the data stream structure (DSS).

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'241B', the 2-byte codepoint of the QRYDTA reply object.

### **aibStream**

An IMS-defined data structure.

When the Open Database Manager (ODBM) services a GU or GN call, it places the aibStream data stream followed by the dbpcbStream at the beginning of each row in the QRYDTA object. The requested row data follows immediately. The concatenation of the aibdbpcbStream and data fields comprises a single row in a query row set.

### **dbpcbStream**

An IMS-defined data structure.

When the Open Database Manager (ODBM) services a GU or GN call, it places the aibStream data stream followed by the dbpcbStream at the beginning of each row in the QRYDTA object. The requested row data follows immediately. The concatenation of the aibdbpcbStream and data fields makes up a single row in a query row set.

### **data**

The data that follows the aibStream and dbpcbStream data structures.

## Usage

The contents of the QRYDTA reply object are described by the QRYDSC reply object. Because IMS sends all of the data for a given row as if it were a single column of type "Fixed Length Byte Stream," the QRYDSC information is the same for each query. Each row is made up of an aibdbpcbStream object followed by the data.

### Related reference:

"aibStream data structure" on page 352

"dbpcbStream data structure" on page 353

## QRYPOPRM reply message (X'220F')

The distributed data management (DDM) architecture QRYPOPRM (query previously opened) reply message is returned when a command is issued for a query that is already open.

### Format



### Parameters

#### DSSHDR

The 6-byte header that contains information about the data stream structure (DSS).

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'220F', the 2-byte codepoint of the QRYPOPRM reply message.

#### PCBNAME

A required parameter that specifies the PCB name that uniquely identifies the query made by a DL/I call. The PCB name is specified as a character string. The value is initially sent with the original OPNQRY command. The same value must subsequently be sent in commands such as CNTQRY, CLSQRY, and RLSE for proper correlation with the original OPNQRY call. The codepoint for the PCBNAME parameter is X'C907'.

#### RDBNAM

Codepoint X'2110'. The IMS PSB name that identifies the target database.

#### SRVDGN

An optional parameter (X'1153') that contains server diagnostic information. The data portion is a string with a maximum length of 32,767 bytes. The string can be any information the server sends to aid in problem diagnosis. For more information about SRVDGN, see *DRDA, Version 4, Volume 3: Distributed Data Management (DDM) Architecture*, which is available from The Open Group at [www.opengroup.org](http://www.opengroup.org).

#### SVRCOD

A required parameter (X'1149') that contains the severity code. SVRCOD consists of a 2-byte length field (LL), a 2-byte codepoint (CP), and the data. The data is a 2-byte binary value. The following list describes the possible 2-byte values:

0	INFO – Information only severity code
4	WARNING – Warning severity code
8	ERROR – Error severity code
16	SEVERE – Severe error severity code
32	ACCDMG – Access damage severity code
64	PRMDMG – Permanent damage severity code
128	SESDMG – Session damage severity code

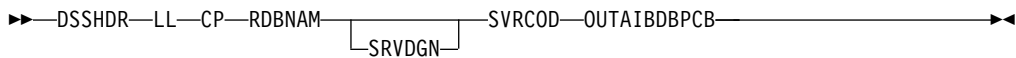
**Related reference:**

“OPNQRY command (X'200C')” on page 304

## RDBAFLRM reply message (X'221A')

The distributed data management (DDM) architecture RDBAFLRM (database access failed) reply message indicates that the database access failed.

### Format



### Parameters

#### DSSHDR

The 6-byte header that contains information about the data stream structure (DSS).

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'221A', the 2-byte codepoint of the RDBAFLRM reply message.

#### RDBNAM

Codepoint X'2110'. The IMS PSB name that identifies the target database.

#### SRVDGN

An optional parameter (X'1153') that contains server diagnostic information. The data portion is a string with a maximum length of 32,767 bytes. The string can be any information the server sends to aid in problem diagnosis. For more information about SRVDGN, see *DRDA, Version 4, Volume 3: Distributed Data Management (DDM) Architecture*, which is available from The Open Group at [www.opengroup.org](http://www.opengroup.org).

#### SVRCOD

A required parameter (X'1149') that contains the severity code. SVRCOD consists of a 2-byte length field (LL), a 2-byte codepoint (CP), and the data. The data is a 2-byte binary value. The following list describes the possible 2-byte values:

0	INFO – Information only severity code
4	WARNING – Warning severity code
8	ERROR – Error severity code
16	SEVERE – Severe error severity code
32	ACCDMG – Access damage severity code
64	PRMDMG – Permanent damage severity code
128	SESDMG – Session damage severity code

#### OUTAIBDBPCB

A required parameter (X'CC02') that contains the AIB and DBPCB data sent from the target to the source.

OUTAIBDBPCB contains the reason for the failure.

## Usage

The RDBAFLRM reply message is returned only when a RDBNAM value is provided and the connection to the database failed. When the RDBAFLRM reply message is returned, the target Structured Query Language Application Manager (SQLAM) instance is destroyed. For more information about the SQLAM instance, see *DRDA, Version 4, Volume 3: Distributed Data Management (DDM) Architecture* by the Open Group.

### Related reference:

“ACCRDB command (X'2001’)” on page 283

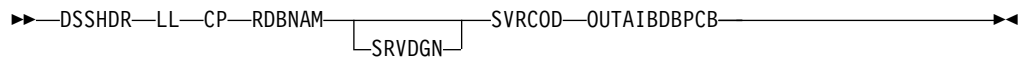
“RDBNAM parameter (X'2110’)” on page 357

“OUTAIBDBPCB parameter (X'CC02’)” on page 355

## RDBATHRM reply message (X'2203')

The distributed data management (DDM) architecture RDBATHRM (not authorized to database) reply message indicates that the user is not authorized to access the database.

### Format



### Parameters

#### DSSHDR

The 6-byte header that contains information about the data stream structure (DSS).

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'2203', the 2-byte codepoint of the RDBATHRM reply message.

#### RDBNAM

Codepoint X'2110'. The IMS PSB name that identifies the target database.

#### SRVDGN

An optional parameter (X'1153') that contains server diagnostic information. The data portion is a string with a maximum length of 32,767 bytes. The string can be any information the server sends to aid in problem diagnosis. For more information about SRVDGN, see *DRDA, Version 4, Volume 3: Distributed Data Management (DDM) Architecture*, which is available from The Open Group at [www.opengroup.org](http://www.opengroup.org).

#### SVRCOD

A required parameter (X'1149') that contains the severity code. SVRCOD consists of a 2-byte length field (LL), a 2-byte codepoint (CP), and the data. The data is a 2-byte binary value. The following list describes the possible 2-byte values:

**0** INFO – Information only severity code

- 4      WARNING – Warning severity code
- 8      ERROR – Error severity code
- 16     SEVERE – Severe error severity code
- 32     ACCDMG – Access damage severity code
- 64     PRMDMG – Permanent damage severity code
- 128    SESDMG – Session damage severity code

#### OUTAIBDBPCB

A required parameter (X'CC02') that contains the AIB and DBPCB data sent from the target to the source.

OUTAIBDBPCB contains the reason for the failure.

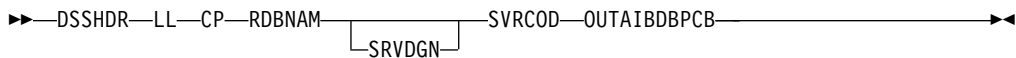
#### Related reference:

“ACCRDB command (X'2001’)” on page 283

## RDBNACRM reply message (X'2204')

The distributed data management (DDM) architecture RDBNACRM (database not accessed) reply message indicates that the access database command (ACCRDB) was not issued prior to a command that requested the database services.

### Format



### Parameters

#### DSSHDR

The 6-byte header that contains information about the data stream structure (DSS).

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'2204', the 2-byte codepoint of the RDBNACRM reply message.

#### RDBNAM

Codepoint X'2110'. The IMS PSB name that identifies the target database.

#### SRVDGN

An optional parameter (X'1153') that contains server diagnostic information. The data portion is a string with a maximum length of 32,767 bytes. The string can be any information the server sends to aid in problem diagnosis. For more information about SRVDGN, see *DRDA, Version 4, Volume 3: Distributed Data Management (DDM) Architecture*, which is available from The Open Group at [www.opengroup.org](http://www.opengroup.org).

#### SVRCOD

A required parameter (X'1149') that contains the severity code. SVRCOD consists of a 2-byte length field (LL), a 2-byte codepoint (CP), and the data. The data is a 2-byte binary value. The following list describes the possible 2-byte values:

- 0      INFO – Information only severity code
- 4      WARNING – Warning severity code
- 8      ERROR – Error severity code
- 16     SEVERE – Severe error severity code
- 32     ACCDMG – Access damage severity code

- 64 PRMDMG – Permanent damage severity code
- 128 SESDMG – Session damage severity code

**OUTAIBDBPCB**

A required parameter (X'CC02') that contains the AIB and DBPCB data sent from the target to the source.

OUTAIBDBPCB contains the reason for the failure.

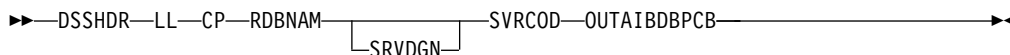
**Related reference:**

“ACCRDB command (X'2001’)” on page 283

## RDBNFNRM reply message (X'2211')

The distributed data management (DDM) architecture RDBNFNRM (database not found) reply message indicates that the requested database was not found.

**Format**



**Parameters**

**DSSHDR**

The 6-byte header that contains information about the data stream structure (DSS).

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'2211', the 2-byte codepoint of the RDBNFNRM reply message.

**RDBNAM**

Codepoint X'2110'. The IMS PSB name that identifies the target database.

**SRVDGN**

An optional parameter (X'1153') that contains server diagnostic information. The data portion is a string with a maximum length of 32,767 bytes. The string can be any information the server sends to aid in problem diagnosis. For more information about SRVDGN, see *DRDA, Version 4, Volume 3: Distributed Data Management (DDM) Architecture*, which is available from The Open Group at [www.opengroup.org](http://www.opengroup.org).

**SVRCOD**

A required parameter (X'1149') that contains the severity code. SVRCOD consists of a 2-byte length field (LL), a 2-byte codepoint (CP), and the data. The data is a 2-byte binary value. The following list describes the possible 2-byte values:

- 0 INFO – Information only severity code
- 4 WARNING – Warning severity code
- 8 ERROR – Error severity code
- 16 SEVERE – Severe error severity code
- 32 ACCDMG – Access damage severity code
- 64 PRMDMG – Permanent damage severity code
- 128 SESDMG – Session damage severity code

**OUTAIBDBPCB**

A required parameter (X'CC02') that contains the AIB and DBPCB data sent from the target to the source.

OUTAIBDBPCB contains the reason for the failure.

## RDBUPDRM reply message (X'2218')

The distributed data management (DDM) architecture RDBUPDRM (database update) reply message indicates that the a DDM command resulted in an update at the target database.

### Format



### Parameters

#### DSSHDR

The 6-byte header that contains information about the data stream structure (DSS).

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'2218', the 2-byte codepoint of the RDBUPDRM reply message.

#### RDBNAM

Codepoint X'2110'. The IMS PSB name that identifies the target database.

#### SVRCOD

A required parameter (X'1149') that contains the severity code. SVRCOD consists of a 2-byte length field (LL), a 2-byte codepoint (CP), and the data. The data is a 2-byte binary value. The following list describes the possible 2-byte values:

- 0 INFO – Information only severity code
- 4 WARNING – Warning severity code
- 8 ERROR – Error severity code
- 16 SEVERE – Severe error severity code
- 32 ACCDMG – Access damage severity code
- 64 PRMDMG – Permanent damage severity code
- 128 SESDMG – Session damage severity code

#### UPDCNT

A required 4-byte integer parameter (X'C90A') that contains the number of rows affected by a batch or singleton INSERT, UPDATE, or DELETE.

#### SRVDGN

An optional parameter (X'1153') that contains server diagnostic information. The data portion is a string with a maximum length of 32,767 bytes. The string can be any information the server sends to aid in problem diagnosis. For more information about SRVDGN, see *DRDA, Version 4, Volume 3: Distributed Data Management (DDM) Architecture*, which is available from The Open Group at [www.opengroup.org](http://www.opengroup.org).

#### OUTAIBDBPCB

A required parameter (X'CC02') that contains the AIB and DBPCB data sent from the target to the source.



OUTAIBDBPCB contains the details.

**Related reference:**

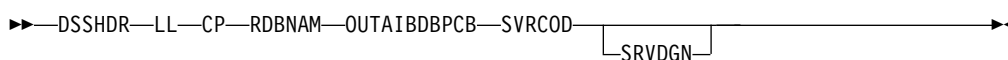
“EXCSQLIMM command (X'200A')” on page 294

“UPDCNT parameter (X'C90A')” on page 358

## RLSERM reply message (X'CA03')

The distributed data management (DDM) architecture RLSERM (release locks) reply message indicates to the requester that an RLSE command has completed normally.

### Format



### Parameters

#### DSSHDR

The 6-byte header that contains information about the data stream structure (DSS).

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'CA03', the 2-byte codepoint of the RLSERM reply message.

#### RDBNAM

Codepoint X'2110'. The IMS PSB name that identifies the target database.

#### OUTAIBDBPCB

A required parameter (X'CC02') that contains the AIB and DBPCB data sent from the target to the source.

#### SVRCOD

A required parameter (X'1149') that contains the severity code. SVRCOD consists of a 2-byte length field (LL), a 2-byte codepoint (CP), and the data. The data is a 2-byte binary value. The following list describes the possible 2-byte values:

- 0 INFO – Information only severity code
- 4 WARNING – Warning severity code
- 8 ERROR – Error severity code
- 16 SEVERE – Severe error severity code
- 32 ACCDMG – Access damage severity code
- 64 PRMDMG – Permanent damage severity code
- 128 SESDMG – Session damage severity code

#### SRVDGN

An optional parameter (X'1153') that contains server diagnostic information. The data portion is a string with a maximum length of 32,767 bytes. The string can be any information the server sends to aid in problem diagnosis. For more information about SRVDGN, see *DRDA, Version 4, Volume 3: Distributed Data Management (DDM) Architecture*, which is available from The Open Group at [www.opengroup.org](http://www.opengroup.org).

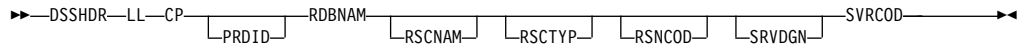
**Related reference:**

“RLSE command (X'C802')” on page 310

## RSCLMTRM reply message (X'1233')

The distributed data management (DDM) architecture RSCLMTRM (resource limits reached) reply message indicates that the requested command cannot be completed because of insufficient target server resources.

### Format



### Parameters

#### DSSHDR

The 6-byte header that contains information about the data stream structure (DSS).

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'1233', the 2-byte codepoint of the RSCLMTRM reply message.

#### PRDID

An optional parameter that specifies the release level of the source DDM server. The codepoint for PRDID is X'112E'.

#### RDBNAM

Codepoint X'2110'. The IMS PSB name that identifies the target database.

#### RSCNAM

An optional String parameter that specifies the name of the resource that reached its limit and sends this RSCLMTRM reply message in response. The codepoint for RSCNAM is X'112D'.

#### RSCTYP

An optional String parameter that specifies the type of the resource that reached its limit and sends this RSCLMTRM reply message. The codepoint for RSCTYP is X'111F'.

#### RSNCOD

An optional String parameter that specifies the reason code. The codepoint for RSNCOD is X'1127'.

#### SRVDGN

An optional parameter (X'1153') that contains server diagnostic information. The data portion is a string with a maximum length of 32,767 bytes. The string can be any information the server sends to aid in problem diagnosis. For more information about SRVDGN, see *DRDA, Version 4, Volume 3: Distributed Data Management (DDM) Architecture*, which is available from The Open Group at [www.opengroup.org](http://www.opengroup.org).

#### SVRCOD

A required parameter (X'1149') that contains the severity code. SVRCOD consists of a 2-byte length field (LL), a 2-byte codepoint (CP), and the data. The data is a 2-byte binary value. The following list describes the possible 2-byte values:

- 0** INFO – Information only severity code
- 4** WARNING – Warning severity code
- 8** ERROR – Error severity code
- 16** SEVERE – Severe error severity code

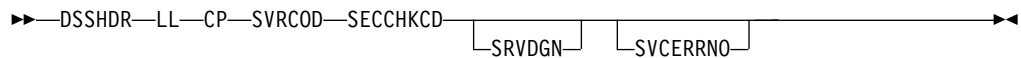
- 32 ACCDMG – Access damage severity code
- 64 PRMDMG – Permanent damage severity code
- 128 SESDMG – Session damage severity code

## SECCHKRM reply message (X'1219')

The distributed data management (DDM) architecture SECCHKRM (security check) reply message indicates the acceptability of the security information.

The security manager specifies the state of the security information by using the security check code (SECCHKCD) parameter.

### Format



### Parameters

#### DSSHDR

The 6-byte header that contains information about the data stream structure (DSS).

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'1219', the 2-byte codepoint of the SECCHKRM reply message.

#### SVRCOD

A required parameter (X'1149') that contains the severity code. SVRCOD consists of a 2-byte length field (LL), a 2-byte codepoint (CP), and the data. The data is a 2-byte binary value. The following list describes the possible 2-byte values:

- 0 INFO – Information only severity code
- 4 WARNING – Warning severity code
- 8 ERROR – Error severity code
- 16 SEVERE – Severe error severity code
- 32 ACCDMG – Access damage severity code
- 64 PRMDMG – Permanent damage severity code
- 128 SESDMG – Session damage severity code

#### SECCHKCD

A required String parameter that indicates the security information and condition for the SECCHKRM reply message. For more information about the possible code values and the relationship of SECCHKCD and SVRCOD in the SECCHKRM reply message, see *DRDA, Version 4, Volume 3: Distributed Data Management (DDM) Architecture* by the Open Group.

#### SRVDGN

An optional parameter (X'1153') that contains server diagnostic information. The data portion is a string with a maximum length of 32,767 bytes. The string can be any information the server sends to aid in problem diagnosis. For more information about SRVDGN, see *DRDA, Version 4, Volume 3: Distributed Data Management (DDM) Architecture*, which is available from The Open Group at [www.opengroup.org](http://www.opengroup.org).

#### SVCERRNO

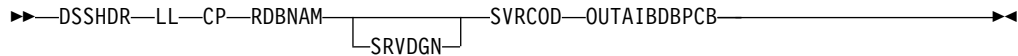
An optional parameter (X'11B4') that contains a security service error number from the local security service. SRVDGN might contain additional information.

SVCERRNO consists of a 2-byte length field (LL), a 2-byte codepoint, followed by a 4-byte binary number data.

## SQLERRRM reply message (X'2213')

The distributed data management (DDM) architecture SQLERRRM (SQL error condition) reply message indicates that an SQL error has occurred.

### Format



### Parameters

#### DSSHDR

The 6-byte header that contains information about the data stream structure (DSS).

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'2213', the 2-byte codepoint of the SQLERRRM reply message.

#### RDBNAM

Codepoint X'2110'. The IMS PSB name that identifies the target database.

#### SRVDGN

An optional parameter (X'1153') that contains server diagnostic information. The data portion is a string with a maximum length of 32,767 bytes. The string can be any information the server sends to aid in problem diagnosis. For more information about SRVDGN, see *DRDA, Version 4, Volume 3: Distributed Data Management (DDM) Architecture*, which is available from The Open Group at [www.opengroup.org](http://www.opengroup.org).

#### SVRCOD

A required parameter (X'1149') that contains the severity code. SVRCOD consists of a 2-byte length field (LL), a 2-byte codepoint (CP), and the data. The data is a 2-byte binary value. The following list describes the possible 2-byte values:

0	INFO – Information only severity code
4	WARNING – Warning severity code
8	ERROR – Error severity code
16	SEVERE – Severe error severity code
32	ACCDMG – Access damage severity code
64	PRMDMG – Permanent damage severity code
128	SESDMG – Session damage severity code

#### OUTAIBDBPCB

A required parameter (X'CC02') that contains the AIB and DBPCB data sent from the target to the source.

OUTAIBDBPCB contains the reason for the error.

---

## DDM parameters used by IMS

In some of the DDM terms used by IMS, IMS defines product-unique parameter values and data structures.

## AIBOALEN parameter (X'C904')

The AIBOALEN parameter is an IMS product-unique distributed data management (DDM) architecture parameter that identifies the maximum output length on all calls that return data.

### Format

▶▶—LL—CP—AIBOALEN—————▶▶

### Parameters

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'C904', the 2-byte codepoint of the AIBOALEN parameter.

#### AIBOALEN

A 4-byte integer containing the maximum output length.

#### Related reference:

“INAIB command object (X'CC01')” on page 302

## AIBRSNM1 parameter (X'C901')

The AIBRSNM1 parameter is an IMS product-unique distributed data management (DDM) architecture parameter that contains the PCB name.

### Format

▶▶—LL—CP—AIBRSNM1—————▶▶

### Parameters

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'C901', the 2-byte codepoint of the AIBRSNM1 parameter.

#### AIBRSNM1

A left-justified, 1- to 8-byte string that contains the PCB name.

#### Related reference:

“INAIB command object (X'CC01')” on page 302

## AIBRSNM2 parameter (X'C902')

The AIBRSNM2 parameter is an IMS product-unique distributed data management (DDM) architecture parameter that contains the identifier of the ODBA startup table.

### Format

▶▶—LL—CP—AIBRSNM2—————▶▶

### Parameters

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'C902', the 2-byte codepoint of the AIBRSNM2 parameter.

**AIBRSNM2**

A 4-byte string that specifies the 4-character identifier of the ODBA startup table. For example, in DFSxxxx0, xxxx is the 4-character identifier.

**Related reference:**

“INAIB command object (X'CC01')” on page 302

**AIBSFUNC parameter (X'C903')**

The AIBSFUNC parameter is an IMS product-unique distributed data management (DDM) architecture parameter that contains the sub-function code, if any, of a DL/I call.

**Format**

▶—LL—CP—AIBSFUNC—▶

**Parameters**

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'C903', the 2-byte codepoint of the AIBSFUNC parameter.

**AIBSFUNC**

A left-justified, 1- to 8-byte string that contains the sub-function code of DL/I call.

**Related reference:**

“INAIB command object (X'CC01')” on page 302

**aibStream data structure**

The distributed data management (DDM) architecture aibStream is a data structure that is contained in the OUTAIBDBPCB DDM object when the object is passed back in a reply message.

**Format**

Table 85. Format of the aibStream data structure

Byte offset	Length	Name	Description
0	1	AIB null indicator	Binary integer.  X'00' Indicates that the rest of the aibStream data structure is present.  X'FF' Indicates that the aibStream data structure contains no data after the AIB null indicator. The total length of the aibStream data structure is one byte.
1	4	Output area used	Binary integer.
5	4	Return code	Binary integer.
9	4	Reason code	Binary integer.
13	4	Error code extension	Binary integer.

## Usage

When the Open Database Manager (ODBM) services a GU or GN call, ODBM returns requested data in rows defined within the data stream of a QRYDTA reply object. Each row begins with a concatenation of the aibStream and the dbpcbStream data structures followed by the requested data.

### Related reference:

“QRYDTA reply object (X'241B’)” on page 340

“dbpcbStream data structure”

“OUTAIBDBPCB parameter (X'CC02’)” on page 355

“OUTAIBIOPCB parameter (X'CC08’)” on page 356

## dbpcbStream data structure

The distributed data management (DDM) architecture dbpcbStream is a data structure that is contained in the OUTAIBDBPCB reply object when the object is passed back in a reply message.

### Format

The following table defines the format of the dbpcbStream data structure.

**Note:** The table shows some fields as having two possible starting byte offsets. For each these fields, if a database name is included in the dbpcbStream data structure, the field starts at the higher of the possible byte offsets.

Table 86. Format of the dbpcbStream data structure

Byte offset	Length	Name	Description
0	1	DBPCB null indicator	Binary integer. <b>X'00'</b> Indicates that the rest of the dbpcbStream data structure is present. <b>X'FF'</b> Indicates that the dbpcbStream data structure contains no data after the DBPCB null indicator. The total length of the dbpcbStream data structure is one byte.
1	1	Database name null indicator	Binary integer. <b>X'00'</b> Indicates that a database name field is present at offset 2. <b>X'FF'</b> Indicates that a database name field is not present.
2	8	Database name	Character string.
2 or 10	2	Segment level number	Right-justified, numeric character data.
4 or 12	2	Status code	Character data.
6 or 14	8	Segment name	Character string.

Table 86. Format of the dbpcbStream data structure (continued)

Byte offset	Length	Name	Description
14 or 22	1	Key feedback null indicator	Binary integer.  X'00' Indicates that a key feedback fields begin at offset 15 or, if a database name is present at offset 2, at offset 23.  X'FF' Indicates that no key feedback fields are present.
15 or 23	4	Key feedback length	Binary integer.
19 or 27	Variable	Key feedback area	Variable-length character string. The length of the key feedback area is defined in the key feedback length field at offset 15 or, if a database name is present in the dbpcbStream data structure, at offset 23.

## Usage

When the Open Database Manager (ODBM) services a GU or GN call, ODBM returns requested data in rows defined within the data stream of a QRYDTA reply object. Each row begins with a concatenation of the aibStream and the dbpcbStream data structures followed by the requested data.

### Related reference:

“aibStream data structure” on page 352

“QRYDTA reply object (X'241B')” on page 340

“OUTAIBDBPCB parameter (X'CC02')” on page 355

## iopcbStream data structure

The distributed data management (DDM) architecture iopcbStream is a data structure that is contained in the OUTAIBIOPCB DDM object when the object is passed back in a reply message.

### Format

Table 87. Format of the iopcbStream data structure

Byte offset	Length	Name	Description
0	1	IOPCB null indicator	Binary integer.  X'00' Indicates that the iopcbStream data structure is present.  X'FF' Indicates that the iopcbStream data structure contains no data after the IOPCB null indicator. The total length of the iopcbStream data structure is 1 byte.
1	1	LTERM name null indicator	Binary integer.  X'00' Indicates that the LTERM name field is present at offset 2 and that the field contains a logical terminal name.  X'FF' Indicates that the LTERM name field is not present.



Table 87. Format of the *iopcbStream* data structure (continued)

Byte offset	Length	Name	Description
2	8	LTERM name	Character string.
2 or 10	2	Reserved	Reserved.
4 or 12	2	Status code	Character data.
6 or 14	4	Local data and time	Byte array.
10 or 18	4	Input message sequence number	Byte array.
14 or 22	8	Message output descriptor name	Character string.
22 or 30	8	User ID	Character string.
30 or 38	8	Group name	Character string.
38 or 46	12	Timestamp	Byte Array.
50 or 58	1	User ID indicator	Character data.

## Usage

The figure and description in the preceding sections serve as the FD:OCA early descriptor definition of the *iopcbStream* data structure. Do not confuse the *iopcbStream* data structure with the OUTAIBIOPCB reply object (X'CC08') that contains it when *iopcbStream* is passed back in a reply message.

### Related reference:

“OUTAIBIOPCB parameter (X'CC08')” on page 356

## OUTAIBDBPCB parameter (X'CC02')

The distributed data management (DDM) architecture OUTAIBDBPCB (output AIBDBPCB) parameter is sent from the target server to the source server and contains a concatenation of the *aibStream* and the *dbpcbStream* data structures.

### Format

►►—LL—CP—*aibStream*—*dbpcbStream*—◄◄

### Parameters

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'CC02', the 2-byte codepoint of the OUTAIBDBPCB parameter.

#### **aibStream**

Required. Contains the following data:

- AIB null indicator
- Output area
- Return code
- Reason code
- Error code extension

#### **dbpcbStream**

Required. Contains the following data:

- DBPCB null indicator

- Database name
- Segment level number
- Status code
- Key feedback

## Usage

This OUTAIBDBPCB parameter is a scalar parameter. No length or codepoint values are placed in front of the aibStream or dbpcbStream data structures.

### Related reference:

“DEALLOCDBRM reply message (X'CA01)’” on page 331

“aibStream data structure” on page 352

“dbpcbStream data structure” on page 353

“RDBAFLRM reply message (X'221A)’” on page 342

## OUTAIBIOPCB parameter (X'CC08')

The distributed data management (DDM) architecture OUTAIBIOPCB (output AIBIOPCB) parameter is sent from the target server to the source server and contains a concatenation of the aibStream and the iopcbStream data structures.

### Format

▶▶—LL—CP—aibStream—iopcbStream————▶▶

### Parameters

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'CC08', the 2-byte codepoint of the OUTAIBIOPCB parameter.

#### aibStream

Required. Contains the following data:

- AIB null indicator
- Output area
- Return code
- Reason code
- Error code extension

#### iopcbStream

Required. Contains the following data:

- IOPCB null indicator
- LTERM name
- Status code
- Input message segment number
- Message output descriptor name
- Group name
- Key feedback

## Usage

The OUTAIBIOPCB parameter is a scalar parameter. No length or codepoint values are placed in front of the aibStream or iopcbStream data structures.

### Related reference:

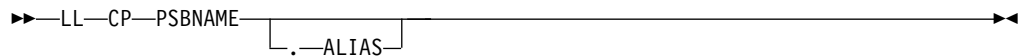
“iopcbStream data structure” on page 354

“aibStream data structure” on page 352

## RDBNAM parameter (X'2110')

The distributed data management (DDM) architecture RDBNAM parameter identifies the target database for a given interaction.

### Format



### Parameters

**LL** The 2-byte specification of the length of the RDBNAM parameter.

**CP** X'2110', the 2-byte codepoint of the RDBNAM parameter.

#### PSBNAME

The IMS PSB name, specified as a character string up to 8 bytes long. The PSB name identifies the target database and must match a PSB name defined to IMS.

#### ALIAS

Optional. The alias name of the IMS data store name. ALIAS must be specified as 4 bytes. If the alias name is fewer than 4 characters, the characters must be left aligned and the remaining bytes must be padded with blank character spaces.

When ALIAS is used, the PSBNAME and the ALIAS must be separated by a period.

### Related reference:

“ACCRDB command (X'2001')” on page 283

“ACCSEC command (X'106D')” on page 285

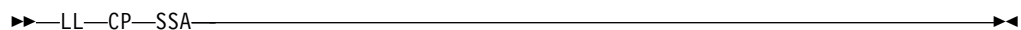
“DEALLOCDB command (X'C801')” on page 289

“RDBAFLRM reply message (X'221A')” on page 342

## SSA parameter (X'C906')

The SSA parameter is an IMS product-unique distributed data management (DDM) architecture parameter that contains a segment search argument (SSA) that qualifies a DL/I call.

### Format



### Parameters

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'C906', the 2-byte codepoint of the SSA parameter.

**SSA**

A byte-string that contains an SSA.

**Related reference:**

"SSALIST command object (X'CC06')" on page 325

## SSACOUNT parameter (X'C905')

The SSACOUNT parameter is an IMS product-unique distributed data management (DDM) architecture parameter that specifies the number of segment search arguments (SSAs) included in the SSALIST command object.

### Format

▶▶—LL—CP—SSACOUNT—————▶▶

### Parameters

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'C905', the 2-byte codepoint of the SSACOUNT parameter.

**SSACOUNT**

A 2-byte binary number field that indicates the number of SSAs included in the SSALIST command object. The minimum value is 1 and the maximum value is 15.

**Related reference:**

"SSALIST command object (X'CC06')" on page 325

## UPDCNT parameter (X'C90A')

The UPDCNT parameter is an IMS product-unique distributed data management (DDM) architecture parameter that identifies the number of rows affected by an individual or batch INSERT, UPDATE, or DELETE action.

### Format

▶▶—LL—CP—UPDCNT—————▶▶

### Parameters

**LL** The length specified as a 2-byte binary integer. This length includes LL and CP.

**CP** X'C90A', the 2-byte codepoint of the UPDCNT parameter.

**UPDCNT**

A 4-byte integer containing the number of rows affected by a batch or singleton INSERT, UPDATE, or DELETE action.

**Related reference:**

"RDBUPDRM reply message (X'2218')" on page 346

---

## Chapter 3. IMS Adapter for REXX reference

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 DFSDDL0 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 *z/OS TSO/E 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 *z/OS TSO/E 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.

The advantages of REXX are:

- 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 these 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 WT0, WTP, WTL, and WT0R

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 bound 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.
- The PSB must be defined as assembler language or COBOL.

- 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.

**Related reading:** For more information on SYSTSPRT and SYSTSIN, see *z/OS TSO/E REXX Reference*.

**Related reference:**  
 “IVPREXX sample application” on page 391

## IMS Adapter for REXX overview

The following figure 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.

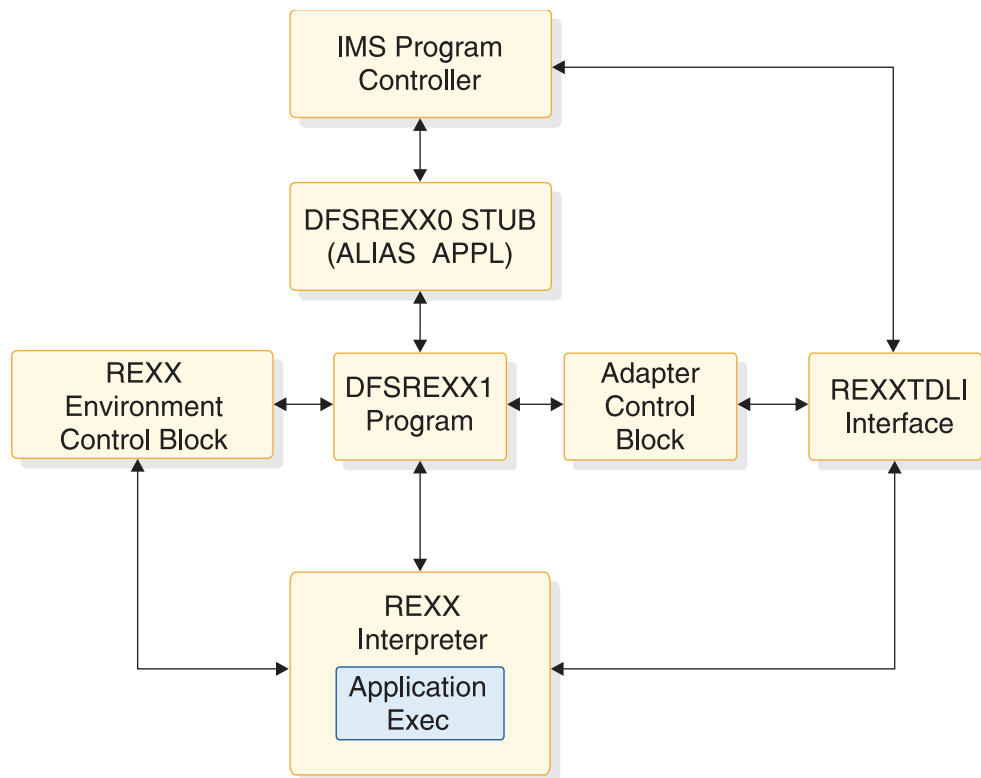


Figure 9. IMS Adapter for REXX logical overview diagram

**Related reference:**

[➡](#) IMS Adapter for REXX exit routine (DFSREXXU) (Exit Routines)

---

## Sample exit routine (DFSREXXU)

IMS provides a sample user exit routine that is used with the IMS Adapter for REXX.

For a description of how to write the user exit routine see *IMS Version 13 Exit Routines*. The sample user exit routine checks to see if it is being called on entry. If so, the user exit routine sets the parameter list to the transaction code with no arguments and sets the start-up IMSRXTRC level to 2. The return code is set to 0. For the latest version of the DFSREXXU source code, see the IMS.ADFSSMPL distribution library; the member name is DFSREXXU.

---

## 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. Other host command environments can be accessed with an IMS EXEC as well.

The z/OS environment is provided by TSO in both TSO and non-TSO address spaces. It is used to run other programs such as EXECIO for file I/O. IMS does not manage the z/OS 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, other environments can be used. The environments are:

### APPCMVS

Used for z/OS-specific APPC interfacing

### CPICOMM

Used for CPI Communications

**LU62** Used for z/OS-specific APPC interfacing

**Related reading:** For more information on addressing environments, see *z/OS TSO/E 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 TP PCB 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:

- 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 Binder.
- Use the Binder to define an alias for DFSREXX0 that is the same as the application PGM name.

For example, the following code sample shows a section from the PGM setup job that uses the binder to perform the copy function to the name IVPREXX. The example uses the IVP.

```
/* REXXTDLI SAMPLE - GENERIC APPLICATION DRIVER
/*
//LINK EXEC PGM=IEWL,
// PARM='XREF,LIST,LET,SIZE=(192K,64K)'
//SYSPRINT DD SYSOUT=*
//SDFSRESL DD DISP=SHR,DSN=IMS.SDFSRESL
//SYSLMOD DD DISP=SHR,DSN=IMS1.PGMLIB
//SYSUT1 DD UNIT=(SYSALLDA,SEP=(SYSLMOD,SYSLIN)),
// DISP=(,DELETE,DELETE),SPACE=(CYL,(1,1))
//SYSLIN DD *
INCLUDE SDFSRESL(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.

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. The following code example shows the JCL necessary for MPP region that runs the IVPREXX sample EXEC.



## JCL code used to run 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
/*
//DFSCCTL 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.SDFSEEXEC
/* REXX INPUT LOCATION WHEN STACK IS EMPTY
//SYSTSIN DD *
/*
/* REXX OUTPUT LOCATION
//SYSTSPT DD SYSOUT=*
/* COBOL OUTPUT LOCATION
//SYSOUT DD SYSOUT=*
```

### Related reference:

 [IMS Adapter for REXX exit routine \(DFSREXXU\) \(Exit Routines\)](#)

---

## REXXTDLI commands

These topics contain 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.

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 z/OS. Consequently, you must either use ADDRESS REXXTDLI or ADDRESS REXXIMS to 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 following information describes usage considerations for REXXTDLI calls.

### Format

```
►►—dlicall—┬──┬──┬──┬──►►  
                  └─parm1─┘ └─parm2─┘ └─...─┘
```

The format of a DL/I call varies depending on call type. The parameter formats for supported DL/I calls can be found in “DL/I calls for database management” on page 1, “DL/I calls for transaction management” on page 81, and “DL/I calls for IMS DB system services” on page 35. The parameters for the calls are case-independent, separated by one or more blanks, and are generally REXX variables. See “Parameter handling” on page 365 for detailed descriptions.

### Issuing synchronous callout requests

To issue a synchronous callout (ICAL call) request using the REXXTDLI interface, you must specify the DFSAIB keyword followed by the input and output areas. Both the input and output areas can be specified as a variable, \*mapname, or !token.

The syntax for the ICAL call from the REXXTDLI interface is:

```
►►—ICAL—DFS AIB—In—Out—►►
```

A default length of 1024 bytes will be passed as an input to the AIBOAUSE field for undefined or implicit variables in the output area. To specify larger messages, you will need to issue the STORAGE command.

### 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.

## 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 88 on page 366.

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 TP 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 88. IMS adapter for REXX parameter types and definitions

Type <sup>1</sup>	Parameter Definition
PCB	<p><b>Important:</b> The PCB parameter is not required if the REXXTDLI interface is used for making a synchronous callout request (ICAL call). Instead, the keyword DFSAIB must be specified before the input and output parameters.</p> <p>PCB identifier specified as a variable containing one of the following:</p> <ul style="list-style-type: none"> <li>• PCB name as defined in the PSB generation on the PCBNAME= parameter. See IMS Version 13 System Utilities 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.</li> <li>• An AIB block formatted to DFSAIB specifications. This variable is returned with an updated AIB.</li> <li>• A # followed by PCB offset number (#1=first PCB). Example settings are: <ul style="list-style-type: none"> <li>– IOPCB=:"#1"</li> <li>– ALTPCB=:"#2"</li> <li>– DBPCB=:"#3"</li> </ul> </li> </ul> <p>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.</p>
In	Input variable. It can be specified as a constant, variable, *mapname <sup>2</sup> or !token <sup>3</sup> .
SSA	Input variable with an SSA (segment search argument). It can be specified as a constant, variable, *mapname <sup>2</sup> or !token <sup>3</sup> .
Out	Output variable to store a result after a successful command. It can be specified as a variable, *mapname <sup>2</sup> or !token <sup>3</sup> .
In/Out	Variable that contains input on entry and contains a result after a successful command. It can be specified as a variable, *mapname <sup>2</sup> or !token <sup>3</sup> .
Const	Input constant. This command argument must be the actual value, not a variable containing the value.

**Note:**

1. The parameter types listed in Table 88 correspond to the types shown in Table 1 on page 2, Table 25 on page 82, and Table 5 on page 35, as well as to those shown in Table 89 on page 369.  
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 373 and "MAPPUT" on page 374.
3. For more information on !token, see "STORAGE" on page 377.

## 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.SDFSISRC 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.

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 z/OS 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."
```

The following sample IMS REXX program shows how to use the DL/I ICAL call to send a synchronous callout request message to an OTMA descriptor name

OTMDEST1 with an input string of "Hello from IMS" and a timeout value of 60 seconds. The output data is set in the variable *Output*.

```
Address REXXIMS
Input = 'Hello from IMS';
Timer = 6000
'SET SUBFUNC SENDRECV'
'SET RSNAM1 OTMDEST1'
'SET TIMER Timer'

'ICAL DFSAIB Input Output'
Say Input
Say Output

Outlen = IMSQUERY('OUTLEN')
Say Outlen
Errxtn = IMSQUERY('ERRXTN')
Say Errxtn
```

The following sample shows the output of an IMS REXX program that issued an DL/I ICAL call:

```
DFS3180I INQY ENVIRON Region=BMP      Number=1
DFS3180I INQY ENVIRON Tran=TXCD255  PGM=DFSREXX0
DFS3180I Starting EXEC Name=DFSREXX0
DFS3160I IMS CMD=SET SUBFUNC SENDRECV
DFS3161I REXXIMS  Command=SET      RC=0
DFS3160I IMS CMD=SET RSNAM1 OTMDEST1
DFS3161I REXXIMS  Command=SET      RC=0
DFS3160I IMS CMD=SET TIMER timer
DFS3161I REXXIMS  Command=SET      RC=0
DFS3160I IMS CMD=ICAL DFSAIB Input Output
DFS3161I REXXTDLI Call=ICAL RC=0000 Reason=0000 Status=".."
Hello from IMS
HELLO FROM TM RA APP
50
0
```

### *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 z/OS SUBCOM command and specify either the REXXTDLI or REXXIMS environments. The code looks like this:

```
Address z/OS 'SUBCOM REXXTDLI'
If RC=0 Then Say "IMS Environment is Available."
      Else Say "Sorry, no IMS Environment is 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."
```

### **Related reference:**

"STORAGE" on page 377

"DLIINFO" on page 369

"SETO call" on page 117

"REXXIMS extended commands" on page 369

---

## 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 the following tables and are available when you ADDRESS REXXIMS. DL/I calls are also available when you address the REXXIMS environment.

Table 89. REXXIMS extended commands.

Command	Parameter Types
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

**Note:**

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.

**Related reference:**

“REXXTDLI calls” on page 364

### DLIINFO

The DLIINFO call requests information from the last DL/I call or on a specific PCB.

**Format**

►► DLIINFO—*infoout*—pcbld—►►

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 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.

### Related reference:

“REXXTDLI calls” on page 364

“PCBINFO exec: display available PCBs in current PSB” on page 383

## IMSRXTRC

The IMSRXTRC command is used primarily for debugging. It controls the tracing action taken (that is, how much trace output through SYSTSPRT is sent to the user) while running a REXX program.

### Format

▶—IMSRXTRC—*level*—▶

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. Traced output is sent to the DDNAME SYSTSPRT. See IMS Version 13 Exit Routines for more information on the IMS adapter for REXX exit routine.

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 <i>IMS Version 13 Messages and Codes, Volume 2: Non-DFS Messages</i> .
9	All previous levels.

### Example

Address REXXIMS 'IMSRXTRC 3'

IMSRXTRC is independent of the REXX TRACE instruction.

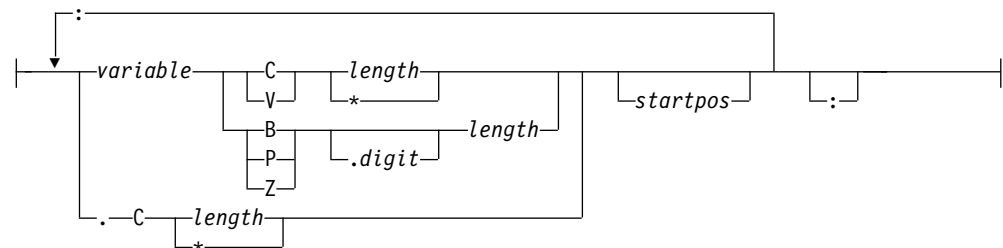
## MAPDEF

The MAPDEF command makes a request to define a data mapping.

### Format

►► MAPDEF *mapname* | A | REPLACE

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 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:

<b>C</b>	Character
<b>V</b>	Variable
<b>B</b>	Binary (numeric)
<b>Z</b>	Zoned decimal (numeric)
<b>P</b>	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 specify an asterisk only for data types C and V. Data type V maps a 2-byte length field preceding the data string, such that when the declared length is 2, it takes 4 bytes.

Valid lengths for data types are:

<b>C</b>	1 - 32767 bytes or *
<b>V</b>	1 - 32765 bytes or *
<b>B</b>	1 - 4 bytes
<b>Z</b>	1 - 12 bytes
<b>P</b>	1 - 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 two 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

►►—MAPGET—*mapname*—*buffer*—◄◄

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

►—MAPPUT—*mapname*—*buffer*—◄

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 following code example. 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.

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

```

▶▶ SET SUBFUNC—variable
      ZZ—variable
      RSNAME1—variable
      TIMER—variable

```

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.

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.

If you are issuing a synchronous callout request by making an ICAL call, the following usage rules apply:

- The SET SUBFUNC command must be issued with the subfunction code set to SENDRECV.
- The SET RSNAME1 command must be issued with the variable set to the OTMA Descriptor name.
- Optionally, the SET TIMER command can be issued to set the ICAL timeout value. The timeout value must be numeric and can contain up to six digits.

## 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

```

┌─── SRRBACK—return_code ───┐
└─── SRRCMIT—return_code ───┘

```

Call Name	DB/DC	DBCTL	DCCTL	DB Batch	TM Batch
SRRBACK, SRRCMIT	X		X		

## Usage

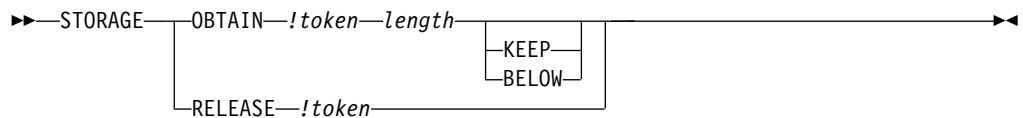
The return code from the SRR command is returned and placed in the *return\_code* variable name as well as the REXX variable RC.

For more information on SRRBACK and SRRCMIT, see *IMS Version 13 Communications and Connections* and *SAA CPI 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. Once a token is allocated, you can use it in REXXTDLI DL/I calls or on the STORAGE RELEASE command.

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.

## 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'
```

### Related reference:

“REXXTDLI calls” on page 364

“IMSQUERY extended functions” on page 379

## 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 z/OS system operator.

### Format

►—WTOR—*message*—*response*—◄

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 z/OS 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 z/OS 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 */

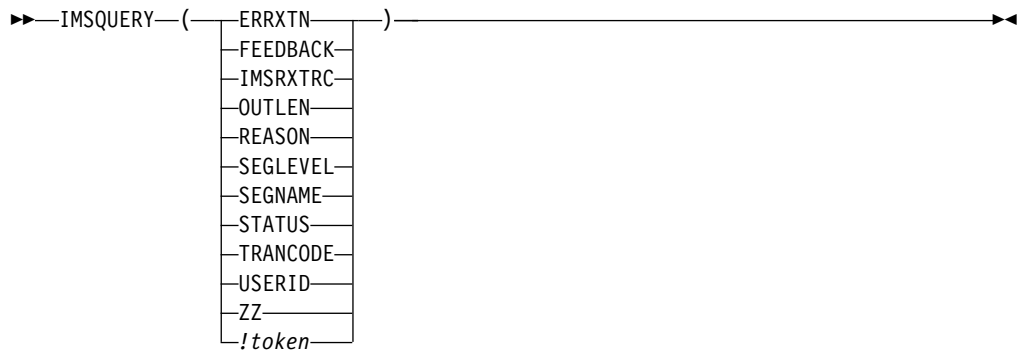
```

## IMSQUERY extended functions

The IMSQUERY function is available to query certain IMS information either on the environment or on the prior DL/I call.

The IMSQUERY function can be used to check for the return and reason codes after a synchronous callout request is made using the ICAL call. If the return or reason code indicates that partial output data is being returned, you can issue the IMSQUERY function to retrieve the output data length and the error extension codes.

### 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 values in the following list.

### Argument

#### Description of Data Returned

#### ERRXTN

The error extension code when an error response message is returned after a synchronous callout request is made using ICAL.

#### FEEDBACK

FEEDBACK area from current PCB.

#### IMSRXTRC

Current IMSRXTRC trace level #.

#### OUTLEN

The output length of the response message when a partial response is returned after a synchronous callout request is made using ICAL.

#### 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 REXXTDLI call (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 .
```

### Related reference:

-  [IMS Adapter for REXX exit routine \(DFSREXXU\) \(Exit Routines\)](#)
- ["REXXTDLI commands" on page 363](#)
- ["STORAGE" on page 377](#)

---

## Sample execs using REXXTDLI

The following samples of REXX execs show how to use REXXTDLI to access IMS services.

The example sets are designed to highlight various features of writing IMS applications in REXX. The samples 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.

## SAY exec: for expression evaluation

The following code example 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

```
/* 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:' Arg '=' X
End
'ISRT IOPCB MSG'
Exit RC
```

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 the following figure. This figure shows how you can enter a new exec to be executed under IMS.

```
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:' Arg '=' X
000009 End
000010
000011 'ISRT IOPCB MSG'
000012 Exit RC
***** ***** BOTTOM OF DATA *****
```

Figure 10. 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 the following figure.

```
EXPRESSION: 5*5+7 = 32
EXEC SAY ended with RC= 0
```

Figure 11. Example output from the SAY exec

## PCBINFO exec: display available PCBs 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. 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=DFSM02
          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 12. 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=DFSM02
          Date=89320 Time=1553243
PCB # 2: Type=DB, DBD =DI21PART Status=   Level=00 Opt=G
EXEC PCBINFO ended with RC= 0
```

Figure 13. Example output of PCBINFO exec on a PSB with a database PCB

### PCBINFO exec listing

```
/* 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, in other words, 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 */
TPPCB=(Substr(PCBInfo,13,1)='00'x) /* Date Field, must be TP PCB */
If TPPCB 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

```

**Related reference:**

“DLIINFO” on page 369

## PART execs: database access examples

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 also described.

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 the figure below.

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 the figure below.

```

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 14. 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 the following figure.

```

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 15. Example output of PARTNAME exec

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.

**Related Reading:** For details, see *IMS Version 13 Installation*.

### **PARTNUM exec: show set of parts near a specified number**

The following code example is designed to be run by the IVPREXX exec with PSB=DFSSAM02.

#### **PARTNUM exec: show set of parts near a specified number**

```

/* 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

```

### **PARTNAME exec: show a set of parts with a similar name**

The REXX exec shown in the following code example is designed to be run by the IVPREXX exec with PSB=DFSSAM02.

The following PARTNAME exec code is used to show parts with similar names.

```

/* 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

```

### **DFSSAM01 exec: load the parts database**

For the latest version of the DFSSAM01 source code, see the IMS.ADFSEEXEC 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 the code example at the end of this topic.

Not all commands are allowed through transaction AOI, and some setup needs to be done to use this AOI.

Some examples of DOCMD are given in the following figures.

```

Please supply an IMS Command to execute.
EXEC DOCMD ended with RC= 0

```

*Figure 16. 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 17. 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 18. 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
NDSL2A1 SLU2 00000000 0 0 0 0 0 ASR IDLE
NDSL2A2 SLU2 00000000 0 0 0 0 0 ASR IDLE
NDSL2A3 SLU2 00000000 0 0 0 0 0 ASR IDLE
NDSL2A4 SLU2 00000000 0 0 0 0 0 ASR IDLE
NDSL2A5 SLU2 00000000 0 0 0 0 0 IDLE
NDSL2A6 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 19. 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 20. 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 21. Output from => DOCMD /DIS LTERM ALL;ENQCT>0

The source code for the DOCMD exec is shown in the following code example.

### DOCMD exec: process an IMS command

```

/*****
/* 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 DFSREXX 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;SYSOUT Output to SYSOUT */
/*           TACP18 DOCMD DIS NODE ALL;CID>0 & TYPE='SLU2'      */
/*****
Address REXXTDLI
Parse Upper Arg Cmd ';' Expression ';' SysOut
Cmd=Strip(Cmd);
Expression=Strip(Expression)
SysOut=(SysOut='')
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
Else Do
  Call ParseCmd Expression
  Do i=1 to Line.0
    If AllOpt then Line=Line.i
    Else Line=Substr(Line.i,5)
    If SysOut then Do
      Push Line
      'EXECIO 1 DISKW DOCMD'
    End
    Else Do
      Call SayIt Line
      Calls=Calls+1
    End
  End
  If SysOut then Do
    'EXECIO 0 DISKW DOCMD ( FINIS'
  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
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
  End
  Else Do
    Call SayIt 'Command did not produce a header',
      'record, first record's type='RecType
  End
  Return 12

```

```

End
Return 0

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
  End
Return

```

```

SayIt: Procedure
  Parse Arg Line
  'ISRT IOPCB LINE'
Return RC

```

### Related concepts:

 [IMS security \(System Administration\)](#)

## IVPREXX sample application

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 the latest version of the IVPREXX source code, look for the IVPREXX member in the IMS.ADFSEXEC distribution library.

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 parameter list of the IMS region program. 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. Check the z/OS system console or region output for more details.

**Related reading:** For more information about REXX errors and messages, see *z/OS TSO/E REXX Reference*.

### *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 concepts:**

Chapter 3, “IMS Adapter for REXX reference,” on page 359





---

## Chapter 4. Java programming reference

These topics contain reference information for the classes, interfaces, and methods supported by the IMS solutions for Java development.


---

### IMS Universal drivers support for JDBC

The IMS Universal JDBC driver and the IMS Universal JCA/JDBC driver supports the following methods in the JDBC 4.0 specifications.

**Related concepts:**

 [Programming with the IMS Universal JDBC driver \(Application Programming\)](#)

 [Programming using the IMS Universal Database resource adapter \(Application Programming\)](#)

**Related reference:**

“Java API documentation (Javadoc)” on page 414

### javax.sql.Clob methods supported

The javax.sql.Clob interface represents the mapping in Java for the SQL CLOB type. In Java applications that use the IMS Universal drivers, the Clob data type is supported only for the retrieval and storage of XML data.

The following table describes which methods are supported by the IMS Universal JDBC driver and the IMS Universal JCA/JDBC driver for the Clob interface.

*Table 90. IMS Universal JDBC driver and IMS Universal JCA/JDBC driver support for the Clob interface*

JDBC method	IMS Universal JDBC driver and IMS Universal JCA/JDBC driver support
free()	No
getAsciiStream()	Yes
getCharacterStream()	Yes
getCharacterStream(long pos, long length)	No
getSubString(long pos, int length)	Yes
length()	Yes
position(Clob searchstr, long start)	No
position(String searchstr, long start)	No
setAsciiStream(long pos)	No
setCharacterStream(long pos)	No
setString(long pos, String str)	No
setString(long pos, String str, int offset, int len)	No
truncate(long len)	No

### java.sql.Connection methods supported

The Connection object represents a connection (session) with a specific database.

**Recommendation:** Because the IMS Universal drivers are built on industry standards and open specifications, and provide more flexibility and improved support for connectivity, data access methods, and transaction processing options, use the IMS Universal drivers for developing your Java applications to access IMS.

The following table lists the methods that are supported by the IMS JDBC drivers for the Connection interface.

*Table 91. IMS JDBC drivers support for Connection.*

JDBC method	IMS Universal JDBC driver and IMS Universal JCA/JDBC driver support	IMS classic JDBC driver support
clearWarnings()	Yes	No
close()	Yes	Yes
commit()	Yes	No
createStatement()	Yes	Yes
createStatement(int resultSetType, int resultSetConcurrency)	Yes	Yes
createStatement(int resultSetType, int resultSetConcurrency, int resultSetHoldability)	Yes	No
getAutoCommit()	Yes	No
getCatalog()	Yes	No
getHoldability()	Yes	No
getMetaData()	Yes	Yes
getTransactionIsolation()	Yes	No
getTypeMap()	Yes	No
getWarnings()	Yes	Yes
isClosed()	Yes	Yes
isReadOnly()	Yes	Yes
nativeSQL(String sql)	Yes	No
prepareCall(String sql)	No	No
prepareCall(String sql, int resultSetType, int resultSetConcurrency)	No	No
prepareCall(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability)	No	No
prepareStatement(String sql)	Yes	Yes
prepareStatement(String sql, int autoGeneratedKeys)	No	No
prepareStatement(String sql, int[] columnIndexes)	No	No
prepareStatement(String sql, int resultSetType, int resultSetConcurrency)	Yes	Yes
prepareStatement(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability)	Yes	No
prepareStatement(String sql, String[] columnNames)	No	No
releaseSavepoint(Savepoint savepoint)	No	No
rollback()	Yes	No
rollback(Savepoint savepoint)	No	No
setAutoCommit(boolean autoCommit)	Yes	No
setCatalog(String catalog)	Yes	No
setHoldability(int holdability)	Yes	No
setReadOnly(boolean readOnly)	Yes	No
setSavepoint()	No	No
setTransactionIsolation(int level)	Yes	No
setTypeMap(Map<String,Class<?>> map)	No	No

## Related concepts:

 Programming with the IMS Universal JDBC driver (Application Programming)

## java.sql.DatabaseMetaData methods supported

The DatabaseMetaData interface provides comprehensive information about the database as a whole.

**Recommendation:** Because the IMS Universal drivers are built on industry standards and open specifications, and provide more flexibility and improved support for connectivity, data access methods, and transaction processing options, use the IMS Universal drivers for developing your Java applications to access IMS.

The following methods are supported by the IMS JDBC drivers for the DatabaseMetaData interface.

Table 92. IMS JDBC drivers support for DatabaseMetaData.

JDBC method	IMS Universal JDBC driver and IMS Universal JCA/JDBC driver support	IMS classic JDBC driver support
allProceduresAreCallable()	Yes	Yes
allTablesAreSelectable()	Yes	Yes
dataDefinitionCausesTransactionCommit()	Yes	Yes
dataDefinitionIgnoredInTransactions()	Yes	Yes
deletesAreDetected(int type)	Yes	Yes
doesMaxRowSizeIncludeBlobs()	Yes	Yes
getAttributes(String catalog, String schemaPattern, String typeNamePattern, String attributeNamePattern)	Yes	No
getBestRowIdentifier(String catalog, String schema, String table, int scope, boolean nullable)	Yes	No
getCatalogs()	Yes	No
	A second column, <code>TIMESTAMP</code> , is added to the returned Resultset object as a String that represents the PSB timestamp. IMS 13 APAR PI62580 (PTF UI39459) is required for this <code>TIMESTAMP</code> column.	
getCatalogSeparator()	Yes	Yes
getCatalogTerm()	Yes	Yes
getColumnPrivileges(String catalog, String schema, String table, String columnNamePattern)	Yes	No
getColumns(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern)	Yes	No
getConnection()	Yes	Yes
getCrossReference(String primaryCatalog, String primarySchema, String primaryTable, String foreignCatalog, String foreignSchema, String foreignTable)	Yes	No
getDatabaseMajorVersion()	Yes	No
getDatabaseMinorVersion()	Yes	No
getDatabaseProductName()	Yes	Yes
getDatabaseProductVersion()	Yes	Yes
getDefaultTransactionIsolation()	Yes	Yes
getDriverMajorVersion()	Yes	Yes
getDriverMinorVersion()	Yes	Yes
getDriverName()	Yes	Yes

Table 92. IMS JDBC drivers support for DatabaseMetaData (continued).

JDBC method	IMS Universal JDBC driver and IMS Universal JCA/JDBC driver support	IMS classic JDBC driver support
getDriverVersion()	Yes	Yes
getExportedKeys(String catalog, String schema, String table)	Yes	No
getExtraNameCharacters()	Yes	Yes
getIdentifierQuoteString()	Yes	Yes
getImportedKeys(String catalog, String schema, String table)	Yes	No
getIndexInfo(String catalog, String schema, String table, boolean unique, boolean approximate)	Yes	No
getJDBCMinorVersion()	Yes	Yes
getJDBCMajorVersion()	Yes	Yes
getMaxBinaryLiteralLength()	Yes	Yes
getMaxCatalogNameLength()	Yes	Yes
getMaxCharLiteralLength()	Yes	Yes
getMaxColumnNameLength()	Yes	Yes
getMaxColumnsInGroupBy()	Yes	Yes
getMaxColumnsInIndex()	Yes	Yes
getMaxColumnsInOrderBy()	Yes	Yes
getMaxColumnsInSelect()	Yes	Yes
getMaxColumnsInTable()	Yes	Yes
getMaxConnections()	Yes	Yes
getMaxCursorNameLength()	Yes	Yes
getMaxIndexLength()	Yes	Yes
getMaxProcedureNameLength()	Yes	Yes
getMaxRowSize()	Yes	Yes
getMaxSchemaNameLength()	Yes	Yes
getMaxStatementLength()	Yes	Yes
getMaxStatements()	Yes	Yes
getMaxTableNameLength()	Yes	Yes
getMaxTablesInSelect()	Yes	Yes
getMaxUserNameLength()	Yes	Yes
getNumericFunctions()	Yes	Yes
getPrimaryKeys(String catalog, String schema, String table)	Yes	No
getProcedureColumns(String catalog, String schemaPattern, String procedureNamePattern, String columnNamePattern)	Yes	No
getProcedures(String catalog, String schemaPattern, String procedureNamePattern)	Yes	No
getProcedureTerm()	Yes	No
getResultSetHoldability()	Yes	No
getSchemas()	Yes	No

The following columns are added as column 3, 4, and 5:

- Column 3: PCB\_PROCESSING\_OPTIONS, a String that represents PCB procopts
- Column 4: DBD\_NAME, a String that represents the referenced DBD name
- Column 5: DBD\_TIMESTAMP, a String that represents the referenced DBD timestamp

IMS 13 APAR PI62580 (PTF UI39459) is required for columns 3, 4, and 5.

Table 92. IMS JDBC drivers support for DatabaseMetaData (continued).

JDBC method	IMS Universal JDBC driver and IMS Universal JCA/JDBC driver support	IMS classic JDBC driver support
getSchemaTerm()	Yes	No
getSearchStringEscape()	Yes	No
getSQLKeywords()	Yes	Yes
getStringFunctions()	Yes	Yes
getSuperTables(String catalog, String schemaPattern, String tableNamePattern)	Yes	No
getSuperTypes(String catalog, String schemaPattern, String typeNamePattern)	Yes	No
getSystemFunctions()	Yes	Yes
getTablePrivileges(String catalog, String schemaPattern, String tableNamePattern)	Yes	No
getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)	Yes	No
getTableTypes()	Yes	No
getTimeDateFunctions()	Yes	Yes
getTypeInfo()	Yes	No
getUDTs(String catalog, String schemaPattern, String typeNamePattern, int[] types)	Yes	No
getURL()	Yes	Yes
getUserName()	Yes	No
getVersionColumns(String catalog, String schema, String table)	Yes	No
insertsAreDetected(int type)	Yes	Yes
isCatalogAtStart()	Yes	Yes
isReadOnly()	Yes	No
locatorsUpdateCopy()	Yes	Yes
nullPlusNonNullsNull()	Yes	Yes
nullsAreSortedAtEnd()	Yes	No
nullsAreSortedAtStart()	Yes	No
nullsAreSortedLow()	Yes	No
othersDeletesAreVisible(int type)	Yes	Yes
othersInsertsAreVisible(int type)	Yes	Yes
othersUpdatesAreVisible(int type)	Yes	Yes
ownDeletesAreVisible(int type)	Yes	Yes
ownInsertsAreVisible(int type)	Yes	Yes
ownUpdatesAreVisible(int type)	Yes	Yes
storesLowerCaseIdentifiers()	Yes	Yes
storesLowerCaseQuotedIdentifiers()	Yes	Yes
storesMixedCaseIdentifiers()	Yes	Yes
storesMixedCaseQuotedIdentifiers()	Yes	Yes
storesUpperCaseIdentifiers()	Yes	Yes
storesUpperCaseQuotedIdentifiers()	Yes	Yes
supportsAlterTableWithAddColumn()	Yes	Yes
supportsAlterTableWithDropColumn()	Yes	Yes
supportsANSI92EntryLevelSQL()	Yes	Yes
supportsANSI92FullSQL()	Yes	Yes
supportsANSI92IntermediateSQL()	Yes	Yes
supportsBatchUpdates()	Yes	Yes
supportsCatalogsInDataManipulation()	Yes	Yes
supportsCatalogsInIndexDefinitions()	Yes	Yes
supportsCatalogsInPrivilegeDefinitions()	Yes	Yes
supportsCatalogsInProcedureCalls()	Yes	Yes


Table 92. IMS JDBC drivers support for DatabaseMetaData (continued).

JDBC method	IMS Universal JDBC driver and IMS Universal JCA/JDBC driver support	IMS classic JDBC driver support
supportsCatalogsInTableDefinitions()	Yes	Yes
supportsColumnAliasing()	Yes	Yes
supportsConvert()	Yes	Yes
supportsConvert(int fromType, int toType)	Yes	Yes
supportsCoreSQLGrammar()	Yes	Yes
supportsCorrelatedSubqueries()	Yes	Yes
supportsDataDefinitionAnd DataManipulationTransactions()	Yes	Yes
supportsDataManipulationTransactionsOnly()	Yes	Yes
supportsDifferentTableCorrelationNames()	Yes	Yes
supportsExpressionsInOrderBy()	Yes	Yes
supportsExtendedSQLGrammar()	Yes	Yes
supportsFullOuterJoins()	Yes	Yes
supportsGetGeneratedKeys()	Yes	Yes
supportsGroupByBeyondSelect()	Yes	Yes
supportsGroupByUnrelated()	Yes	Yes
supportsIntegrityEnhancementFacility()	Yes	Yes
supportsLikeEscapeClause()	Yes	Yes
supportsLimitedOuterJoins()	Yes	Yes
supportsMinimumSQLGrammar()	Yes	Yes
supportsMixedCaseIdentifiers()	Yes	Yes
supportsMixedCaseQuotedIdentifiers()	Yes	Yes
supportsMultipleOpenResults()	Yes	Yes
supportsMultipleResultSets()	Yes	Yes
supportsMultipleTransactions()	Yes	Yes
supportsNamedParameters()	Yes	Yes
supportsNonNullableColumns()	Yes	Yes
supportsOpenCursorsAcrossCommit()	Yes	Yes
supportsOpenCursorsAcrossRollback()	Yes	Yes
supportsOpenStatementsAcrossCommit()	Yes	Yes
supportsOpenStatementsAcrossRollback()	Yes	Yes
supportsOrderByUnrelated()	Yes	Yes
supportsOuterJoins()	Yes	Yes
supportsPositionedDelete()	Yes	Yes
supportsPositionedUpdate()	Yes	Yes
supportsResultSetConcurrency(int type, int concurrency)	Yes	Yes
supportsResultSetHoldability(int holdability)	Yes	No
supportsResultSetType(int type)	Yes	Yes
supportsSavepoints()	Yes	Yes
supportsSchemasInDataManipulation()	Yes	Yes
supportsSchemasInIndexDefinitions()	Yes	Yes
supportsSchemasInPrivilegeDefinitions()	Yes	Yes
supportsSchemasInProcedureCalls()	Yes	Yes
supportsSchemasInTableDefinitions()	Yes	Yes
supportsSelectForUpdate()	Yes	Yes
supportsStatementPooling()	Yes	Yes
supportsStoredProcedures()	Yes	Yes
supportsSubqueriesInComparisons()	Yes	Yes
supportsSubqueriesInExists()	Yes	Yes

Table 92. IMS JDBC drivers support for DatabaseMetaData (continued).

JDBC method	IMS Universal JDBC driver and IMS Universal JCA/JDBC driver support	IMS classic JDBC driver support
supportsSubqueriesInIns()	Yes	Yes
supportsSubqueriesInQuantifieds()	Yes	Yes
supportsTableCorrelationNames()	Yes	Yes
supportsTransactionIsolationLevel(int level)	Yes	Yes
supportsTransactions()	Yes	Yes
supportsUnion()	Yes	Yes
supportsUnionAll()	Yes	Yes
updatesAreDetected(int type)	Yes	Yes
usesLocalFilePerTable()	Yes	Yes
usesLocalFiles()	Yes	Yes

**Related concepts:**

 Programming with the IMS Universal JDBC driver (Application Programming)

## javax.sql.DataSource methods supported

A DataSource object is a factory for connections to the physical data source that this DataSource object represents.

The following table list which methods are supported by the IMS Universal JDBC driver and the IMS Universal JCA/JDBC driver for the DataSource interface. The DataSource interface is not supported by the IMS classic JDBC driver.

Table 93. IMS Universal JDBC driver and IMS Universal JCA/JDBC driver support for DataSource.

JDBC method	IMS Universal JDBC driver and IMS Universal JCA/JDBC driver support
getConnection()	Yes
getConnection(String username, String password)	Yes
getLoginTimeout()	Yes
getLogWriter()	Yes
setLoginTimeout(int seconds)	Yes
setLogWriter(PrintWriter out)	Yes

**Related concepts:**

 Programming with the IMS Universal JDBC driver (Application Programming)

## java.sql.Driver methods supported

The Driver class is used for connecting to a database using the JDBC DriverManager interface.

The following table lists which methods are supported by the IMS JDBC drivers for the Driver interface.


Table 94. IMS JDBC drivers support for Driver.

JDBC method	IMS Universal JDBC driver and IMS Universal JCA/JDBC driver support	IMS classic JDBC driver support
acceptsURL(String url)	Yes	Yes
connect(String url, Properties info)	Yes	Yes

Table 94. IMS JDBC drivers support for Driver (continued).

JDBC method	IMS Universal JDBC driver and IMS Universal JCA/JDBC driver support	IMS classic JDBC driver support
getMajorVersion()	Yes	Yes
getMinorVersion()	Yes	Yes
getPropertyInfo(String url, Properties info)	Yes	Yes
jdbcCompliant()	Yes	Yes

**Related concepts:**

 [Programming with the IMS Universal JDBC driver \(Application Programming\)](#)

## java.sql.ParameterMetaData methods supported

An object that can be used to get information about the types and properties of the parameters in a PreparedStatement object.

The following table list which methods are supported by the IMS Universal JDBC driver and the IMS Universal JCA/JDBC driver for the ParameterMetaData interface. The ParameterMetaData interface is not supported by the IMS classic JDBC driver.

Table 95. IMS Universal JDBC driver and IMS Universal JCA/JDBC driver support for ParameterMetaData.

JDBC method	IMS Universal JDBC driver and IMS Universal JCA/JDBC driver support
getParameterCount	Yes
isNullable	Yes
isSigned	Yes
getPrecision	Yes
getScale	Yes
getParameterType	Yes
getParameterTypeName	Yes
getParameterClassName	Yes
getParameterMode	Yes

## java.sql.PreparedStatement methods supported

The PreparedStatement object represents a precompiled SQL statement.

**Recommendation:** Because the IMS Universal drivers are built on industry standards and open specifications, and provide more flexibility and improved support for connectivity, data access methods, and transaction processing options, use the IMS Universal drivers for developing your Java applications to access IMS.

The following methods are supported by the IMS JDBC drivers for the PreparedStatement interface.

Table 96. IMS JDBC drivers support for PreparedStatement.


JDBC method	IMS Universal JDBC driver and IMS Universal JCA/JDBC driver support	IMS classic JDBC driver support
addBatch()	No	No
clearParameters()	Yes	Yes
execute()	Yes	Yes
executeQuery()	Yes	Yes



Table 96. IMS JDBC drivers support for PreparedStatement (continued).

JDBC method	IMS Universal JDBC driver and IMS Universal JCA/JDBC driver support	IMS classic JDBC driver support
executeUpdate()	Yes	Yes
getMetaData()	Yes	No
getParameterMetaData()	Yes	No
setArray(int i, Array x)	Yes	No
setAsciiStream(int parameterIndex, InputStream x, int length)	No	No
setBigDecimal(int parameterIndex, BigDecimal x)	Yes	Yes
setBinaryStream(int parameterIndex, InputStream x, int length)	No	No
setBlob(int i, Blob x)	No	No
setBoolean(int parameterIndex, boolean x)	Yes	Yes
setByte(int parameterIndex, byte x)	Yes	Yes
setBytes(int parameterIndex, byte[] x)	Yes	Yes
setCharacterStream(int parameterIndex, Reader reader, int length)	No	No
setClob(int i, Clob x)	Yes	No
setDate(int parameterIndex, Date x)	Yes	Yes
setDate(int parameterIndex, Date x, Calendar cal)	No	No
setDouble(int parameterIndex, double x)	Yes	Yes
setFloat(int parameterIndex, float x)	Yes	Yes
setInt(int parameterIndex, int x)	Yes	Yes
setLong(int parameterIndex, long x)	Yes	Yes
setNull(int parameterIndex, int sqlType)	No	No
setNull(int paramIndex, int sqlType, String typeName)	No	No
setObject(int parameterIndex, Object x)	Yes	No
setObject(int parameterIndex, Object x, int targetSqlType)	No	No
setObject(int parameterIndex, Object x, int targetSqlType, int scale)	No	No
setRef(int i, Ref x)	No	No
setShort(int parameterIndex, short x)	Yes	Yes
setString(int parameterIndex, String x)	Yes	Yes
setTime(int parameterIndex, Time x)	Yes	Yes
setTime(int parameterIndex, Time x, Calendar cal)	No	No
setTimestamp(int parameterIndex, Timestamp x)	Yes	Yes
setUnicodeStream(int parameterIndex, InputStream x, int length)	No	No
setURL(int parameterIndex, URL x)	No	No

**Related concepts:**

 [Programming with the IMS Universal JDBC driver \(Application Programming\)](#)

## java.sql.Statement methods supported

The Statement object is used for executing a static SQL statement and returning the results it produces.

**Recommendation:** Because the IMS Universal drivers are built on industry standards and open specifications, and provide more flexibility and improved

support for connectivity, data access methods, and transaction processing options, use the IMS Universal drivers for developing your Java applications to access IMS.

The following table lists the methods that are supported by the IMS JDBC drivers for the Statement interface.

*Table 97. IMS JDBC drivers support for Statement.*

JDBC method	IMS Universal JDBC driver and IMS Universal JCA/JDBC driver support	IMS classic JDBC driver support
addBatch(String sql)	No	No
cancel()	No	No
clearBatch()	No	No
clearWarnings()	Yes	Yes
close()	Yes	Yes
execute(String sql)	Yes	Yes
execute(String sql, int autoGeneratedKeys)	No	No
execute(String sql, int[] columnIndexes)	No	No
execute(String sql, String[] columnNames)	No	No
executeBatch()	No	No
executeQuery(String sql)	Yes	Yes
executeUpdate(String sql)	Yes	Yes
executeUpdate(String sql, int autoGeneratedKeys)	No	No
executeUpdate(String sql, int[] columnIndexes)	No	No
executeUpdate(String sql, String[] columnNames)	No	No
getConnection()	Yes	Yes
getFetchDirection()	Yes	No
getFetchSize()	Yes	No
getGeneratedKeys()	No	No
getMaxFieldSize()	Yes	No
getMaxRows()	Yes	No
getMoreResults()	Yes	No
getMoreResults(int current)	Yes	No
getQueryTimeout()	Yes	No
getResultSet()	Yes	Yes
getResultSetConcurrency()	Yes	No
getResultSetHoldability()	Yes	No
getResultSetType()	Yes	No
getUpdateCount()	Yes	No
getWarnings()	Yes	Yes
setCursorName(String name)	No	No
setEscapeProcessing(boolean enable)	Yes	No
setFetchDirection(int direction)	Yes	No
setFetchSize(int rows)	Yes	No
setMaxFieldSize(int max)	Yes	No
setMaxRows(int max)	Yes	Yes
setQueryTimeout(int seconds)	Yes	No

**Related concepts:**

 [Programming with the IMS Universal JDBC driver \(Application Programming\)](#)

## java.sql.ResultSet methods supported

A `ResultSet` object is a table of data that represents a database result set, which is usually generated by executing a statement that queries the database.

**Recommendation:** Because the IMS Universal drivers are built on industry standards and open specifications, and provide more flexibility and improved support for connectivity, data access methods, and transaction processing options, use the IMS Universal drivers to develop Java applications that access IMS.

The following table describes the `ResultSet` field constants that are supported by the IMS Universal JDBC driver and the IMS Universal JCA/JDBC driver.

Table 98. `ResultSet` field constants supported by the IMS Universal JDBC driver and the IMS Universal JCA/JDBC driver

Field constant	IMS Universal JDBC driver and IMS Universal JCA/JDBC driver support
<code>ResultSet.CLOSE_CURSORS_AT_COMMIT</code>	Yes <sup>1</sup>
<code>ResultSet.CONCUR_READ_ONLY</code>	Yes
<code>ResultSet.CONCUR_UPDATABLE</code>	Yes
<code>ResultSet.FETCH_FORWARD</code>	Yes <sup>2</sup>
<code>ResultSet.FETCH_REVERSE</code>	Yes <sup>2</sup>
<code>ResultSet.FETCH_UNKNOWN</code>	Yes <sup>2</sup>
<code>ResultSet.HOLD_CURSORS_OVER_COMMIT</code>	No <sup>3</sup>
<code>ResultSet.TYPE_FORWARD_ONLY</code>	Yes
<code>ResultSet.TYPE_SCROLL_INSENSITIVE</code>	Yes
<code>ResultSet.TYPE_SCROLL_SENSITIVE</code>	No <sup>3</sup>

**Note:**

1. This is the processing model that is used by IMS DB.
2. This is a hint to the JDBC driver. No special processing is performed by IMS DB.
3. Not supported by IMS DB.

The following methods are supported by the IMS JDBC drivers for the `ResultSet` interface.

Table 99. IMS JDBC drivers support for the `ResultSet` interface

JDBC method	IMS Universal JDBC driver and IMS Universal JCA/JDBC driver support	IMS classic JDBC driver support
<code>absolute(int row)</code>	Yes	No
<code>afterLast()</code>	Yes	Yes
<code>beforeFirst()</code>	Yes	No
<code>cancelRowUpdates()</code>	Yes	No
<code>clearWarnings()</code>	Yes	Yes
<code>close()</code>	Yes	Yes
<code>deleteRow()</code>	Yes	No
<code>findColumn(String columnName)</code>	Yes	Yes

Table 99. IMS JDBC drivers support for the ResultSet interface (continued)

JDBC method	IMS Universal JDBC driver and IMS Universal JCA/JDBC driver support	IMS classic JDBC driver support
first()	Yes	No
getArray(int i)	Yes	No
getArray(String colName)	Yes	No
getAsciiStream(int columnIndex)	No	No
getAsciiStream(String columnName)	No	No
getBigDecimal(int columnIndex)	Yes	Yes
getBigDecimal(int columnIndex, int scale)	Yes	Yes
getBigDecimal(String columnName)	Yes	Yes
getBigDecimal(String columnName, int scale)	Yes	Yes
getBinaryStream(int columnIndex)	No	No
getBinaryStream(String columnName)	No	No
getBlob(int i)	No	No
getBlob(String colName)	No	No
getBoolean(int columnIndex)	Yes	Yes
getBoolean(String columnName)	Yes	Yes
getByte(int columnIndex)	Yes	Yes
getByte(String columnName)	Yes	Yes
getBytes(int columnIndex)	Yes	Yes
getBytes(String columnName)	Yes	Yes
getCharacterStream(int columnIndex)	No	No
getCharacterStream(String columnName)	No	No
getClob(int i)	Yes (for retrieval of XML only)	No
getClob(String colName)	Yes (for retrieval of XML only)	No
getConcurrency()	Yes	No
getCursorName()	No	No
getDate(int columnIndex)	Yes	Yes
getDate(int columnIndex, Calendar cal)	Yes	No
getDate(String columnName)	Yes	No
getDate(String columnName, Calendar cal)	Yes	No
getDouble(int columnIndex)	Yes	Yes
getDouble(String columnName)	Yes	Yes
getFetchDirection()	Yes	No
getFetchSize()	Yes	No
getFloat(int columnIndex)	Yes	Yes
getFloat(String columnName)	Yes	Yes
getInt(int columnIndex)	Yes	Yes

Table 99. IMS JDBC drivers support for the *ResultSet* interface (continued)

JDBC method	IMS Universal JDBC driver and IMS Universal JCA/JDBC driver support	IMS classic JDBC driver support
<code>getInt(String columnName)</code>	Yes	Yes
<code>getLong(int columnIndex)</code>	Yes	Yes
<code>getLong(String columnName)</code>	Yes	Yes
<code>getMetaData()</code>	Yes	Yes
<code>getObject(int columnIndex)</code>	Yes	No
<code>getObject(String columnName)</code>	Yes	No
<code>getObject(int i, Map&lt;String,Class&lt;?&gt;&gt; map)</code>	No	No
<code>getRef(int i)</code>	No	No
<code>getRef(String colName)</code>	No	No
<code>getRow()</code>	Yes	No
<code>getShort(int columnIndex)</code>	Yes	Yes
<code>getShort(String columnName)</code>	Yes	Yes
<code>getStatement()</code>	Yes	No
<code>getString(int columnIndex)</code>	Yes	Yes
<code>getString(String columnName)</code>	Yes	Yes
<code>getTime(int columnIndex)</code>	Yes	Yes
<code>getTime(String columnName)</code>	Yes	No
<code>getTime(String columnName, Calendar cal)</code>	Yes	No
<code>getTime(int columnIndex, Calendar cal)</code>	Yes	No
<code>getTimestamp(int columnIndex)</code>	Yes	Yes
<code>getTimestamp(int columnIndex, Calendar cal)</code>	Yes	No
<code>getTimestamp(String columnName)</code>	Yes	Yes
<code>getTimestamp(String columnName, Calendar cal)</code>	Yes	No
<code>getType()</code>	Yes	No
<code>getUnicodeStream(int columnIndex)</code>	No	No
<code>getUnicodeStream(String columnName)</code>	No	No
<code>getURL(int columnIndex)</code>	No	No
<code>getURL(String columnName)</code>	No	No
<code>getWarnings()</code>	Yes	Yes
<code>insertRow()</code>	No	No
<code>isAfterLast()</code>	Yes	Yes
<code>isBeforeFirst()</code>	Yes	Yes
<code>isFirst()</code>	Yes	No
<code>isLast()</code>	Yes	No
<code>last()</code>	Yes	No

Table 99. IMS JDBC drivers support for the ResultSet interface (continued)

JDBC method	IMS Universal JDBC driver and IMS Universal JCA/JDBC driver support	IMS classic JDBC driver support
moveToCurrentRow()	No	No
moveToInsertRow()	No	No
next()	Yes	Yes
previous()	Yes	No
refreshRow()	No	No
relative(int rows)	Yes	No
rowDeleted()	No	No
rowInserted()	No	No
rowUpdated()	No	No
setFetchDirection(int direction)	Yes	No
setFetchSize(int rows)	Yes	No
updateArray(int columnIndex, Array x)	Yes	No
updateArray(String columnName, Array x)	Yes	No
updateAsciiStream(int columnIndex, InputStream x, int length)	No	No
updateAsciiStream(String columnName, InputStream x, int length)	No	No
updateBigDecimal(int columnIndex, BigDecimal x)	Yes	No
updateBigDecimal(String columnName, BigDecimal x)	Yes	No
updateBinaryStream(int columnIndex, InputStream x, int length)	No	No
updateBinaryStream(String columnName, InputStream x, int length)	No	No
updateBlob(int columnIndex, Blob x)	No	No
updateBlob(String columnName, Blob x)	No	No
updateBoolean(int columnIndex, boolean x)	Yes	No
updateBoolean(String columnName, boolean x)	Yes	No
updateByte(int columnIndex, byte x)	Yes	No
updateByte(String columnName, byte x)	Yes	No
updateBytes(int columnIndex, byte[] x)	Yes	No
updateBytes(String columnName, byte[] x)	Yes	No


Table 99. IMS JDBC drivers support for the ResultSet interface (continued)

JDBC method	IMS Universal JDBC driver and IMS Universal JCA/JDBC driver support	IMS classic JDBC driver support
updateCharacterStream(int columnIndex, Reader x, int length)	No	No
updateCharacterStream(String columnName, Reader reader, int length)	No	No
updateClob(int columnIndex, Clob x)	No	No
updateClob(String columnName, Clob x)	No	No
updateDate(int columnIndex, Date x)	Yes	No
updateDate(String columnName, Date x)	Yes	No
updateDouble(int columnIndex, double x)	Yes	No
updateDouble(String columnName, double x)	Yes	No
updateFloat(int columnIndex, float x)	Yes	No
updateFloat(String columnName, float x)	Yes	No
updateInt(int columnIndex, int x)	Yes	No
updateInt(String columnName, int x)	Yes	No
updateLong(int columnIndex, long x)	Yes	No
updateLong(String columnName, long x)	Yes	No
updateNull(String columnName)	No	No
updateObject(int columnIndex, Object x)	Yes	No
updateObject(int columnIndex, Object x, int scale)	No	No
updateObject(String columnName, Object x)	Yes	No
updateObject(String columnName, Object x, int scale)	No	No
updateRef(int columnIndex, Ref x)	No	No
updateRef(String columnName, Ref x)	No	No
updateRow()	Yes	No
updateShort(int columnIndex, short x)	Yes	No
updateShort(String columnName, short x)	Yes	No
updateString(int columnIndex, String x)	Yes	No
updateString(String columnName, String x)	Yes	No
updateTime(int columnIndex, Time x)	Yes	No

Table 99. IMS JDBC drivers support for the *ResultSet* interface (continued)

JDBC method	IMS Universal JDBC driver and IMS Universal JCA/JDBC driver support	IMS classic JDBC driver support
<code>updateTime(String columnName, Time x)</code>	Yes	No
<code>updateTimestamp(int columnIndex, Timestamp x)</code>	Yes	No
<code>updateTimestamp(String columnName, Timestamp x)</code>	Yes	No
<code>wasNull()</code>	Yes	No

**Related concepts:**

 [Programming with the IMS Universal JDBC driver \(Application Programming\)](#)

## java.sql.ResultSetMetaData methods supported

A `ResultSetMetaData` object can be used to get information about the types and properties of the columns in a `ResultSet` object.

**Recommendation:** Because the IMS Universal drivers are built on industry standards and open specifications, and provide more flexibility and improved support for connectivity, data access methods, and transaction processing options, use the IMS Universal drivers for developing your Java applications to access IMS.


The following methods are supported by the IMS JDBC drivers for the `ResultSetMetaData` interface.

Table 100. IMS JDBC drivers support for *ResultSetMetaData*.

JDBC method	IMS Universal JDBC driver and IMS Universal JCA/JDBC driver support	IMS classic JDBC driver support
<code>getCatalogName(int column)</code>	Yes	Yes
<code>getColumnClassName(int column)</code>	Yes	No
<code>getColumnCount()</code>	Yes	Yes
<code>getColumnDisplaySize(int column)</code>	Yes	Yes
<code>getColumnLabel(int column)</code>	Yes	Yes
<code>getColumnName(int column)</code>	Yes	Yes
<code>getColumnType(int column)</code>	Yes	Yes
<code>getColumnTypeName(int column)</code>	Yes	Yes
<code>getPrecision(int column)</code>	Yes	Yes
<code>getScale(int column)</code>	Yes	Yes
<code>getSchemaName(int column)</code>	Yes	Yes
<code>getTableName(int column)</code>	Yes	Yes
<code>isAutoIncrement(int column)</code>	Yes	Yes
<code>isCaseSensitive(int column)</code>	Yes	Yes
<code>isCurrency(int column)</code>	Yes	Yes
<code>isDefinitelyWritable(int column)</code>	Yes	Yes
<code>isNullable(int column)</code>	Yes	Yes
<code>isReadOnly(int column)</code>	Yes	Yes
<code>isSearchable(int column)</code>	Yes	Yes
<code>isSigned(int column)</code>	Yes	Yes
<code>isWritable(int column)</code>	Yes	Yes



**Related concepts:**

 Programming with the IMS Universal JDBC driver (Application Programming)

---

## IMS Universal drivers support for the Common Client Interface

The IMS Universal Database resource adapter supports the Common Client Interface (CCI) API in the Java EE Connector Architecture 1.5 specification.

**Related concepts:**

 Programming using the IMS Universal Database resource adapter (Application Programming)

### **javax.resource.cci.Connection methods supported**

The `javax.resource.cci.Connection` interface represents an application-level handle used by client to access the underlying physical connection.

The following table list which methods are supported by the IMS Universal Database resource adapter for the `javax.resource.cci.Connection` interface.

*Table 101. IMS Universal Database resource adapter support for the `javax.resource.cci.Connection` interface.*

<code>javax.resource.cci.Connection</code> method	IMS Universal Database resource adapter support
<code>close()</code>	Yes
<code>createInteraction()</code>	Yes
<code>getLocalTransaction()</code>	Yes
<code>getMetaData()</code>	Yes
<code>getResultSetInfo()</code>	Yes

**Related concepts:**

 Programming using the IMS Universal Database resource adapter (Application Programming)

### **javax.resource.cci.ConnectionFactory methods supported**


The `javax.resource.cci.ConnectionFactory` interface provides an application component with a `Connection` instance to an EIS.

The following table list which methods are supported by the IMS Universal Database resource adapter for the `javax.resource.cci.ConnectionFactory` interface.

*Table 102. IMS Universal Database resource adapter support for the `javax.resource.cci.ConnectionFactory` interface.*

<code>javax.resource.cci.ConnectionFactory</code> method	IMS Universal Database resource adapter support
<code>getConnection()</code>	Yes
<code>getConnection(ConnectionSpec)</code>	Yes
<code>getMetaData()</code>	Yes
<code>getRecordFactory()</code>	Yes

**Related concepts:**

 Programming using the IMS Universal Database resource adapter (Application Programming)

## javax.resource.cci.ConnectionMetaData methods supported


The javax.resource.cci.ConnectionMetaData interface provides information about an EIS instance connected through a Connection instance.

The following table list which methods are supported by the IMS Universal Database resource adapter for the javax.resource.cci.ConnectionMetaData interface.

Table 103. IMS Universal Database resource adapter support for the javax.resource.cci.ConnectionMetaData interface.

javax.resource.cci.ConnectionMetaData method	IMS Universal Database resource adapter support
getEISProductName()	Yes
getEISProductVersion()	Yes
getUserName()	Yes

### Related concepts:

 Programming using the IMS Universal Database resource adapter (Application Programming)

## javax.resource.cci.Interaction methods supported

The javax.resource.cci.Interaction interface provides a means for an application component to execute EIS functions, such as relational database queries.

The following table list which methods are supported by the IMS Universal Database resource adapter for the javax.resource.cci.Interaction interface.

Table 104. IMS Universal Database resource adapter support for the javax.resource.cci.Interaction interface.

javax.resource.cci.Interaction method	IMS Universal Database resource adapter support
clearWarnings()	Yes
close()	Yes
execute(InteractionSpec, Record)	Yes
execute(InteractionSpec, Record, Record)	No
getConnection()	Yes
getWarnings()	Yes

### Related concepts:

 Programming using the IMS Universal Database resource adapter (Application Programming)

## javax.resource.cci.LocalTransaction methods supported


The javax.resource.cci.LocalTransaction interface defines a transaction demarcation interface for resource manager local transactions.

The following table list which methods are supported by the IMS Universal Database resource adapter for the javax.resource.cci.LocalTransaction interface.

Table 105. IMS Universal Database resource adapter support for the javax.resource.cci.LocalTransaction interface.

javax.resource.cci.LocalTransaction method	IMS Universal Database resource adapter support
begin()	Yes
commit()	Yes
rollback()	Yes

**Related concepts:**

 Programming using the IMS Universal Database resource adapter (Application Programming)

## **javax.resource.cci.ResultSetInfo methods supported**


The interface `javax.resource.cci.ResultSetInfo` provides information on the support provided for `ResultSet` by a connected EIS instance.

The following table list which methods are supported by the IMS Universal Database resource adapter for the `javax.resource.cci.ResultSetInfo` interface.

*Table 106. IMS Universal Database resource adapter support for the `javax.resource.cci.ResultSetInfo` interface.*

<b>javax.resource.cci.ResultSetInfo method</b>	<b>IMS Universal Database resource adapter support</b>
<code>deletesAreDetected(int)</code>	Yes
<code>insertsAreDetected(int)</code>	Yes
<code>othersDeletesAreVisible(int)</code>	Yes
<code>othersInsertsAreVisible(int)</code>	Yes
<code>othersUpdatesAreVisible(int)</code>	Yes
<code>ownDeletesAreVisible(int)</code>	Yes
<code>ownInsertsAreVisible(int)</code>	Yes
<code>ownUpdatesAreVisible(int)</code>	Yes
<code>supportsResultSetType(int)</code>	Yes
<code>supportsResultTypeConcurrency(int, int)</code>	Yes
<code>updatesAreDetected(int)</code>	Yes

**Related concepts:**

 Programming using the IMS Universal Database resource adapter (Application Programming)

## **javax.resource.cci.ResourceAdapterMetaData methods supported**

The interface `javax.resource.cci.ResourceAdapterMetaData` provides information about capabilities of a resource adapter implementation.

The following table list which methods are supported by the IMS Universal Database resource adapter for the `javax.resource.cci.ResourceAdapterMetaData` interface.

*Table 107. IMS Universal Database resource adapter support for the `javax.resource.cci.ResourceAdapterMetaData` interface.*

<b>javax.resource.cci.ResourceAdapterMetaData method</b>	<b>IMS Universal Database resource adapter support</b>
<code>getAdapterName()</code>	Yes
<code>getAdapterShortDescription()</code>	Yes
<code>getAdapterVendorName()</code>	Yes
<code>getAdapterVersion()</code>	Yes
<code>getInteractionSpecsSupported()</code>	Yes

Table 107. IMS Universal Database resource adapter support for the `javax.resource.cci.ResourceAdapterMetaData` interface. (continued)

<code>javax.resource.cci.ResourceAdapterMetaData</code> method	IMS Universal Database resource adapter support
<code>getSpecVersion()</code>	Yes
<code>supportsExecuteWithInputAndOutputRecord()</code>	Yes
<code>supportsExecuteWithInputRecordOnly()</code>	Yes
<code>supportsLocalTransactionDemarcation()</code>	Yes

**Related concepts:**

 Programming using the IMS Universal Database resource adapter (Application Programming)

## `javax.resource.cci.RecordFactory` methods supported


The `javax.resource.cci.RecordFactory` interface provides an application component with a `Record` instance.

The following table list which methods are supported by the IMS Universal Database resource adapter for the `javax.resource.cci.RecordFactory` interface.

Table 108. IMS Universal Database resource adapter support for the `javax.resource.cci.RecordFactory` interface.

<code>javax.resource.cci.RecordFactory</code> method	IMS Universal Database resource adapter support
<code>createIndexedRecord(String)</code>	Yes
<code>createMappedRecord(String)</code>	Yes

**Related concepts:**

 Programming using the IMS Universal Database resource adapter (Application Programming)

## Java API documentation (Javadoc)

These topics contain the Java API documentation (Javadoc) for the IMS solutions for Java development.

- Javadoc for IMS Universal drivers and IMS Java dependent region resource adapter
- Javadoc for Classic Java APIs
- Javadoc for IMS TM resource adapter

### IMS Universal drivers and IMS Java dependent region resource adapter

Because IMS Java dependent region (JDR) resource adapter reuses some of the interfaces or classes in the IMS Universal drivers, the classes and interfaces are packaged together as one .jar file, `imsudb.jar`.

**Important:** IMS 13 APAR PI62580 (PTF UI39459) is required.

The following packages provide Java interfaces or classes for interacting with IMS JBPs, JMPs or IMS database resources.

Table 109. Packages for IMS database access for IMS Universal drivers and IMS Java dependent region resource adapter

Package	Description	Used by
com.ibm.ims.application	Provides classes for IMS Java dependent region transaction and message processing. It contains classes for managing error messages, sending and receiving messages providing program access to IMS transaction services such as commit and rollback, and more.	IMS Java dependent region resource adapter (imsudb.jar)
com.ibm.ims.base	Provides classes for mapping Java calls to DL/I APIs.	IMS Java dependent region resource adapter (imsudb.jar)
com.ibm.ims.db.cci	Provides classes for interacting IMS database resources using the Common Client Interface (CCI) architecture.	<ul style="list-style-type: none"> <li>IMS Universal Database resource adapter with local transaction support (imsudbLocal.rar)</li> <li>IMS Universal Database resource adapter with additional two-phase commit processing (imsudbXA.rar)</li> </ul>
com.ibm.ims.dli	Provides an API to write Java applications that can access IMS databases using DL/I programming semantics.	imsudb.jar <ul style="list-style-type: none"> <li>IMS Universal DL/I driver</li> <li>IMS Universal JDBC driver</li> <li>IMS Java dependent region resource adapter</li> </ul>
com.ibm.ims.dli.converters	Provides an API to convert Java data types to and from byte arrays.	imsudb.jar <ul style="list-style-type: none"> <li>IMS Universal DL/I driver</li> <li>IMS Universal JDBC driver</li> <li>IMS Java dependent region resource adapter</li> </ul>
com.ibm.ims.dli.tm	Provides a Java interface to interact with IMS Java batch processing regions (JBPs) and Java message processing regions (JMPs).	IMS Java dependent region resource adapter (imsudb.jar)
com.ibm.ims.dli.types	Provides an extensible, abstract type converter to assist in creating custom user type converters.	imsudb.jar <ul style="list-style-type: none"> <li>IMS Universal DL/I driver</li> <li>IMS Universal JDBC driver</li> <li>IMS Java dependent region resource adapter</li> </ul>

Table 109. Packages for IMS database access for IMS Universal drivers and IMS Java dependent region resource adapter (continued)

Package	Description	Used by
com.ibm.ims.jdbc	Provides IMS-specific extensions for connecting to IMS databases using JDBC.	<ul style="list-style-type: none"> <li>IMS Universal JDBC driver (imsudb.jar)</li> <li>IMS Universal JDBC driver with JCA support (imsudbJLocal.rar or imsudbJXA.rar)</li> </ul>
com.ibm.ims.jdbc.xa	Provides IMS-specific extensions for connecting to IMS databases in two-phase commit (XA) mode using JDBC.	IMS Universal JDBC driver with JCA support (imsudbJXA.rar)
com.ibm.ims.jms	Includes IMS-specific extensions for issuing synchronous callout requests from JMP and JBP regions.	IMS Java dependent region resource adapter (imsudb.jar)

## Javadoc for IMS Classic Java APIs

The following packages provide Java interfaces or classes for interacting with IMS JBPs or JMPs, sending and receiving messages, providing program access to IMS transaction services such as commit and rollback, and mapping Java calls to the DL/I APIs for database operations.

IMS Version 13 is the last release to support the IMS Classic Java™ APIs. Customers that are using these APIs should migrate to the IMS Universal drivers.

Table 110. Packages in IMS Classic Java APIs

Package	Description
com.ibm.connector2.ims.db	Provides classes for connection to IMS databases from WebSphere Application Server for z/OS.
com.ibm.ims.application	Provides classes for IMS Java dependent region transaction and message processing. It contains classes for managing error messages, sending and receiving messages providing program access to IMS transaction services such as commit and rollback, and more.
com.ibm.ims.base	Provides classes for mapping Java calls to DL/I APIs.
com.ibm.ims.db	Provides classes for the JDBC driver and for the IMS Java hierarchic database interface.
com.ibm.ims.util	Provides general utility classes for the IMS Classic Java APIs.


## Javadoc for IMS TM resource adapter

The IMS TM resource adapter Java API documentation is provided separately at IMS TM resource adapter Java API reference.

Table 111. Javadoc for IMS TM resource adapter

Package	Description	Used by
com.ibm.connector2.ims.ico	Includes interfaces and classes for extensions for managing the connections to IMS and interactions with IMS transactions.	IMS TM resource adapter (imsicoxxx.rar, where xxx is the version number such as 1322 for V13.2.2)
com.ibm.connector2.ims.ico.inbound	Includes the class for configuring properties for inbound communication from IMS.	IMS TM resource adapter (imsicoxxx.rar, where xxx is the version number such as 1322 for V13.2.2)

**Related concepts:**

-  [IMS solutions for Java development overview \(Application Programming\)](#)
- [Java API specifications for Universal drivers, JDR resource adapter, and JMS](#)
- [Java API specification for the classic Java APIs for IMS](#)





---

## Chapter 5. Message Format Service (MFS) reference

These topics contain reference information for the IMS Message Format Service (MFS).

---

### 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.

### 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.

The following figure shows the interrelationships between MFS control blocks. The subsequent illustrate the four levels of linkages, which are then summarized in the final figure.

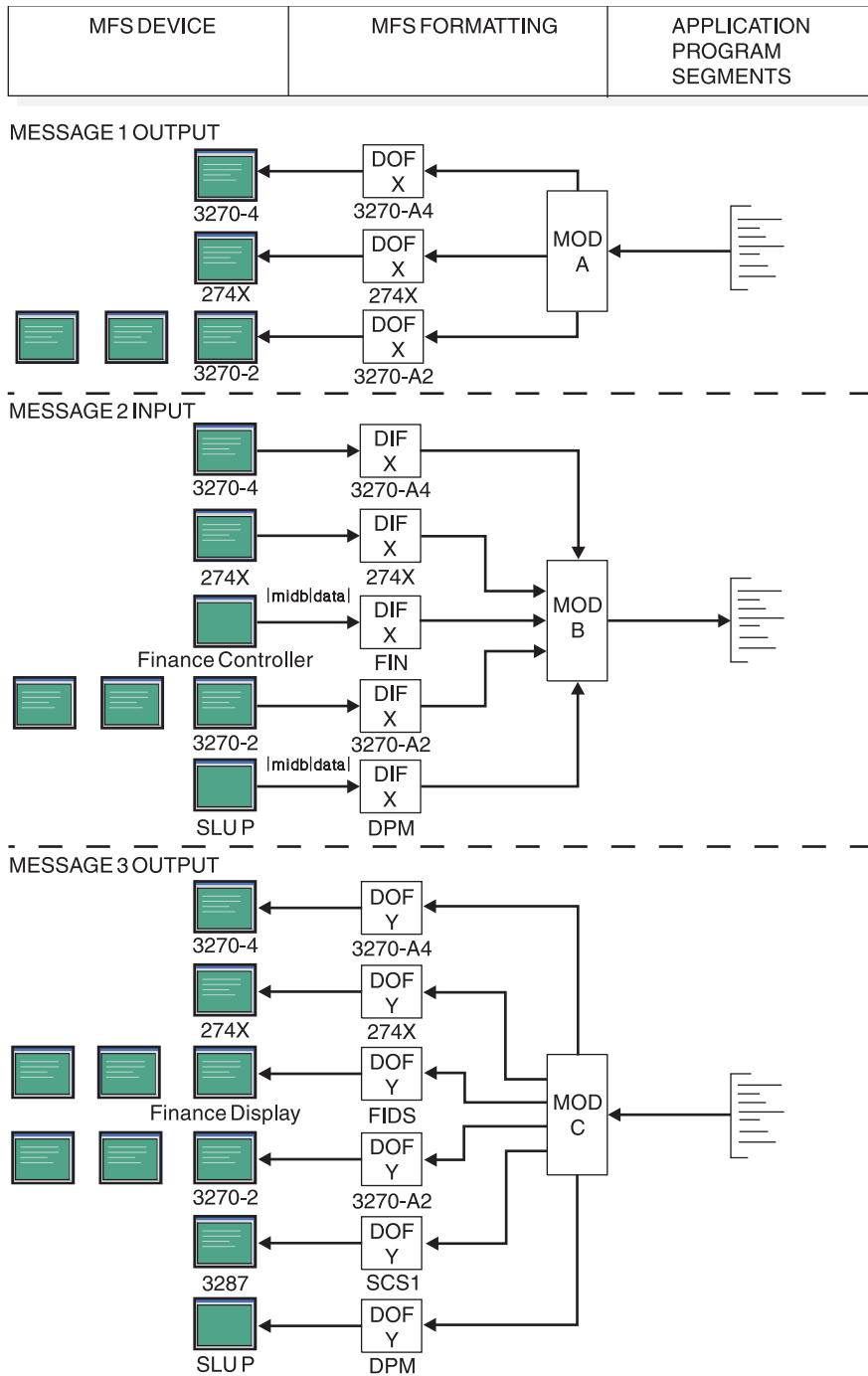


Figure 22. Control block interrelationships

The following figure shows the highest-level linkage, that of chained control blocks.

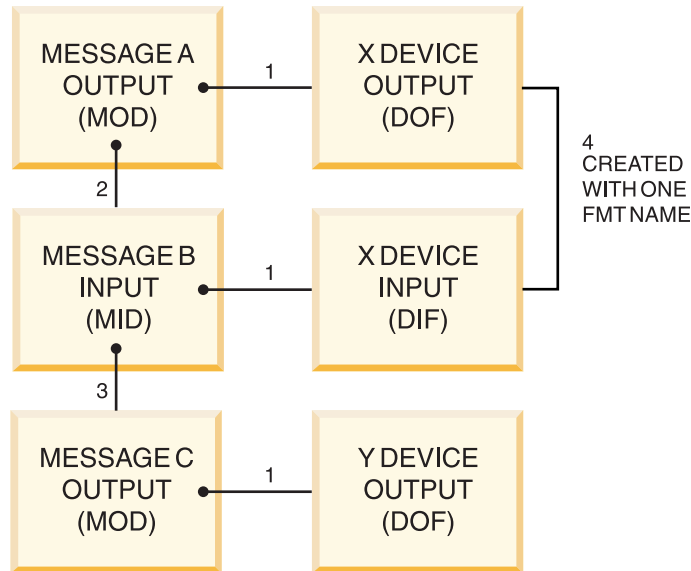


Figure 23. Chained control block linkage

**Notes to the previous figure:**

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 by way of 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.

The following figure 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.

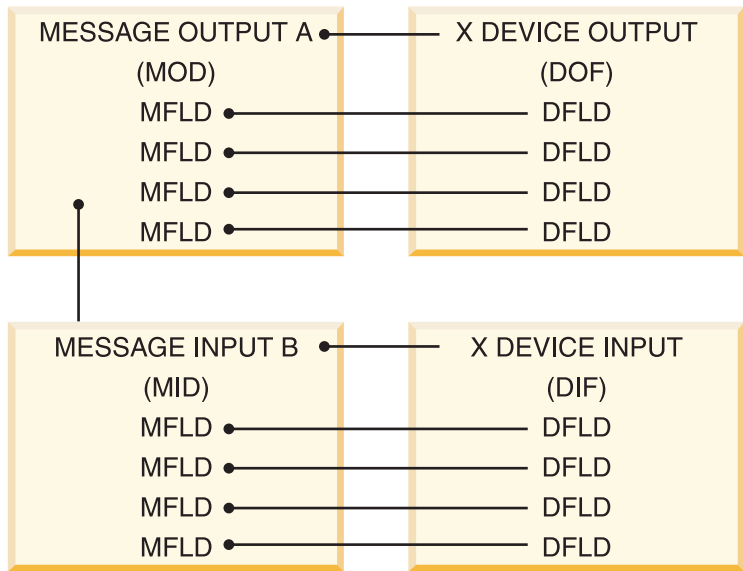


Figure 24. Linkage between message fields and device fields

The following figure shows a third level of linkage, which exists between the LPAGE and the DPAGE.

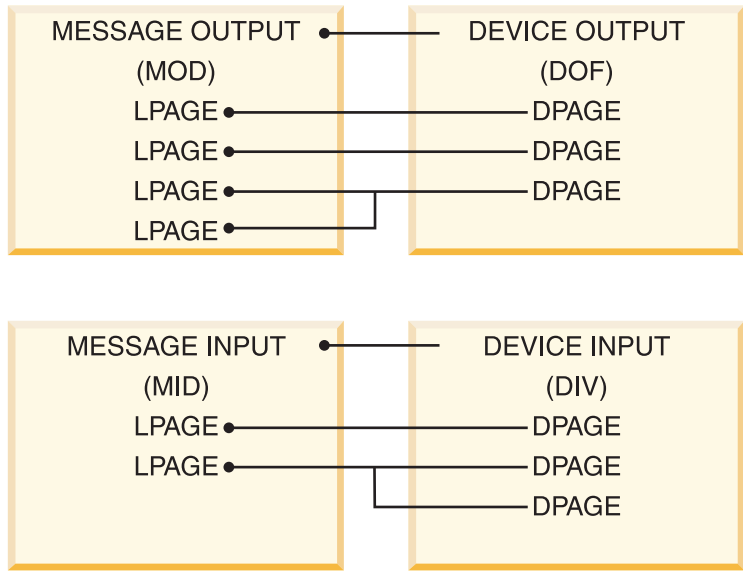


Figure 25. LPAGE and DPAGE relationships

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.

The following figure 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.

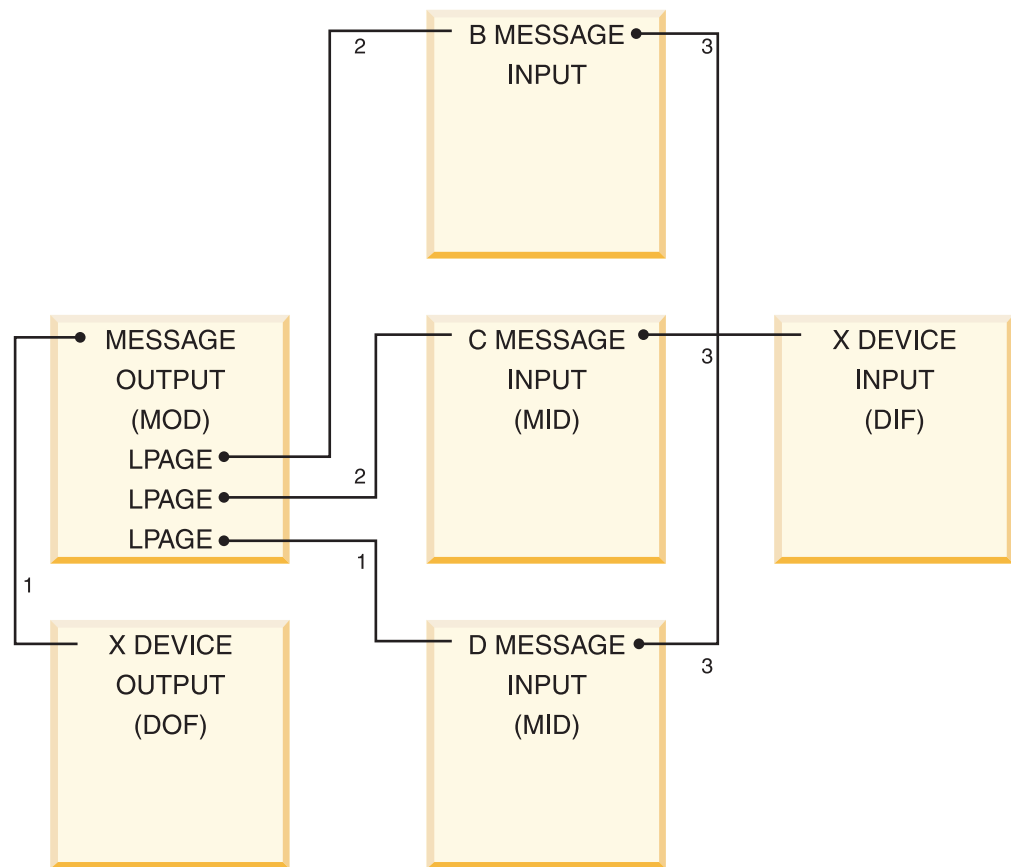


Figure 26. Optional message descriptor linkage

**Notes to the previous figure:**

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.

The following figure summarizes the previously explained MFS control block linkages.

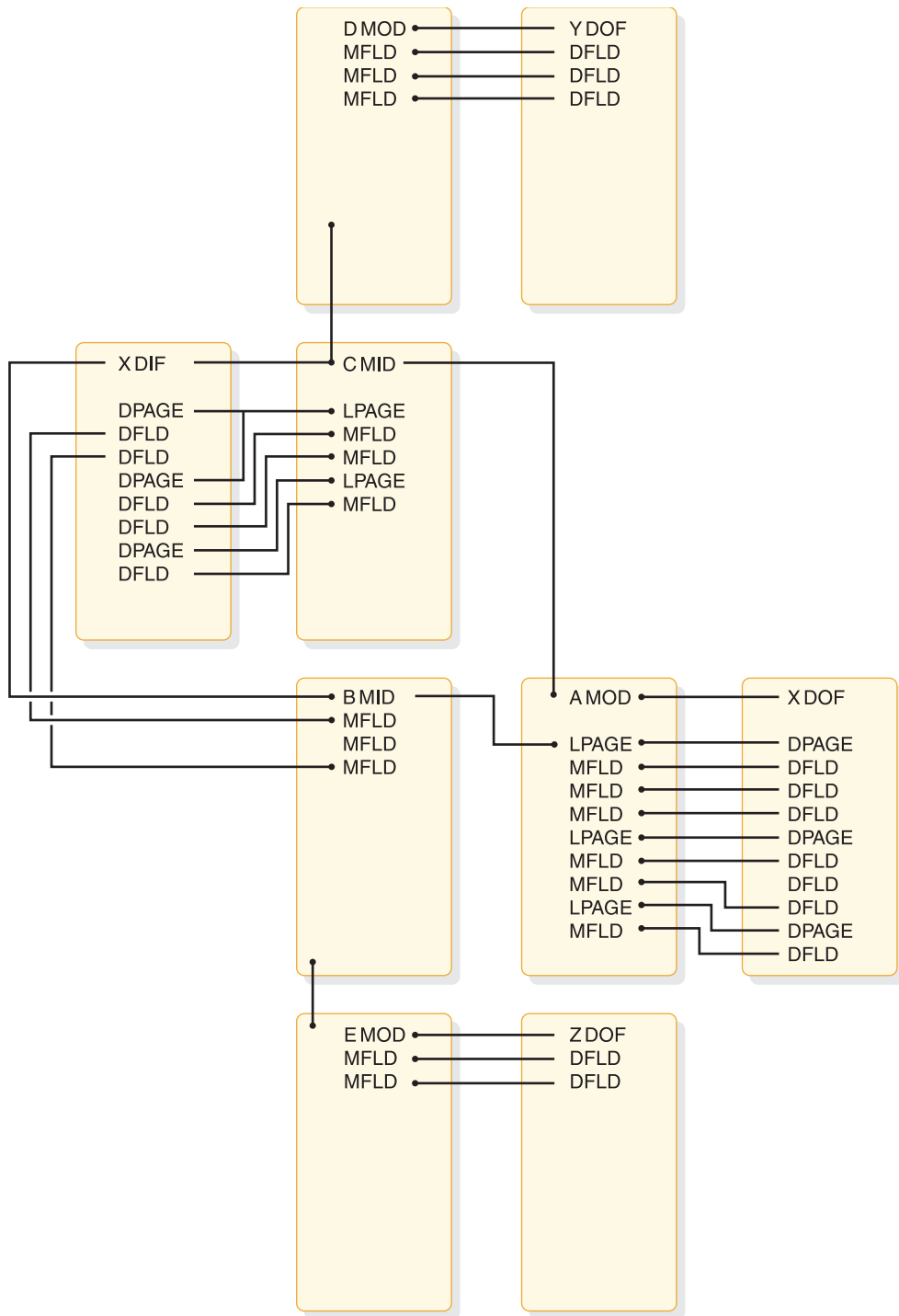


Figure 27. Summary of control block linkages

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 through 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.

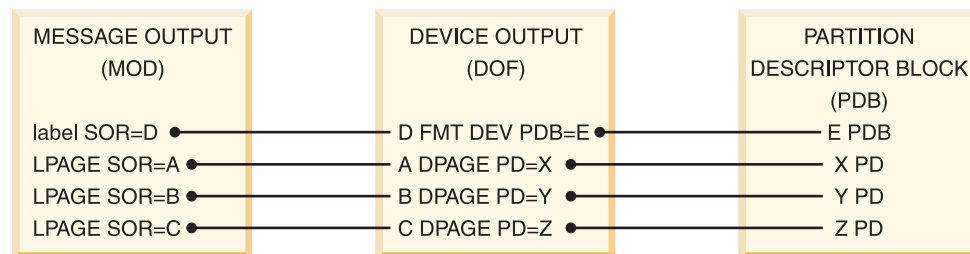


Figure 28. Linkages in partitioned format mode

### Finance, 3770, SLU 1, NTO, or SLU P devices

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 the following table.
- 2 Device feature indicator (hexadecimal). For a list of indicators by feature, see the subsequent table.
- 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 listed in the following table.

*Table 112. Device type indicators for FMT=*

Device Feature	Indicator (Hex)
SLU 2, Model 1 display	00
3284-1 or 3286-1 printer	01
3277, or SLU 2, Model 2 display	02
3284-2 or 3286-2 printer	03
3604-1 or 2 (FIDS)	05
3604-3 (FIDS3)	06
3604-4 (FIDS4)	07
3600 (FIN)	08
3610, 3612 journal printer (FIJP)	09



Table 112. Device type indicators for FMT= (continued)

Device Feature	Indicator (Hex)
3611, 3612 passbook printer (FIPB)	0A
3618 administrative printer (FIFP)	0B
SCS1: 3770; NTO; and SLU 1 (print data set)	0C
SCS2: 3521 card punch, 3501 card reader, 2502 card reader, and SLU 1 (transmit data set),	0D
3604-7 (FIDS7)	0E
DPM-A1 through DPM-A15, respectively	11 through 1F
DPM-B1 through DPM-B15, respectively	21 through 2F
3270-A1 through 3270-A15, respectively	41 through 4F

**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.

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.

For 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 the following table for byte 2 of the DEV FEAT= specification:

*Table 113. Example of device feature indicator values*

<b>Device Feature</b>	<b>Indicator (Hex)</b>
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

Table 113. Example of device feature indicator values (continued)

Device Feature	Indicator (Hex)
DUAL (dual platen)	C1
P.L. 132,DUAL	61
No features (3270)	40
3270,3270P,3770,SLU 1, SLU 2, 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, because 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. The screen is cleared and the next output is a full format operation.
- The operator uses the operator identification card reader. The screen is cleared and the next output is a full format operation.

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 non-literal 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
  - 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 non-literal 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. Pre-modified attributes can be requested by the application program to ensure input of field data. If pre-modified 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:
  - 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.

**Recommendations:** 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 message format buffer 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 non-literal 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.

**Related concepts:**

“System control area (SCA) and default SCA (DSCA)” on page 509

### **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 SYSMMSG 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 SLU 2 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 using 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.

**Related concepts:**

“3290 in partitioned format mode” on page 543

### **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, the 3180 customer set up installation instructions can be used and no special IMS code is necessary.

## **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 the following table include the recommended standard for relating the device type symbolic name to the screen size:

Table 114. MFS device definition compatibility for 3270 devices.

Device and Screen Size	Device and Screen Size <sup>1</sup>	New IMS System Definition <sup>1</sup>
3275 or 3277 (12X40)	MFS: DEV TYPE= (3270,1) Model 1	MFS: DEV TYPE= 3270-A5 <sup>2,4</sup>
3275, 3276, 3277, 3278 (24X80)	MFS: DEV TYPE= (3270,2) Model 2	MFS: DEV TYPE= 3270-A2 <sup>2,4</sup>
3276, 3278 (12X80)	MFS: DEV TYPE= (3270,1) Model 1	MFS: DEV TYPE= 3270-A1 <sup>2,3</sup>
3276, 3278 (32X80)	MFS: DEV TYPE= (3270,2) Model 2	MFS: DEV TYPE= 3270-A3 <sup>2,3</sup>
3276, 3278 (43X80)	MFS: DEV TYPE= (3270,2) Model 2	MFS: DEV TYPE= 3270-A4 <sup>2,3</sup>
3278 (27X132)	MFS: DEV TYPE= (3270,2) Model 2	MFS: DEV TYPE= 3270-A7 <sup>2,3</sup>

**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 the following table.
4. See option 4 in the following table.

The following table 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 115. 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 the previous table.)
2	You can use full screen.	You must design new device formats.	No (Define new formats.)
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 the previous table.)
4	Consistency in definition for current and new screen sizes.	You must modify all formats.	Yes (Refer to the previous table.)

### 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.

## Converting MFS 3270 device formats to symbolic name formats using STACK/UNSTACK

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.

For 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 (using 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

This example is provided to clarify MFS device definition compatibility for 3270 devices.

Assume that the installation has 3270 model 1 and model 2 display devices and has installed additional display devices with 12x80, 24x80, 32x80, and 43x80 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:

```

label    FMT
          DEV      TYPE=3270-A5 (changed from (3270,1) to 3270
                                display with 12x40 screen size)
          STACK   ON
          DIV      TYPE=INOUT

```

```

label 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
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 the following table. The following table lists the current device, the MFS device type, new control units, system definitions, and MFS device types, and the z/OS changes required.

*Table 116. 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	z/OS Changes Required
3284/ 3286	3270P	3827/3289 attached to a 3274 or 3276 SNA control unit	SLUTYPE1	SCS1	See Note

**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.

## Enhancing system performance of MFS message and device formats

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.

### Enhancing system performance of MFS-supported devices

To enhance system performance when using MFS-supported devices, you can do the following.

### Evaluating the message format buffer pool operation

The IMS /DISPLAY POOL command can be used to evaluate the message format buffer 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.)

### Hints and tips for improving performance

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 MFBP (Message Format 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.
- 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. If duplicate libraries exist in the concatenated libraries, there is no guarantee that the copy from the first library will be fetched.
- 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.

**Related reference:**

“Input message formatting options” on page 484

 /DISPLAY POOL command (Commands)

**Enhancing system performance of 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. 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**

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 concepts:**

“3270 or SLU 2 screen formatting” on page 429

#### **Related tasks:**

 Extended Terminal Option (ETO) (Communications and Connections)

### **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 programmed symbol buffers:**

If the operator knows the programmed symbol buffers need to be loaded (because the device was just turned on), 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.

For 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).

## **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. The following samples show the MFS definition format.

```

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'
        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),
        PRN=DFLDINP3,
        RDPN=DFLDINP4,
        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),
        NXT=MFSMID1
        SEG
        MFLD  DFLDOUT1,LTH=5
        MFLD  DFLDOUT2,LTH=100
        MSGEND
MFSMODE2 MSG  TYPE=OUTPUT,SOR=(FMTDPM,IGNORE),
        NXT=MFSMID1
        SEG
        MFLD  DFLDOUT3,LTH=5
        MFLD  DFLDOUT4,LTH=100
        MFLD  (,SCA),LTH=2
        MSGEND
MFSMID1 MSG  TYPE=INPUT,SOR=(FMTDPM,IGNORE),
        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),
        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
```

---

## MFS message formats

Use these topics if your application programs communicate with devices by using Message Format Service (MFS).

### 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.

### 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.

When LPAGEs are defined, each LPAGE is related to one or more DPAGEs.

#### Related reference:

“Input message formatting options” on page 484

### Device-dependent input information (3270 or SLU 2)

Using certain options for inputting information can make the application program device-dependent.

### 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 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.

The table below lists the maximum line and column values for MFS device types.

Table 117. Maximum line and column values for 3270 device types

MFS Device Type	Maximum Value	
	Line	Column
3270,1	12	40
3270,2	24	80
3270-An		
	12	40
SIZE=(12,40)		
	12	80
SIZE=(12,80)		
	24	80
SIZE=(24,80)		
	32	80
SIZE=(32,80)		
	43	80
SIZE=(43,80)		
	27	132
SIZE=(27,132)		
	62	160
SIZE=(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 programs.

### **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 that apply for messages with one LPAGE apply. Those rules are:

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. 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 that defines the length of the segment and, if required, specifies 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. The following table illustrates the format of an output segment.

*Table 118. Format of an output segment*

LL	Z1	Z2	SN	FIELDs
----	----	----	----	--------

Where:

**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.

**Restriction:** 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 Version 13 Communications and Connections*.

**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.

The following example shows how to code a null character in COBOL.

### **Coding a null character in COBOL**

```
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).

```

### 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.

#### Related concepts:

“Output message formatting options” on page 506

### 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, as shown in the following table.

*Table 119. Format of an option 3 output segment*

FL	FO	DATA
----	----	------

Where:

- 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.

**Related concepts:**

“Output message formatting options” on page 506

**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 these tables:

Usage notes follow both tables.

*Table 120. 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.

Table 120. Valid bytes and bits for TYPE=3270, SLU 2, DPM-An, or DPM-Bn (continued)

Byte	Bit	Description
<b>Notes:</b>		
		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.
		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. Any invalid bits for the device type are reset. The valid bits are used.

Table 121. 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.

**Notes:**

- 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.
- For TYPE=SCS1, or SCS2, the SCA parameter is ignored.
- 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.

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. The following table lists the valid line and column values.

*Table 122. Maximum line and column values for MFS device types*

MFS Device Type	Maximum Value	
	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
SIZE=(27,132) (3564 characters)	27	132
SIZE=(62,160) (9920 characters)	62	160

**Related concepts:**

“3290 in partitioned format mode” on page 543

“Output format control for ISC (DPM-Bn) subsystems” on page 529

**Related reference:**

“Input message formatting options” on page 484

“Dynamic attribute modification”

**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 the table below.

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:

- 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 the following table.

*Table 123. 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	<ol style="list-style-type: none"> <li>1. If on, these attribute specifications are to replace the attribute byte defined for the field.</li> <li>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.)</li> </ol>
	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.

**Related reference:**

“Device-dependent output information” on page 452

**Dynamic modification of extended field attributes**

For an application program to modify extended attribute data, the MFLD statement must specify ATTR=*nm*. 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.

The following table shows the format of the extended attribute modification bytes.

*Table 124. 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:

- | 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 the following code example.

```

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)
*

```

Ways to specify field outlining attributes, input control types, and values in COBOL are shown in the following code example.

```

01 BINVALUE.
02 VAL0000 PIC S999 COMP VALUE +0.
02 VAL0000X REDEFINES VAL0000.
03 FILLER PIC X.
03 VAL00 PIC X.
* (NO FIELD OUTLINE)
02 VAL0001 PIC S999 COMP VALUE +1.
02 VAL0001X REDEFINES VAL0001.
03 FILLER PIC X.
03 VAL01 PIC X.
* (UNDERLINE)
02 VAL0002 PIC S999 COMP VALUE +2.
02 VAL0002X REDEFINES VAL0002.
03 FILLER PIC X.
03 VAL02 PIC X.
* (RIGHTLINE)
02 VAL0003 PIC S999 COMP VALUE +3.
02 VAL0003X REDEFINES VAL0003.
03 FILLER PIC X.
03 VAL03 PIC X.
* (RIGHTLINE & UNDERLINE)
02 VAL0004 PIC S999 COMP VALUE +4.
02 VAL0004X REDEFINES VAL0004.
03 FILLER PIC X.
03 VAL04 PIC X.
* (OVERLINE)
02 VAL0005 PIC S999 COMP VALUE +5.
02 VAL0005X REDEFINES VAL0005.
03 FILLER PIC X.
03 VAL05 PIC X.
* (OVERLINE & UNDERLINE)

```



```

02 VAL0006          PIC S999 COMP VALUE +6.
02 VAL0006X REDEFINES VAL0006.
   03 FILLER        PIC X.
   03 VAL06         PIC X.
*
*                   (OVERLINE & RIGHTLINE)
02 VAL0007          PIC S999 COMP VALUE +7.
02 VAL0007X REDEFINES VAL0007.
   03 FILLER        PIC X.
   03 VAL07         PIC X.
*
*                   (OVERLINE & RIGHTLINE
*                   & UNDERLINE)
02 VAL0008          PIC S999 COMP VALUE +8.
02 VAL0008X REDEFINES VAL0008.
   03 FILLER        PIC X.
   03 VAL08         PIC X.
*
*                   (LEFTLINE)
02 VAL0009          PIC S999 COMP VALUE +9.
02 VAL0009X REDEFINES VAL0009.
   03 FILLER        PIC X.
   03 VAL09         PIC X.
*
*                   (LEFTLINE & UNDERLINE)
02 VAL000A          PIC S999 COMP VALUE +10.
02 VAL000AX REDEFINES VAL000A.
   03 FILLER        PIC X.
   03 VAL0A         PIC X.
*
*                   (LEFTLINE & RIGHTLINE)
02 VAL000B          PIC S999 COMP VALUE +11.
02 VAL000BX REDEFINES VAL000B.
   03 FILLER        PIC X.
   03 VAL0B         PIC X.
*
*                   (LEFTLINE & RIGHTLINE
*                   & UNDERLINE)
02 VAL000C          PIC S999 COMP VALUE +12.
02 VAL000CX REDEFINES VAL000C.
   03 FILLER        PIC X.
   03 VAL0C         PIC X.
*
*                   (LEFTLINE & OVERLINE)
02 VAL000D          PIC S999 COMP VALUE +13.
02 VAL000DX REDEFINES VAL000D.
   03 FILLER        PIC X.
   03 VAL0D         PIC X.
*
*                   (LEFTLINE & OVERLINE
*                   & UNDERLINE)
02 VAL000E          PIC S999 COMP VALUE +14.
02 VAL000EX REDEFINES VAL000E.
   03 FILLER        PIC X.
   03 VAL0E         PIC X.
*
*                   (LEFTLINE & OVERLINE
*                   & RIGHTLINE)
02 VAL000F          PIC S999 COMP VALUE +15.
02 VAL000FX REDEFINES VAL000F.
   03 FILLER        PIC X.
   03 VAL0F         PIC X.
*
*                   (BOX)

```

### 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 extended attribute types and values shown in the following table must be placed in the field referenced by the MFLD "AY" in the preceding example.

*Table 125. 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 the following table.

*Table 126. 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.

**Related reference:**

“Dynamic modification of DBCS/EBCDIC mixed data” on page 462

**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.

The following table illustrates what the 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 127. 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' <sup>1</sup>	Send X'40_FE'	Send X'40_FE'	Send no attribute
Default X'00' <sup>1</sup>	Send X'00'	Send X'00'	Send no attribute

Table 127. Attribute type value byte contents (continued)

Application Program Programmed Symbol Attribute Bytes of X and:	C3 EATTR=	ATTR=	EATTR=
Not specified <sup>2</sup>	Send programmed symbol DFLD specification	Send no attribute	N/A
Omitted or Invalid <sup>3</sup>	Send programmed symbol DFLD specification	Send X'00'	Send no attribute

**Notes:**

1. ATTR=*nm* 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=*nm* is not specified on any MFLD statement that maps to this DFLD statement.
3. ATTR=*nm* 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 the following figure.

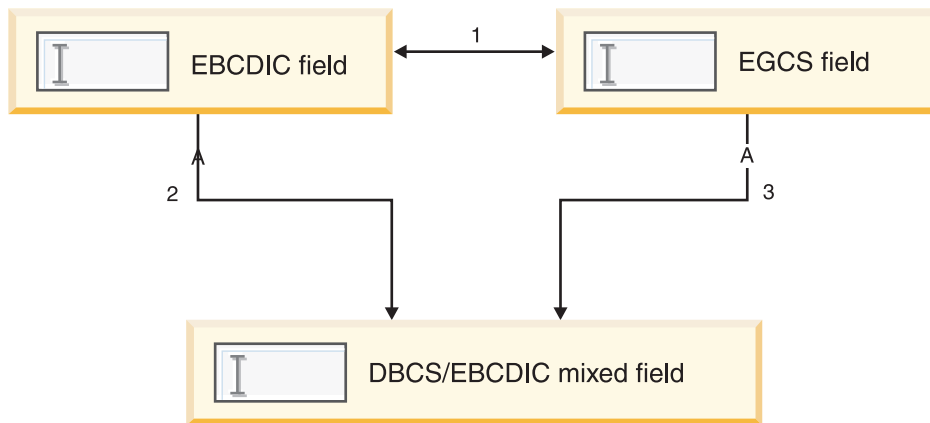


Figure 29. Dynamic modification of a DBCS/EBCDIC mixed field

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  $2 \times nn$  additional bytes preceding the data field.

When  $nn=2$ , the initial attribute is changed as shown in the following table according to the value of the two attribute byte sets (4 bytes) specified in front of the data field by the application program.

Table 128. 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.

**Related reference:**

“Dynamic modification of extended field attributes” on page 456

**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 TP PCB or alternate response PCB, IMS updates the MESSAGE OUTPUT DESCRIPTOR NAME field of the TP 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. 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 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. 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 hexadecimal EBCDIC command codes for use with the 3271/3274 controllers are:

**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.

The two forms of input are:

- 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.

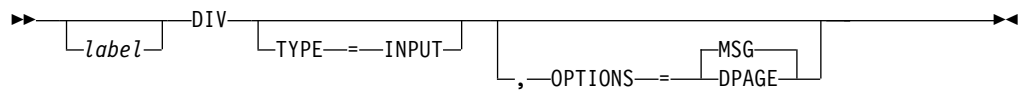
For input with DFS.EDT or DFS.EDTN, the first AID byte, X'88', causes the proper decoding of the second AID byte. Depending on the second AID byte, one of the following occurs:

- 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.

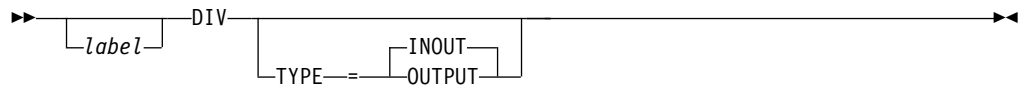
**DIV statement:**

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=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.

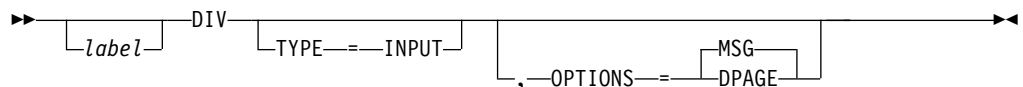
**Format for DEV TYPE=SCS1, or SCS2 and DIV TYPE=INPUT**



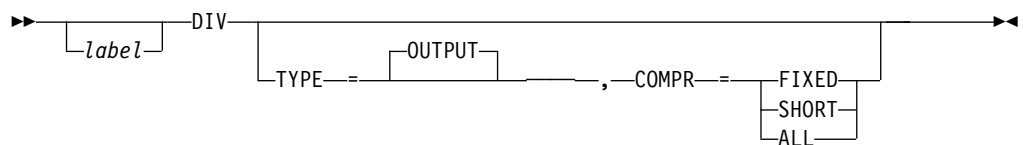
**Format for DEV TYPE=3270 or 3270-An**



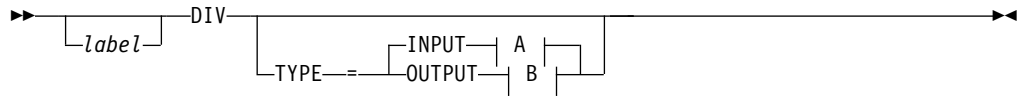
**Format for DEV TYPE=FIN**



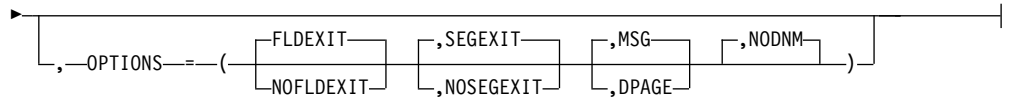
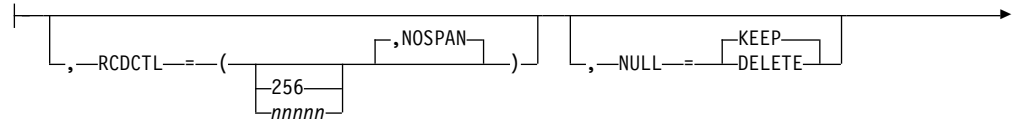
**Format for DEV TYPE=SCS1, SCS2, 3270P, FIDS, FIDS3, FIDS4, FIDS7, FIJP, FIPB, or FIFP and DIV TYPE=OUTPUT**



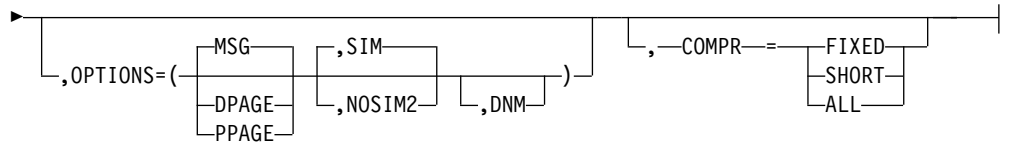
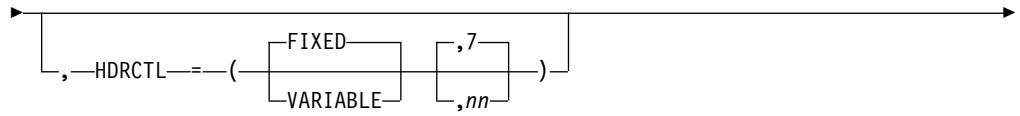
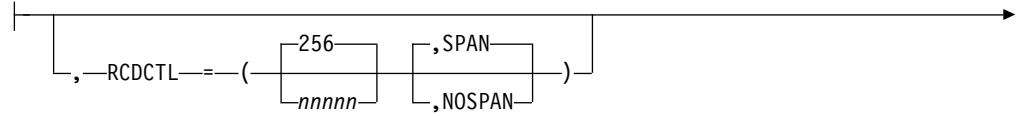
### Format for DEV TYPE=DPM-An



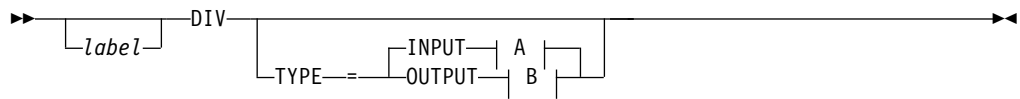
#### A:



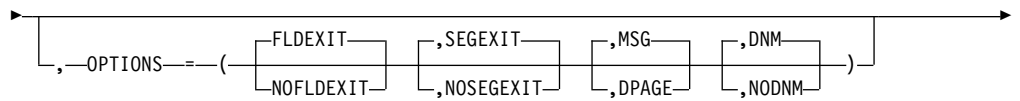
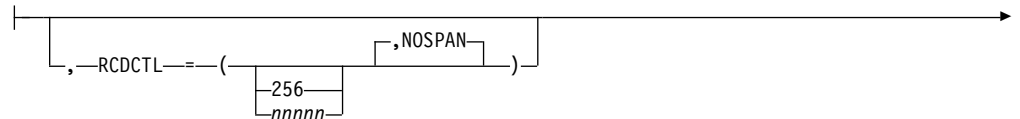
#### B:

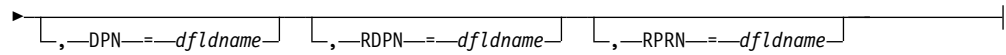


### Format for DEV TYPE=DPM-Bn

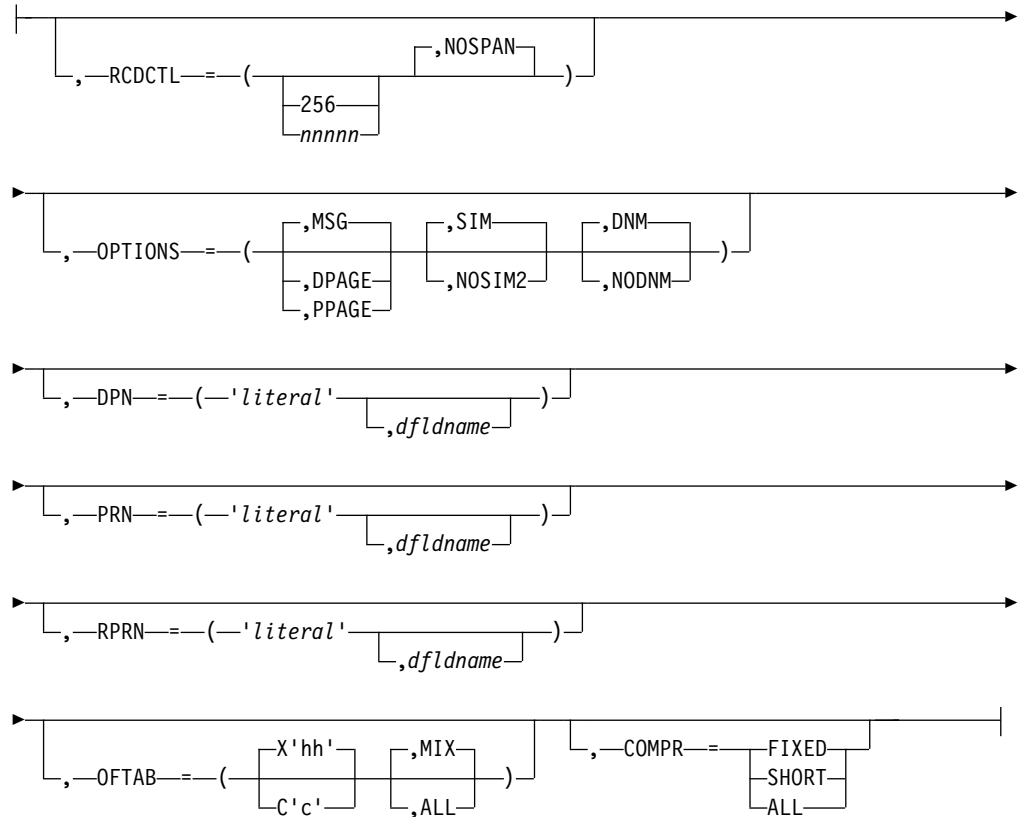


#### A:





**<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.

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.

**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 topic "Message Processing" in *IMS Version 13 Application Programming*. 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 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=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.

For more information on the ATTR parameter on the MFLD statement, see MFS Language utility (DFSUPAA0) (System Utilities).

#### **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 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. 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 *dflename* specification permits the suggested return destination process name (RDPN) to be supplied in the input message MFLD referencing this *dflename*. If *dflename* 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 *dflename* is also specified, the data supplied in the MFLD referencing this *dflename* is used as the DPN in the output ATTACH message header. If no output message MFLD reference to the *dflename* exists, *literal* is used. If the data in the MFLD referencing the *dflename* is greater than eight characters, the first eight characters are used.

**PRN=**

For DIV TYPE=INPUT, the *dflename* specification permits the suggested primary resource name (PRN) to be supplied in the input message MFLD referencing this *dflename*. If the *dflename* 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 *dflename* is also specified, the data supplied in the MFLD referencing this *dflename* is used as the PRN in the output ATTACH message header. If no output message MFLD reference to the *dflename* exists, *'literal'* is used. If the data in the MFLD referencing the *dflename* is greater than eight characters, the first eight characters are used.

**RPRN=**

For DIV TYPE=INPUT, the *dflename* specification permits the suggested return primary resource name (RPRN) to be supplied in the input message MFLD referencing this *dflename*. If *dflename* 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 *dflename* is also specified, the data supplied in the MFLD referencing this *dflename* is used as the RPRN in the output ATTACH message header. If no output message MFLD reference to the *dflename* exists, *'literal'* is used. If the data in the MFLD referencing the *dflename* 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. For more information on the FILL= operand of the MFLD statement, see MFS Language utility (DFSUPAA0) (System Utilities).

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.

**Related concepts:**

“Output format control for SLU P DPM-An” on page 525

“Trailing blank compression” on page 531

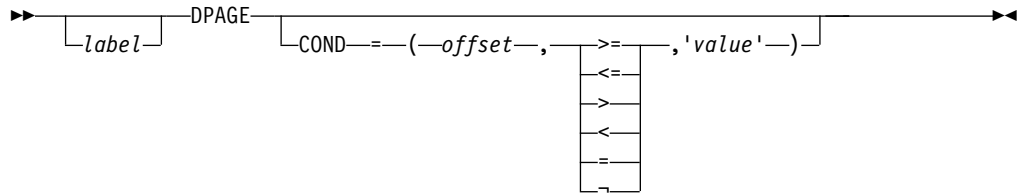
**Related reference:**

“Optional deletion of null characters for DPM-An” on page 498

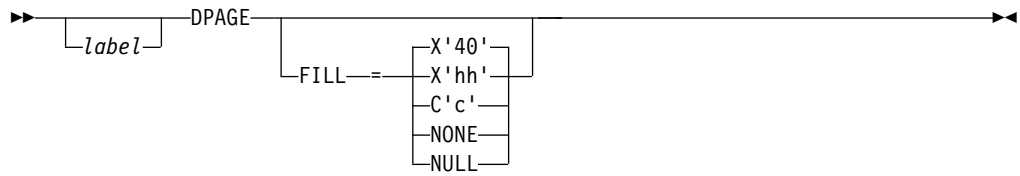
**DPAGE statement:**

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.

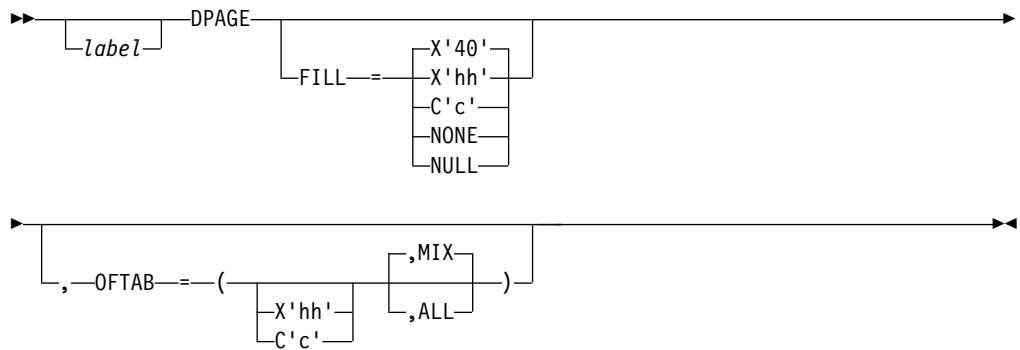
**Format for DEV TYPE=DPM-An or DPM-Bn AND DIV TYPE=INPUT**



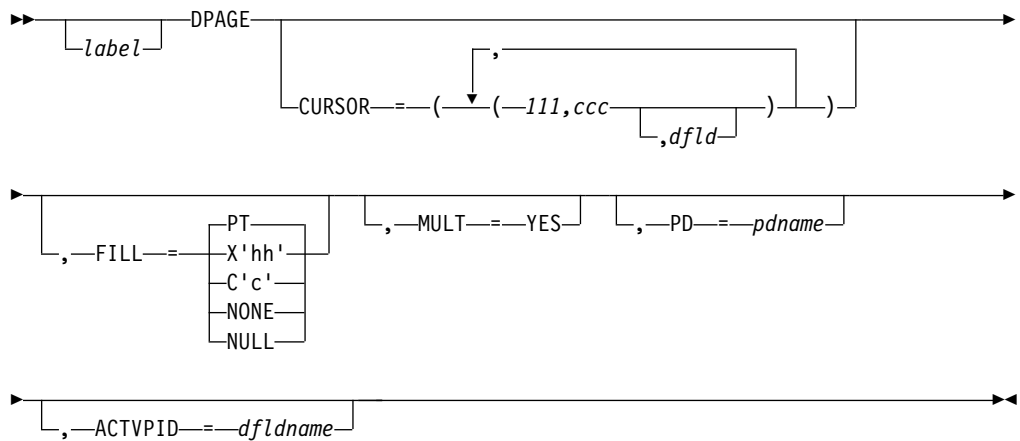
**Format for DEV TYPE=DPM-An AND DIV TYPE=OUTPUT**



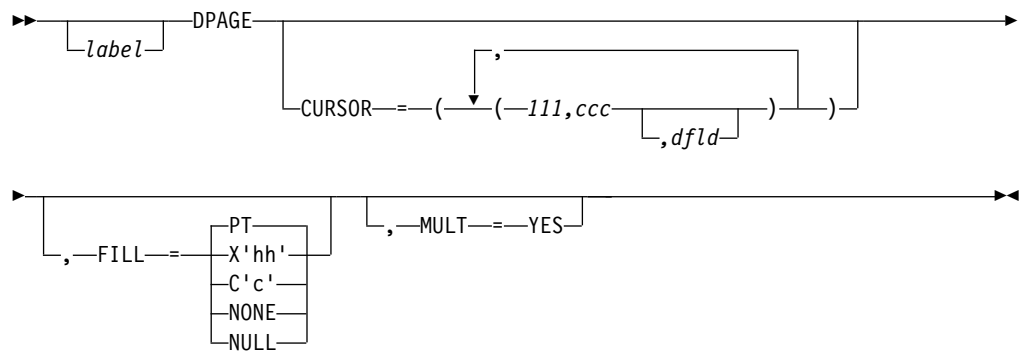
**Format for DEV TYPE=DPM-Bn AND DIV TYPE=OUTPUT**



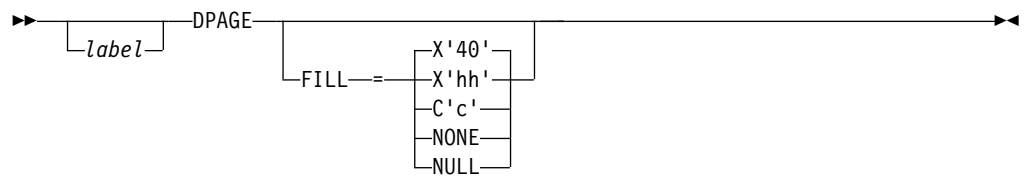
**Format for DEV TYPE=3270-An**



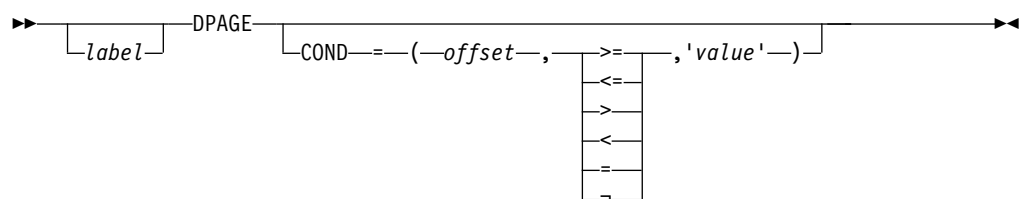
**Format for DEV TYPE=3270**



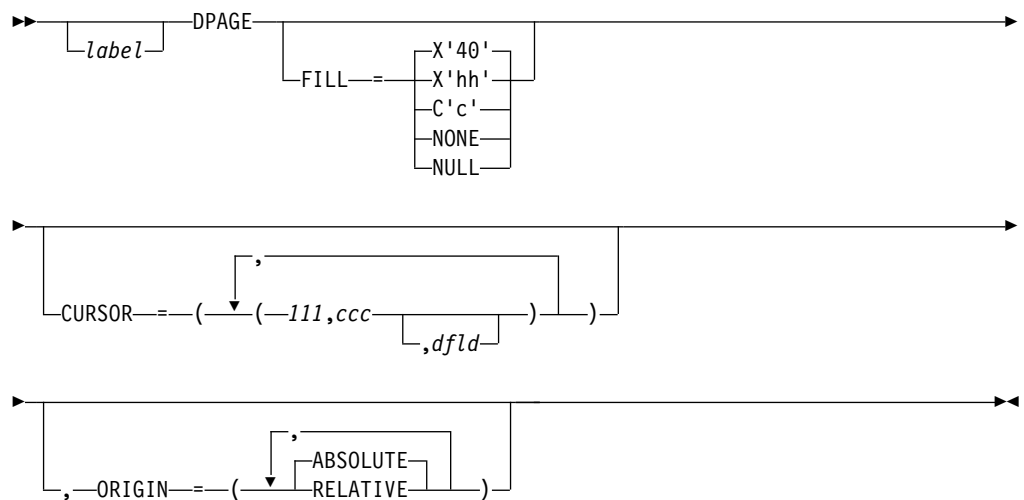
**Format for DEV TYPE=3270P**



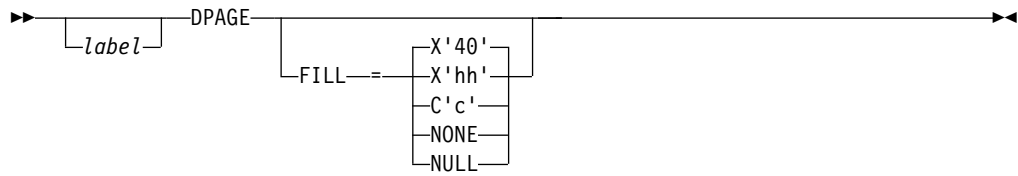
**Format for DEV TYPE=FIN**



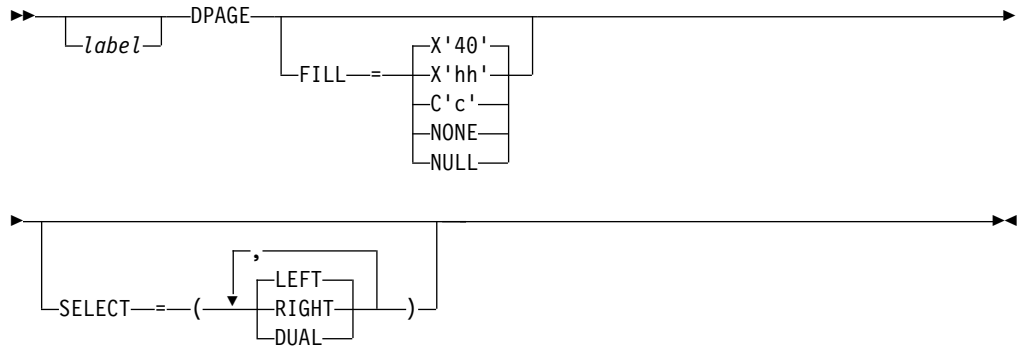
**Format for DEV TYPE=FIDS, FIDS3, FIDS4, or FIDS7**



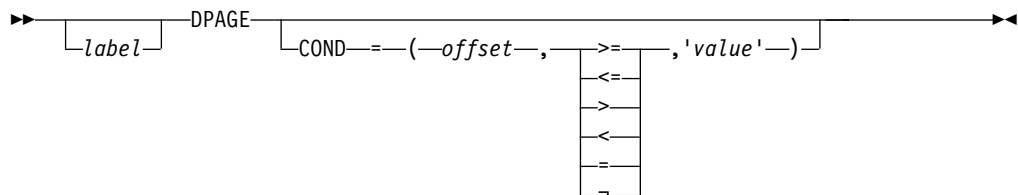
### Format for DEV TYPE=FIJP or FIPB



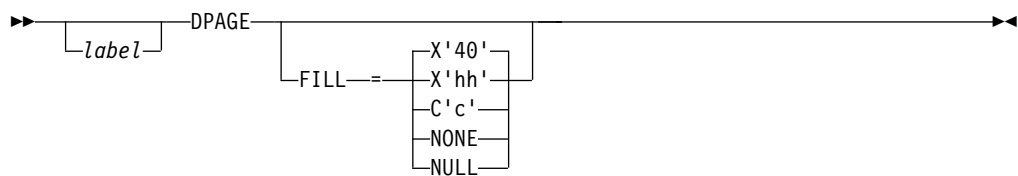
### Format for DEV TYPE=FIFP



### 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 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 *dfl*d 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 *dfl*d 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 *dfl*d 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 *dfllname* 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.

## MFS message formatting functions

IMS provides message formatting functions for MFS. The control blocks format messages for different device types.

### Input message formatting

Use the following information to format your MFS input messages.

#### Related concepts:

“Output message formatting options” on page 506

“Input format control for ISC (DPM-Bn) subsystems” on page 503

#### Input messages accepted by MFS:

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.

#### 3770, SLU 1, and NTO

For MFS to process data from a 3770, SLU 1, or NTO, 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.

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.

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.

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.
- *//bH* 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.



**Restriction:** 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 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. 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.

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. 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.

### **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. 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.


#### **Related tasks:**

 [Extended Terminal Option \(ETO\) \(Communications and Connections\)](#)

#### **Related reference:**

 [/SET command \(Commands\)](#)

 [/FORMAT command \(Commands\)](#)

 [Using Intersystem Communication Edit \(Application Programming\)](#)

#### **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 Version 13 Communications and Connections*.

#### *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 option descriptions, a NULL character is X'3F'.

### **MFS 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.

### **MFS 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.

### **MFS 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.

**Related concepts:**

“Input message formats” on page 446

“Fill characters for output device fields” on page 508

**Related reference:**

“Device-dependent output information” on page 452

*Examples of message segment definitions:*

The 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 the following table.

*Table 129. 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

The following table describes the definition for an input message.

Table 130. Example 1: input message definition

Segment Number	Field Name	Field Length	Field Value
1	LL	2	0072
	ZZ	2	XXXX
	TRANCODE	8	YYYY
	Text	10	MAN NO.
	Text	50	NAME
2	LL	2	0059
	ZZ	2	XXXX
	Text	5	DEPT
	Text	50	LOCATION
3	LL	2	0064
	ZZ	2	XXXX
	Text	10	PART NO.
	Text	50	DESCRIPTION
4	LL	2	0019
	ZZ	2	XXXX
	Text	10	QUANTITY
	Text	5	ORDER PRIORITY

All fields defined as left justified, with a fill character of blank.

You enter:

**Field Name**

**Input**

**NAME**

ABJONES

**PART NO.**

23696

**DESCRIPTION**

WIDGET

The transaction code is provided from the message input description as a literal. The input message would appear to the application program as shown in one of the following tables.

Table 131. Example 1: application program view for option 1

Segment Number	Field Type	Field Length	Field Value
1	A	2	0072
	B	1	XX
	C	1	01
	TRANCODE	8	YYYY
	Text	10	blanks
	Text	50	ABJONES
2	A	2	0059
	B	1	XX
	C	1	01
	Text	5	blanks
	Text	50	blanks
3	A	2	0064
	B	1	XX
	C	1	01
	Text	10	23696
	Text	50	WIDGET
4	A	2	0019
	B	1	XX
	C	1	01
	Text	10	blanks
	Text	5	blanks

Table 132. Example 1: application program view for option 2

Segment Number	Field Type	Field Length	Field Value
1	A	2	0072
	B	1	XX
	C	1	02
	TRANCODE	8	YYYY
	Text	10	blanks
	Text	50	ABJONES
2	A	2	0005
	B	1	XX
	C	1	02
	Text	1	X'3F'
3	A	2	0064
	B	1	XX
	C	1	02
	Text	10	23696
	Text	50	WIDGET

Table 133. Example 1: application program view for option 3

Segment Number	Field Type	Field Length	Field Value
1	A	2	0060
	B	1	XX
	C	1	03
	D	2	0001
	E	2	0054
	F	2	0022
	G	50	ABJONES
2	A	2	0074
	B	1	XX
	C	1	03
	Text	2	0003
	D	2	0014
	E	2	0004
	F	2	23696
	G	2	0054
	F	2	0014
G	50	WIDGET	

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

The segments are similar to those in 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 Finance workstation. The input message appears as shown in one of the following tables.

*Table 134. Example 2: application program view for option 1*

Segment Number	Field Type	Field Length	Field Value
1	A	2	0029
	B	1	XX
	C	1	01
	TRANCODE	8	YYYY
	Text	10	blanks
	Text	50	ABJONES
2	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.		
3	A	2	0064
	B	1	XX
	C	1	01
	Text	10	0000023696
	Text	50	WIDGET
4	A	2	0009
	B	1	XX
	C	1	01
	Text	5	HI

*Table 135. Example 2: application program view for option 2*

Segment Number	Field Type	Field Length	Field Value
1	A	2	0029
	B	1	XX
	C	1	02
	TRANCODE	8	YYYY
	Text	10	blanks
	Text	7	ABJONES



Table 135. Example 2: application program view for option 2 (continued)

Segment Number	Field Type	Field Length	Field Value
2	A	2	0009
	B	1	XX
	C	1	02
	Text	1	X'3F'
3	A	2	0064
	B	1	XX
	C	1	02
	Text	10	0000023696
	Text	50	WIDGET
4	A	2	0009
	B	1	XX
	C	1	02
	Text	5	HI

Table 136. Example 2: application program view for option 3

Segment Number	Field Type	Field Length	Field Value
1	A	2	0029
	B	1	XX
	C	1	03
	D	2	0001
	E	2	0012
	F	2	0004
	G	8	TRANCODE
	E	2	0011
	F	2	0022
	G	7	ABJONES
2	A	2	0074
	B	1	XX
	C	1	03
	D	2	0003
	E	2	0014
	F	2	0004
	G	10	0000023696
	E	2	0054
	F	2	0014
G	50	WIDGET	

Table 136. Example 2: application program view for option 3 (continued)

Segment Number	Field Type	Field Length	Field Value
3	A	2	0015
	B	1	XX
	C	1	03
	D	2	0004
	E	2	0009
	F	2	0014
	G	5	HI

*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 Version 13 Exit Routines* lists the field and segment edit routines provided by IMS. The input message field or segment edit 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 Version 13 Exit Routines* lists the IMS-supplied routines.

Under z/OS, 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.

### **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 also 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”.

For example, consider this MFLD definition:

```
MFLD (DFLD1, 'NO DATA'),LTH=7,JUST=R,FILL=C'0'
```

For example, an application program would view your entries as follows:

<b>Your Entry</b>	<b>Program Data Viewed</b>
296	0000296
0	0000000
<b>no data entered</b>	<b>NO DATA</b>

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 discussed in this section 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.

*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 3770, or SLU 1; 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.

For example, the following figure shows some DFLD field definitions and the device format that results from these definitions.

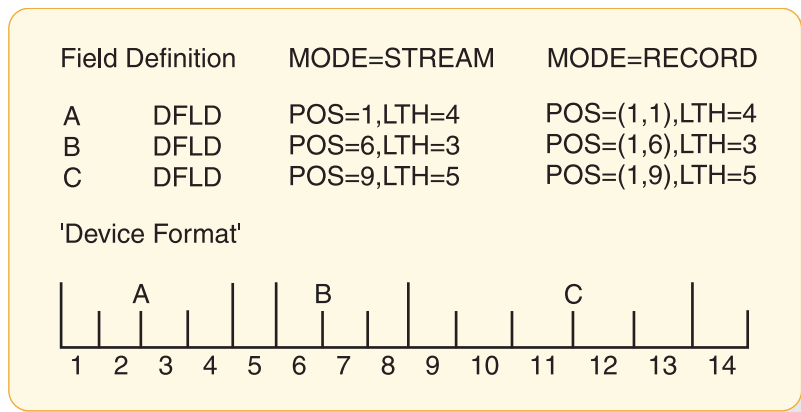


Figure 30. FTAB qualification descriptions

When an FTAB is defined, its use is qualified by specifying FORCE, MIX, or ALL. The previous figure shows how the FTAB qualification affects the results of an MFS input scan following variable operator input of a three-field message.

The following figure provides examples of correct and failed results produced by FTAB specifications. 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. The following sections (FORCE, MIX, and ALL) specify which examples have failed results and why these results are undesirable.

## **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 the following figure, 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 the following figure, 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 the following figure, 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 PARTØ02315231	A	4	PART ←→ 4	4	PART	4	PART
	B	3	023	3	023	0	
	C	5	15231	5	15231	0	
2 P1,23,15231	A	2	P1 ←→ 2	2	P1 ←→ 2	2	P1
	B	2	23	2	23	2	23
	C	5	15231	5	15231	5	15231
3 PART,23,15	A	4	PART ←→ 4	4	PART ←→ 4	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	4	PART ←→ 4	4	PART
	B	0	15231	0	15231	0	15231
	C	5		5		5	
7 PARTØ,15231	A	4	PART ←→ 4	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 31. 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 either IMS TM or the delete operation is costly in execution time. Weigh the relative costs when you decide whether to use the NULL=DELETE option or to delete the nulls using 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'.

**Related reference:**

"DIV statement" on page 467

*Examples of optional null character deletion for DPM-An:*

The following are 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'C1000C2000C3C3C3000000'	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'0000000000'	3	G	no data	3	C'*****'
(2) X'C1000C2000C3C3C3'	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'C5C56BC6C66B0000000000000'	2	E	C'EE'	2	C'EE***'	
		F	C'FF'		C'FF*****'	
X'0000000000000000'	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'	
	X'C5C56BC6C6'	2	D	C'EE'	2	C'EE*'
			E	C'FF'		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, 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.

**Related concepts:**

“Physical paging of output messages” on page 508

**General rules for multiple DPAGE input**

Follow these rules for 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 these 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**

Use the major input message formatting functions of MFS with ISC nodes.

You can use the following DPAGE selection options to format messages and create 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.

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:

- 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.

### Related concepts:

“Input message formatting” on page 482

## Output message formatting

IMS supports the following MFS output message formatting, physical and logical paging, and requirements for output devices.

### Output messages accepted by MFS:

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 3770, Finance workstation, SLU 1, NTO, 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.

**Related reference:**

 /MODIFY command (Commands)

**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 through the 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.

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. 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 Segment	Output data length
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

```

#### Related concepts:

“Input message formatting” on page 482

#### Related reference:

“Field format (options 1 and 2)” on page 451

“Field format (option 3)” on page 451

*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.

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.

#### Related concepts:

“Paging action at the device” on page 537

“Your control of MFS” on page 535

*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. The following figure 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 24x80 screen size.

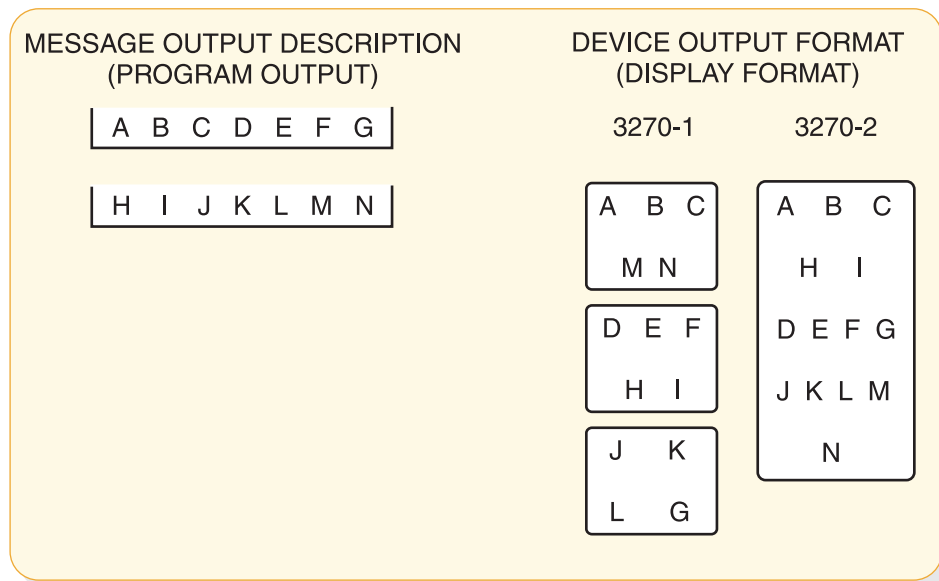


Figure 32. Physical paging for 3270 or SLU 2

**Related concepts:**

“Multiple physical page input messages (3270 and SLU 2 display devices)” on page 501

*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 (SCA).

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.

**Related concepts:**

"System control area (SCA) and default SCA (DSCA)"

**Related reference:**

"Input message formatting options" on page 484

*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 (using the SCA) or in the device format definition (using 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 concepts:**

“Fill characters for output device fields” on page 508

“3270 or SLU 2 screen formatting” on page 429

*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:

- 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 concepts:**

“Extended Graphic Character Set (EGCS)” on page 513

**Related reference:**

 MFLD statement (System Utilities)

*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, 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:

- 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:

**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 exclamation point (!) 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=*nm*. 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.

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.

**Related concepts:**

“Output message literal fields” on page 510

*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 to represent some Asian languages, such as Chinese, Japanese, and Korean; because each of these written languages consists of more than 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.

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.

For example, the following figure shows the case of a DBCS/EBCDIC mixed field.

The DBCS/EBCDIC mixed data shown in the following figure 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'.

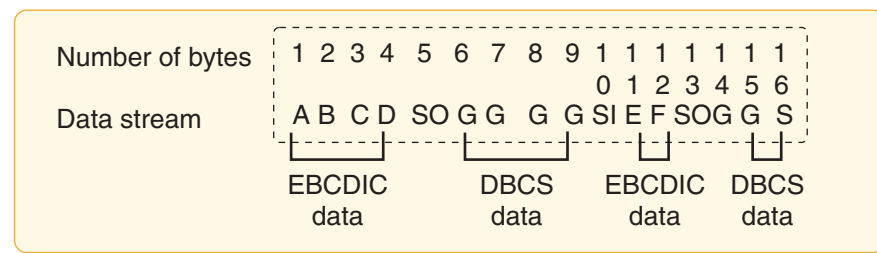


Figure 33. DBCS/EBCDIC mixed data

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 the previous figure 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 these 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 the following code example.

```
literal format: ' .....SO___SI..SO__SI'

DFLD
'literal'

MFLD
,'literal'
,(dlfname,'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.

The following table shows the processing performed by the IMS MFS Language utility for SO/SI control characters within a DBCS/EBCDIC mixed field. The Device and Field are listed, followed by the DFLD/MFLD output literal, and the MFLD input literal.

Table 137. SO/SI processing performed by IMS MFS language utility

Device, Field	DFLD/MFLD Output Literal	MFLD Input Literal
3270 display, DBCS/EBCDIC mixed field	<ul style="list-style-type: none"> <li>• Check SO/SI pairing.</li> <li>• Check even length.</li> <li>• Adjust boundary alignment (with warning message).</li> </ul>	SO/SI checking not done
SCS1 printer, DBCS/EBCDIC mixed field	<ul style="list-style-type: none"> <li>• Check SO/SI pairing.</li> <li>• Check even length.</li> <li>• Perform SO/SI correction and boundary adjustment according to SO/SI blank print option.</li> </ul>	Not applicable

The following table shows the processing performed by the MFS message editor on SO/SI control characters within a DBCS/EBCDIC field. The Device and Field are listed, followed by the outbound data fields and the inbound data fields.

Table 138. SO/SI processing performed by MFS message editor

Device, Field	Outbound Data Fields	Inbound Data Fields
3270 display, DBCS/EBCDIC mixed field	<ul style="list-style-type: none"> <li>• Check SO/SI pairing.</li> <li>• Check even length.</li> <li>• Adjust boundary alignment.</li> </ul>	SO/SI checking not done
SCS1 printer, DBCS/EBCDIC mixed field	<ul style="list-style-type: none"> <li>• Check SO/SI pairing.</li> <li>• Check even length.</li> <li>• Perform SO/SI correction and boundary alignment according to SO/SI blank print option.</li> </ul>	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 the following code sample.



### 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 pairing 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.

Field outlines are referred to as OVER, UNDER, LEFT, and RIGHT lines and they can be specified independently or in any combination.

The area at the left and right ends of the field shown in the following figure 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.

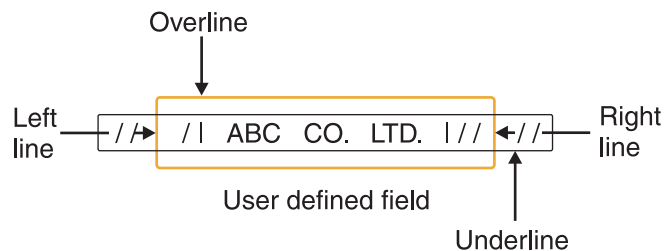


Figure 34. User field and field outlining

### Connecting field outlines and joining fields

You can outline multiple fields jointly as shown in the following figure.

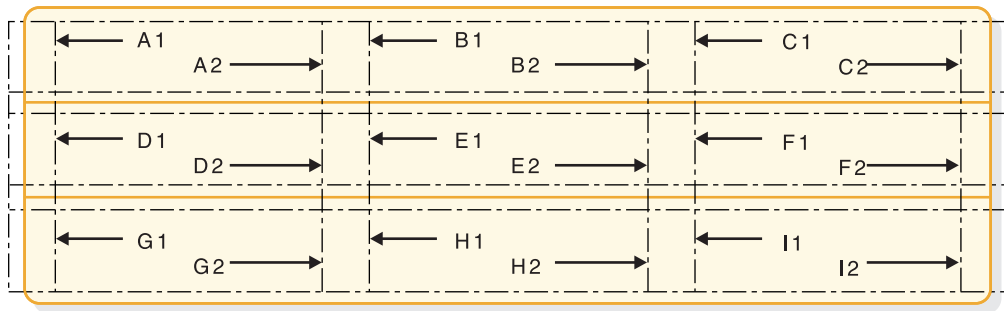


Figure 35. Field outlining when connecting user fields

The previous figure 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 the previous figure is shown in the following table.

Table 139. Outline specification for each field

Fields	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	
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 Version 13 System Definition*.

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.

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 (Extended Recovery Facility) 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 or SLU 1 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. These are the PAGE= operands:

**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, or SLU 1 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, or SLU 1 printers.

**Related concepts:**

“Output format control for 3270P printers” on page 524

“Format control for 3770 and SLU 1 printers”

*Format control for 3770 and SLU 1 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).

**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.

### **Related concepts:**

"Printed page format control" on page 522

*Output format control for 3270P printers:*

MFS provides several specifications to control the format of messages to 3270P printer devices.

## 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 using the FEAT= keyword on the DEV statement.

### Related concepts:

“Printed page format control” on page 522

*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 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.

The following table shows the format of the fixed output message header for OPTIONS=MSG.

*Table 140. Fixed output message header format for OPTIONS=MSG*

FIELD BYTES	BASE 7	LI 1	MIDNAME 8	L2 1	DATANAME 6	L3 1	FORMSNAME (user-coded literal)

**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.

The following table shows the format of the fixed basic output message header (without FORMSNAME) for OPTIONS=DPAGE or PPAGE.

*Table 141. Fixed basic output message header (without FORMSNAME) for OPTIONS=DPAGE or PPAGE*

FIELD BYTES	BASE 7	L1 1	MIDNAME 8	L2 1	DATANAME 8
-------------	--------	------	-----------	------	------------

**BASE** Content is the same as for OPTIONS=MSG.

**L1** Content is the same as for OPTIONS=MSG.

**MIDNAME**

Content is the same as for OPTIONS=MSG.

**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.

The following table shows the format of the optional forms output message header for OPTIONS=DPAGE or PPAGE.

*Table 142. Optional forms output message header for OPTIONS=DPAGE or PPAGE*

FIELD BYTES	BASE 5	L1 1	MIDNAME 8	L2 1	FORMSNAME (user-coded literal)
-------------	--------	------	-----------	------	--------------------------------

**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.

**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.

**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 *IMS Version 13 Database Utilities* for a discussion of deletion of null characters in transmission records.

#### **Related reference:**

"DIV statement" on page 467

### **Output format control for ISC (DPM-Bn) subsystems**

IMS supports 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).

#### **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).

**Related reference:**

“Device-dependent output information” on page 452

**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

The following code example shows the data entered by the IMS application:

Segment 1:

```
DLZZ FIELD A1 | FIELD A2 | FIELD A3 | FIELD A4 | FIELD C1 | 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 BBBB88888 | 4444444444 | DDDDD43. | 3..... | D3D3D3D3
0400      | 0000000000 |      0F | F
```

**Note:** Both segments entered are shortened by the program.

The following table shows the MFS definitions used in the previous code example.

Table 143. MFS definitions for data entered by IMS application

MSGOUT	MSG	TYPE=OUTPUT, SOR=FMTOUT
	SEG	

Table 143. MFS definitions for data entered by IMS application (continued)

MSGOUT	MSG	TYPE=OUTPUT, SOR=FMTOUT
	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	

Examples of variable-length output with blank compression are shown in the code examples below.

The following code example shows variable-length output with blank compression in record mode:

```

VLVB  FIELD A1 THRU A4: (First record)
01    AAAAA,123456,,A4A4A4
06
VLVB  FIELD B1:         (Second record)
00    BBBB BBBB
0C
VLVB  NO DATA:        (Third record)
00
03
VLVB  FIELDS D1 and D3: (Fourth record)
01    DDDDD,,D3D3D3D3
02

```

**Note:**

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.

The following table shows the MFS definitions used for record mode output as shown in the previous code example.

Table 144. MFS definitions for record mode

Field	Type	Definition	
	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	

The following code example shows variable-length output with blank compression 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.

The following table shows the MFS definitions used for stream mode output as shown in the previous code example.

Table 145. MFS definitions for stream mode

Field	Type	Definition	
	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	

Table 145. MFS definitions for stream mode (continued)

Field	Type	Definition
D2	DFLD	LTH=10
D3	DFLD	LTH=10
E1	DFLD	LTH=10
	FMTEND	

**Related reference:**

“DIV statement” on page 467

**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**

IMS provides MFS facilities that can assist you, or allow a remote program to control the display or transmission of output messages.

**Related concepts:**

“Operator logical paging of output messages” on page 507

**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. The maximum value for *nnnn* is 4095.
- Enter `=+n`, `=+nn`, or `=+nnn` to display the *n* th logical page past the current logical page. The maximum value for *nnn* is 999.

- Enter =*n*, =-*nn*, =-*nnn* , or =>*nnn* to display the *n* th logical page before the current logical page. The maximum value for *nnn* is 999.
- 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, 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 that you define. The remaining characters must be  $n[mmm]$ ,  $+n[nn]$ ,  $-n[nn]$ , 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.

**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.

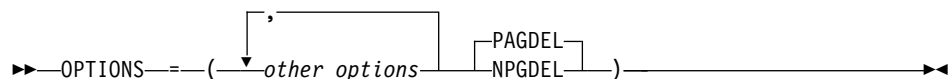
If operator logical paging is **not** specified for a 3604, 3270, SLU 2 display, or SLU P using the DPM paging option, each physical page of each logical page can be viewed in sequence using the NEXTTPP function. Because operator logical paging is not specified, entering NEXTTPP after the last physical 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 NEXTTPP function can be used to view pages sequentially. However, entering NEXTTPP 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. 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.

The following tables describe 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: NEXTTPP 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.

- The following tables use 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 the following tables should be ignored. If the message is sent to the terminal with the unprotected screen option set to “protect”, the IMS actions shown in the following tables apply.

The following tables assume the system and message definition values and page position in the current message that apply in the following four cases:

**Case 1** PAGE=NO and the last physical page of the last logical page of the current message was sent.

**Case 2** PAGE=NO and the last physical page of the last logical page of the current message was *not* sent.

**Case 3** PAGE=YES and the last physical page of the last logical page of the current message was sent.

**Case 4** PAGE=YES and the last physical page of the last logical page of the current message was *not* sent.

**Note:** If an error message has been sent to the last page, the following tables do not apply. The original message is still in the queue. See *IMS Version 13 System Utilities* for the proper response to the message.

For the following table, the IMS action, after successful IMS transmission of the message and terminal receipt of the message, for each of these four cases is PROTECT, that is, IMS prevents the device from receiving output from IMS. For Case 1, IMS also dequeues the message from the IMS message queue.

*Table 146. Paging operation for a device with MFS with PAGDEL specified. IMS-MFS action and resulting terminal and message status*

Operator Action	IMS Action for Case 1	IMS Action for Case 2	IMS Action for Case 3	IMS Action for Case 4
Request PAGE ADVANCE (NEXTTPP)	Unprotected	Send next physical page unprotected	Send error message, protected <sup>1</sup>	Send next physical page, protected
Request LOGICAL PAGE ADVANCE (NEXTLP)	Unprotected	Send first physical page of next logical page in current msg <sup>2</sup>	Send error message, protected <sup>1</sup>	Send first physical page of next logical page in current msg <sup>2</sup>
Request specific logical page using =PAGE	Send error message, protected <sup>3</sup>	MSG DEQ, send error message protected <sup>3</sup>	If valid, send first physical page of requested logical page, protected. <sup>1</sup> If invalid, send error message protected. <sup>1</sup>	
Request specific logical page using PAGEREQ	Send error message, protected	Send error message, protected <sup>1</sup>	If valid, send first physical page of requested logical page, protected. <sup>1</sup> If invalid, send error message protected. <sup>1</sup>	
Request MESSAGE ADVANCE (NEXTMSG)	Unprotected	MSG DEQ, unprotected	MSG DEQ, unprotected	MSG DEQ, unprotected
Request MESSAGE ADVANCE PROTECT (NEXTMSGP)	Protected <sup>4</sup>	MSG DEQ, protected <sub>4</sub>	MSG DEQ, protected <sub>4</sub>	MSG DEQ, protected <sub>4</sub>
Enter data	MSG ENQ, unprotected	MSG DEQ, MSG ENQ, unprotected	MSG DEQ, MSG ENQ, unprotected	MSG DEQ, MSG ENQ, unprotected

**Notes:**

1. The original message is still in the queue. See *IMS Version 13 System Utilities* for the proper response to the message.
2. If the current page was the last logical page, no new page is sent, and device status is unprotected.
3. If the device is preset or in conversation, the input is queued; no error message is sent and the device status is unprotected.
4. 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.

For the following table, the IMS action, after successful IMS transmission of the message and terminal receipt of the message, for Cases 2, 3, and 4 is PROTECT,

that is, IMS prevents the device from receiving output from IMS. For Case 1, IMS dequeues the message from the IMS message queue.

Table 147. Paging operation for a device with MFS with NPAGDEL specified. IMS-MFS action and resulting terminal and message status

Operator Action	IMS Action for Case 1	IMS Action for Case 2	IMS Action for Case 3	IMS Action for Case 4
Request PAGE ADVANCE (NEXTPP)	Unprotected	Send next physical page, protected	Send error message, protected <sup>1</sup>	Send next physical page, protected
Request LOGICAL PAGE ADVANCE (NEXTLP)	Unprotected	Send first physical page of next logical page in current msg <sup>2</sup>	Send error message, protected <sup>1</sup>	Send first physical page of next logical page in current msg <sup>2</sup>
Request specific logical page using =PAGE	Send error message, protected <sup>3</sup>	Send error message, protected <sup>1,3</sup>	If valid, send first physical page of requested logical page, protected. If invalid, send error message, protected. <sup>1</sup>	
Request specific logical page using PAGEREQ	Send error message, protected	Send error message, protected <sup>1</sup>	If valid, send first physical page of requested logical page, protected. If invalid, send error message, protected. <sup>1</sup>	
Request MESSAGE ADVANCE (NEXTMSG)	Unprotected	MSG DEQ, unprotected	MSG DEQ, unprotected	MSG DEQ, unprotected
Request MESSAGE ADVANCE PROTECT (NEXTMSGP)	Protected <sup>4</sup>	MSG DEQ, unprotected	MSG DEQ, protected <sup>4</sup>	MSG DEQ, protected <sup>4</sup>
Enter data	MSG ENQ, unprotected	MSG ENQ <sup>5</sup>	MSG ENQ <sup>5</sup>	MSG ENQ <sup>5</sup>

**Notes:**

1. The original message is still in the queue. See *IMS Version 13 System Utilities* for the proper response to the message.
2. If the current page was the last logical page, no new page is sent, and device status is unprotected.
3. If the device is preset or in conversation, the input is queued; no error message is sent and the device status is unprotected.
4. 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.
5. 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.

**Related concepts:**

- “Operator logical paging of output messages” on page 507
- “Unprotected screen option” on page 542

### 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 SYSMMSG 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.

The following table 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 148. IMS protect or unprotect action based on OPTIONS specification.*

<b>Output Message</b>	<b>IMS System Definition (PRO)</b>	<b>IMS System Definition (UNPRO)</b>
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.

**Related concepts:**

“Paging action at the device” on page 537

**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 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 using 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.

For 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.

For 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.

#### **Related concepts:**

“3290 screen formatting” on page 431

“3180 in partitioned format mode”

#### **Related reference:**

“Device-dependent output information” on page 452

### **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.

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.

Clearing the display and scrolling is handled in the same way on the 3180 as on the 3290 in partitioned format mode.

**Related concepts:**

“3290 in partitioned format mode” on page 543

## **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 the DFSDF1, DFSDF2, or DFSDF4 format definitions. 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.

### **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 Version 13 Communications and Connections*.) The format applies to 3270 and SLU 2 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.

#### **Related concepts:**

“MFS formatting for the 3270 or SLU 2 master terminal”

### **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.

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 Version 13 System Administration*.

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.

#### Related concepts:

“MFS format sets supplied by IMS” on page 546

### 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 'x' in DFSUDT0 x corresponds to the parameter specified on the SUFFIX= keyword of the IMSGEN macro.

The MFS Device Characteristics table can be updated with the MFS Device Characteristics Table utility (DFSUTB00), which allows updates to the table without system regeneration. 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.

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.SDFSRESL library.

The MFS Language utility (DFSUPAA0) 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 Device Characteristics Table utility can be used to add to or update the table without regenerating the system definition.

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.SDFSRESL 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.SDFSRESL library, the MFS Language utility attempts to load the IMS Device Characteristics table using the default name (DFSUDT00).

**Note:** If no default table (DFSUDT00) was created at system definition a failure will result.

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.

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 listed screen sizes for each of the user-defined symbolic names:

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

**Related reference:**

-  MFS Language utility (DFSUPAA0) (System Utilities)
-  MFS Device Characteristics Table utility (DFSUTB00) (System Utilities)
-  TYPE macro (System Definition)
-  TERMINAL macro (System Definition)

**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 communication 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 6. OTMA Callable Interface API reference

IMS provides OTMA access with the OTMA Callable Interface (C/I) API.

See *IMS Version 13 Messages and Codes, Volume 2: Non-DFS Messages* for codes and messages used by the OTMA C/I.

---

### OTMA Callable Interface API calls

The OTMA callable interface application programming interfaces (APIs) are listed here.

#### Using the Header File DFSYC0.H:

The header file included in the API calling program declares each API invocation and variables used for the invocation.

For a C/C++ program using OTMA Callable Interface, the C/C++ header file, DFSYC0.H, needs to be included in the C/C++ program.

#### Load Module DFSYCRET:

The object stub, DFSYCRET, receives all the API invocations and issues a SVC call to perform the requested function. The object stub needs to be available during the binding of the API invoking program. DFSYCRET can be found in SDFSRESL or ADFSLOAD data sets.

### OTMA C/I hints and tips

Use the following hints and tips when programming with OTMA C/I.

- Some OTMA C/I API calls have an ECB parameter that is posted by the function or by an SRB routine that the function precipitates. The caller must check the ECB and wait for it to be posted before inspecting the return code and output data. Be sure to initialize the ECB with 0 before passing to the OTMA C/I call. The calls that include the ECB parameter are:
  - otma\_open
  - otma\_openx
  - otma\_send\_receive
  - otma\_send\_receivex
  - otma\_send\_async
  - otma\_receive\_async
- Each otma\_alloc call creates an independent session for the subsequent otma\_send\_receive call. One of the otma\_alloc calls can be used to specify the name of IMS transaction or IMS command to be sent to IMS. The maximum length of the transaction name is 8 characters. If no transaction name or command is specified in the otma\_alloc call, the transaction name, followed by one or more blanks, or command needs be specified in the beginning of the send buffer of the otma\_send\_receive call. After the otma\_send\_receive call, otma\_free is required, except for the IMS conversation transaction. See the invocation sample C for sending a conversation transaction.

- The OTMA C/I builds the standard LLZZ prefix of IMS application data format. You do not need to build the LLZZ prefix.
- To send a multi-segment message to IMS, the send segment list of the `otma_send_receive` call must identify the length of each input segment. The first element in the segment list specifies the number of the segment. The first element is then followed by the length of segment 1, the length of segment 2, and so on.
- When a multi-segment output message is received, an output segment list is provided for the `otma_send_receive` call. The first element in the output segment list contains the number of the output segment. The first element is then followed by the length of output segment 1, the length of output segment 2, and so on.
- Sample programs (DFSYCSMP) are shipped with IMS.
- The OTMA C/I can be used to send a protected transaction to IMS by passing a context token to the `otma_send_receive` call.
- Because some of the OTMA C/I calls require the calling program to wait, implementing the time-out routine in the calling program is highly recommended to avoid long running transactions in IMS and the internal OTMA C/I hang.
- To run the OTMA C/I application efficiently, limit the number of `otma_open` and `otma_close` calls in the application. Also, for all `otma_open` and `otma_create` calls, try to use the same member name rather than generating a different member name for each call.
- If the size of the output receive buffer specified in the `otma_send_receive` call is too small, the actual data returned is limited by the size of the receive buffer. The output can be rejected if a special option, `SyncLevel1`, is specified in the `otma_alloc` call. However, if the size of the output receive buffer is too small for the `otma_receive_async` call, the OTMA C/I always rejects the output.
- The OTMA C/I can support various program-to-program switches in IMS. See *IMS Version 13 Communications and Connections* for more information.
- In some cases, OTMA C/I returns a return code to inform the caller about an abnormal condition. Logging or saving the return code for debugging purpose is recommended.
- The `otma_send_receive` call sends an OTMA send-then-commit message with `synclevel=none` to IMS. The caller can set a `synclevel=confirm` for `otma_send_receive`.
- When an input z/OS Resource Recovery Services (RRS) context token is given in the `otma_send_receive` call, the `synclevel` is then changed to `SYNCPT` to support the protected transaction.
- For complex program-to-program switches in IMS, a send-then-commit input message could result in a commit-then-send output message instead of the expected send-then-commit output message. The OTMA C/I works in this scenario. See *IMS Version 13 Communications and Connections* for more information on program-to-program switches.
- The `otma_send_async` call sends an OTMA commit-then-send message to IMS.
- The `otma_receive_async` call receives an OTMA commit-then-send output message from IMS.
- The OTMA C/I does not support either the OTMA resync protocol or the OTMA security PROFILE option.

**Related reference:**

“OTMA C/I sample program for synchronous processing” on page 569



“OTMA C/I sample program for asynchronous processing” on page 581

## otma\_create API

Use the `otma_create` API to allocate storages for z/OS cross-system coupling facility (XCF) and IMS communications.

### Description

After the call, an anchor will be returned. The anchor must be used for the subsequent calls. Invoking `otma_create` is not required. During the `otma_open`, OTMA C/I will allocate storages for communication, if it detects that `otma_create` has not been called. If `otma_create` is invoked first, the same input parameters need to be used again for the subsequent `otma_open` call.

### Invocation

Called by the client in TCB mode.

### Input

**\*ecb** Pointer to the next event control block.

**\*group\_name**  
Pointer to the string containing the XCF group name. (char[8])

**\*member\_name**  
Pointer to the string containing the XCF member name for this member.  
(char[16])

**\*partner\_name**  
Pointer to the string containing the XCF member name for IMS. (char[16])

**\*sessions**  
Number of parallel sessions that are intended to be supported with IMS.  
Long integer from 001 to 999.

**\*tpipe\_prefix**  
First 1 to 4 characters of the tpipe names. (char[4])  
  
For more information on OTMA tpipe naming conventions, see *IMS Version 13 Communications and Connections*.

**Attention:** For the input fields **group\_name**, **member\_name**, and **partner\_name**, all XCF names that are pointed to must be left justified, filled with blanks, and consist of legal upper case EBCDIC characters. If any of those naming rules are violated, underlying XCF errors will be reported.

### Output

**\*anchor**  
Pointer to the anchor word.

**\*retrsn**  
Pointer to the return code structure.

### C-language function prototype

```
otma_create(  
    otma_anchor_t      *anchor,      [out]  
    otma_retrsn_t     *retrsn,      [out]  
    ecb_t              *ecb,        [in]
```

otma_grp_name_t	*group_name,	[in]
otma_clt_name	*member_name,	[in]
otma_srv_name	*partner_name,	[in]
signed long int	*sessions,	[in]
unsigned char	*tpipe_name);	[in]

## Return values (rc value)

The rc and reason are valid after ECB has been posted. For the complete description of each error, see *IMS Version 13 Communications and Connections*.

- 0 The call was completed successfully.
- 8 User error.
- 12 Storage obtain failure.

## otma\_open API

The caller must call `otma_open` to connect when IMS is available. The caller must wait on the ECB, that is posted when the connection is completed or when the attempt has failed. When IMS is not up or OTMA is not started the attempt will fail.

### Description

The caller can cancel the attempt to connect with IMS by issuing an `otma_close` call at any time. The ECB will be posted accordingly.

If IMS fails after this connection is established, any call to a function interface will receive a return code to indicate that IMS is no longer listening for messages. If IMS resumes before a close is performed, the connection will be reestablished without any action from the client. The `otma_close` and `otma_open` interfaces may be called again to reestablish communications with IMS. All existing conversations will have been terminated. This implementation does not use OTMA Resynchronization Protocol.

An extended version of the `otma_open` API, which is called `otma_openx`, provides extended functionality.

### Invocation

Called by the client in TCB mode.

### Input

#### \*anchor

Pointer to the anchor word. If `otma_create` is not used to set up the anchor environment, the anchor word must be initialized with a zero.

#### \*group\_name

Pointer to the string containing the z/OS cross-system coupling facility (XCF) group name. (char[8])

#### \*member\_name

Pointer to the string containing the XCF member name for this member. (char[16])

#### \*partner\_name

Pointer to the string containing the XCF member name for IMS. (char[16])

**\*sessions**

Number of parallel sessions that are intended to be supported with IMS.  
Long integer from 001 to 999.

**\*tpipe\_prefix**

First 1 to 4 characters of the tpipe names. (char[4]).

For more information on OTMA tpipe naming conventions, see *IMS Version 13 Communications and Connections*.

**Attention:** For the input fields **group\_name**, **member\_name**, and **partner\_name**, all XCF names that are pointed to must be left justified, filled with blanks, and consist of legal upper case EBCDIC characters. If any of those naming rules are violated, underlying XCF errors will be reported.

**Output****\*anchor**

Pointer to the anchor word to receive the address of global storage.

**\*retrsn**

Pointer to the return code structure.

**\*ecb**

Pointer to the event control block to be posted when the open completes.

**C-language function prototype**

```
otma_open(
    otma_anchor_t    *anchor           [in/out]
    otma_retrsn_t   *retrsn,          [out]
    ecb_t            *ecb,             [out]
    otma_grp_name_t *group_name,       [in]
    otma_clt_name_t *member_name,      [in]
    otma_srv_name_t *partner_name,     [in]
    signed long int *sessions,         [in]
    unsigned char   *tpipe_name);     [in]
```

**Post codes**

The caller of the OPEN routine must check the ECB that was provided to OPEN. If this ECB is not already posted, the caller must wait for this ECB (for the OPEN protocol to complete).

- 0 XCF OPEN completes successfully.
- 4 IMS is not ready. Try again later.
- 8 Your XCF group and member are already active.
- 12 A system error occurred.

**Return values (rc value)**

The rc and reason are valid after ECB has been posted.

- 0 XCF JOIN was successful, client-bid was sent, and acknowledgment received. For the complete description of each error, see *IMS Version 13 Communications and Connections*.
- 4 IMS is not ready. Try again later.
- 8 Your XCF group and member are already active.
- 12 A system error occurred.

**Related reference:**

“otma\_openx API”

## otma\_openx API

The otma\_openx API is an extended version of the otma\_open API, with additional features. The caller must call otma\_openx to connect when IMS is available. The caller must wait on the ECB, that is posted when the connection is completed or when the attempt has failed. When IMS is not up or OTMA is not started the attempt will fail.

### Description

The extended features include:

- The ability to specify an OTMA DRU exit routine.
- Added capability for future enhancements to the API.

### Invocation

Called by the client in TCB mode.

### Input

Same as for the otma\_open API, with two additional parameters:

**\*anchor**

Pointer to the anchor word. If otma\_create is not used to set up the anchor environment, the anchor word must be initialized with a zero.

**\*group\_name**

Pointer to the string containing the z/OS cross-system coupling facility (XCF) group name. (char[8])

**\*member\_name**

Pointer to the string containing the XCF member name for this member. (char[16])

**\*partner\_name**

Pointer to the string containing the XCF member name for IMS. (char[16])

**\*sessions**

Number of parallel sessions that are intended to be supported with IMS. Long integer from 001 to 999.

**\*tpipe\_prefix**

First 1 to 4 characters of the tpipe names. (char[4]).

For more information on OTMA tpipe naming conventions, see *IMS Version 13 Communications and Connections*.

**\*ims\_dru\_name**

Pointer to the string containing the user-defined OTMA User Data Formatting exit routine. This is an extended API parameter.

**\*special\_options**

Pointer to an area codifying non-standard options. Currently, there are no special options supported. Specify a NULL for this parameter. This is an extended API parameter.

**Attention:** For the input fields **group\_name**, **member\_name**, and **partner\_name**, all XCF names that are pointed to must be left justified, filled with blanks, and consist of legal upper case EBCDIC characters. If any of those naming rules are violated, underlying XCF errors will be reported.

## Output

### \*anchor

Pointer to the anchor word to receive the address of global storage.

### \*retrsn

Pointer to the return code structure.

### \*ecb

Pointer to the event control block to be posted when the open completes.

## C-language function prototype

```
otma_openx(  
    otma_anchor_t      *anchor,          [out]  
    otma_retrsn_t      *retrsn,         [out]  
    ecb_t              *ecb,            [out]  
    otma_grp_name_t    *group_name,     [in]  
    otma_clt_name_t    *member_name,    [in]  
    otma_srv_name_t    *partner_name,   [in]  
    signed long int    *sessions,       [in]  
    tpipe_prfx_t       *tpipe_prefix,   [in]  
    otma_dru_name_t    *ims_dru_name,   [in]  
    otma_profile4_t    *special_options); [in]
```

## Post codes

Same as for the `otma_open` API.

## Return values (rc value)

Same as for the `otma_open` API.

### Related reference:

“`otma_open` API” on page 556

## otma\_alloc API

The `otma_alloc` API is called to create an independent session to exchange messages.

## Invocation

Called by the client in TCB mode.

## Input

### \*anchor

Pointer to anchor word that was set up by `otma_open`.

### \*username

Pointer to string holding the RACF username for transaction commands.

For calls from authorized programs, the input username is trusted and passed to IMS. For calls from unauthorized programs, OTMA C/I invokes a RACF call with the current accessor environment element (ACEE) context to obtain the username. The input username, if any, will be ignored. A NULL can be specified for callers from unauthorized programs.

**\*transaction**

Name of IMS transaction or command to be sent to IMS.

If the IMS command entered is longer than eight characters, the first eight characters of the command can be provided in this parameter. The rest of the characters of the command need to be provided in the beginning of the send buffer of the subsequent `otma_send_receive` API.

If this parameter is left blank, then the IMS transaction name or command must be specified (left aligned) in the beginning of the send buffer of the subsequent `otma_send_receive` API.

**\*prfname**

Pointer to a string holding the RACF group name for transactions/commands.

**\*special\_options**

Pointer to the processing options for the subsequent `otma_send_receive` or `otma_send_receivex` API call. The supported processing options include:

- Bit 0** SyncOnReturn - with this option, IMS is asked to process the message without the z/OS Resource Recovery Services (RRS) context token; in this case, the user ID is obtained when RRS CTXRDTA is invoked.
- Bit 1** SyncLevel1 - with this option, OTMA `send_then_commit` sync level 1 is used instead of sync level 0, which is the default for OTMA C/I. Refer to the DFSYCO header file for additional information.

**Output****\*retrsn**

Pointer to return code structure.

**\*session\_handle**

Pointer to session handle that uniquely identifies the session for the subsequent `otma_send_receive`.

**C-language function prototype**

```
otma_alloc(
    otma_anchor_t *anchor,           [in]
    otma_retrsn_t *retrsn,          [out]
    sess_handle_t *session_handle,  [out]
    otma_profile_t *special_options, [in]
    tran_name_t *transaction,       [in]
    racf_uid_t *username,           [in]
    racf_prf_t *prfname);          [in]
```

**Return values (rc value)**

The rc and reason are valid after ECB has been posted. For the complete description of each error, see *IMS Version 13 Communications and Connections*.

- 0 Success.
- 4 Session limit reached.
- 8 Null anchor.

**otma\_send\_receive API**

The `otma_send_receive` API is invoked to initiate a message exchange with IMS.

## Description

The caller gives buffer definitions for both send and receive. Both send buffer and receive buffer information is provided. By providing receive information at the same time as send there are no unexpected messages from IMS, greatly simplifying the protocol. When the reply arrives from IMS the ECB will be posted. All the work of buffer management is handled in the message exit routine.

An extended version of the `otma_send_receive` API, which is called `otma_send_receivex`, provides extended functionality.

## Invocation

Called by the client in TCB mode.

## Input

### \***anchor**

Pointer to anchor word that was set up by `otma_open`.

### \***session\_handle**

Pointer to session handle for `tpipe` returned by `otma_alloc`.

\***lterm** Pointer to `lterm` name field. On input is passed to IMS. Will be updated on output to `lterm` field returned by IMS. Can be blank in both cases.

### \***modname**

Pointer to `MODname` name field. On input is passed to IMS. Will be updated on output to `MODname` field returned by IMS. May be blank in both cases.

If the input `modname` is `DFSM01`, `DFSMO2`, or `DFSM05`, it will be treated as blanks.

### \***send\_buffer**

Pointer to the data to be sent to IMS. When a NULL is specified for the transaction parameter, the client code must provide the transaction name or command, and a blank, to the data in this buffer when sending to IMS.

### \***send\_length**

Length of send data.

### \***send\_segment\_list**

An array of lengths of message segments to be sent to IMS. First element is count of following segment lengths. Optional: If a single segment is to be sent, either the first element or the address of the array can be zero.

### \***receive\_buffer**

Pointer to buffer to receive reply message from IMS.

### \***receive\_length**

Length of buffer available to receive message.

### \***receive\_segment\_list**

An array to hold the number of segments sent by IMS. First element must be set as the number of elements in the array. Optional: If a single segment is to be received, either the first element or the address of the array can be zero. In which case all segments will be received contiguously without indication of segmentation boundaries.

### \***context\_id**

Null or Distributed Sync Point Context ID from z/OS Resource Recovery Services.

- For an authorized caller, OTMA C/I passes the Context ID directly to IMS and does not validate the Context ID data.
- For an unauthorized caller, OTMA C/I invokes the CTXSWCH call to disassociate the token and to validate if the token is current for a task. When OTMA C/I receives a response from IMS, it switches the context back onto the task before returning control to the caller.

## Output

### \*retrsn

Pointer to return code structure.

### \*ecb Event

Control block to be posted when the message exchange is complete.

### \*received\_length

Field to receive length of data received to receive\_buffer. Should be equal to the sum of the segment lengths.

### \*receive\_segment\_list

An array of lengths of message segments received from IMS. First element is count of following segment lengths and must be set by client to indicate maximum length of array. It will be modified by receive.

### \*error\_message

Address of the pointer to the error message field. It is provided by the user to receive error or informational messages from IMS. If the post code returns a 20, then this field will contain data.

## C-language function prototype

```
otma_send_receive(
    otma_anchor_t *anchor,           [in]
    otma_retrsn_t *retrsn,          [out]
    ecb_t *ecb,                      [out]

    sess_handle_t *session_handle,   [in]
    lterm_name_t *lterm,             [in/out]
    mod_name_t *modname,            [in/out]

    char *send_buffer,              [in]
    data_leng_t *send_length,        [in]
    ioseg_list_t *send_seg_list,     [in]

    char *receive_buffer,           [in]
    data_leng_t *receive_length,     [in]
    data_leng_t *received_length,    [out]
    ioseg_list_t *receive_segment_list, [in/out]
    context_t *context_id,          [in]

    char *error_message);           [out]
```

## Post codes

- 0 Normal completion.
- 8 No anchor/bad session handle/segment too large.
- 12 Send failed.
- 16 Receive has been canceled.
- 20 Error from IMS.



## Return values (rc value)

The rc and reason are valid after ECB has been posted. For the complete description of each error, see *IMS Version 13 Communications and Connections*.

- 0 Normal completion.
- 8 No anchor/bad session handle/segment too large.
- 12 Send failed.
- 16 Receive has been canceled.
- 20 Error from IMS.

## otma\_send\_receivex API

The `otma_send_receivex` API has the same functionality as the `otma_send_receive` API, but adds the extended ability to pass OTMA user data.

### Invocation

Same as for the `otma_send_receive` API.

### Input

Same as for the `otma_send_receive` API, with the following additional parameter:

#### **\*otma\_user\_data**

Pointer to the OTMA user data. The OTMA user data field can contain any user data that is used to identify the user input, or to correlate input with output. If a value is specified in this field, the data is sent to IMS. IMS user exits OTMAIOED and DFSYDRU0 can read or change the data. The data is returned to the user if the `otma_receive_async` API with `otma_user_data` is issued.

If there is no OTMA user data, specify a NULL for this field.

### Output

Same as for the `otma_send_receive` API.

### C-language function prototype

```
otma_send_receivex(  
    otma_anchor_t *anchor,           [in]  
    otma_retrsn_t *retrsn,          [out]  
    ecb_t *ecb,                     [out]  
  
    sess_handle_t *session_handle,   [in]  
    lterm_name_t *lterm,             [in/out]  
    mod_name_t *modname,            [in/out]  
  
    char *send_buffer,              [in]  
    data_leng_t *send_length,       [in]  
    data_leng_t *send_segment_list, [in]  
  
    char *receive_buffer,           [in]  
    data_leng_t *receive_length,    [in]  
    data_leng_t *received_length,   [out]  
    data_leng_t *receive_segment_list, [in/out]  
    context_t *context_id,         [in]
```

```

char          *error_message,          [out]
otma_user_t  *otma_userdata);        [in/out]

```

### Post codes

Same as for the `otma_send_receive` API.

### Return values (rc value)

Same as for the `otma_send_receive` API.

## otma\_send\_async API

The `otma_send_async` API is invoked to send a transaction or command to IMS.

### Invocation

Called by the client in TCB mode.

**Restriction:** This API cannot be used to submit an IMS fast path transaction, a protected transaction (the transactions with z/OS Resource Recovery Services context IDs), or an IMS conversational transaction. For these three types of transactions, use the `otma_send_receive` API instead.

### Input

#### \*anchor

Pointer to anchor word that was set up by `otma_open`.

\***lterm** Pointer to lterm name field. If there is no input lterm, specify a NULL.

#### \*modname

Pointer to MODname name field. If there is no input MODname, specify a NULL.

#### \*otma\_user\_data

Pointer to the OTMA user data. This 1022-byte field is optional. The OTMA user data field can contain any data that is used to identify your input, or to correlate input with output. If a value is specified in this field, the data is sent to IMS. IMS user exits OTMAIOED and DFSYDRU0 can read or change the data. The data is returned if the `otma_receive_async` API with `otma_user_data` is issued.

If there is no OTMA user data, specify a NULL for this field.

#### \*prfname

Pointer to string holding the RACF group name for transaction commands. This parameter is optional. If there is no input RACF group name, specify a NULL.

#### \*send\_buffer

Pointer to the data to be sent to IMS. When a NULL is specified for the transaction parameter, the client code must provide the transaction name or command, and a blank, to the data in this buffer when sending to IMS.

#### \*send\_length

Length of send data.

#### \*send\_segment\_list

An array of lengths of message segments to be sent to IMS. This parameter

is required for multi-segment input messages. If specified, the first element needs to contain the count of total input segments. This field is optional for single segment input. If a single segment is to be sent, either the first element or the address of the array can be zero.

**\*special\_options**

Pointer to an area codifying non-standard options. Currently, no special options are supported. Specify a NULL for this parameter.

**\*tpipe\_name**

Pointer to OTMA tpipe name field. This name must be different from the tpipe name specified for the `otma_create` and `otma_open` APIs.

**\*transaction**

Name of IMS transaction or command to be sent to IMS.

If the IMS command entered is longer than eight characters, the first eight characters of the command can be provided in this parameter. The rest of the characters of the command need to be provided in the beginning of the send buffer.

If NULL or blanks are specified in this parameter, OTMA C/I expects you to include the IMS transaction name or command in the beginning of the send buffer.

**\*username**

Pointer to a string holding the RACF username for transaction/commands.

For calls from authorized programs, the input username is trusted and passed to IMS. For calls from unauthorized programs, OTMA C/I invokes a RACF call with the current accessor environment element (ACEE) context to obtain the username. The input username, if any, will be ignored. A NULL can be specified for callers from unauthorized programs.

## Output

**\*ecb Event**

Event control block to be posted when IMS receives or rejects the input.

**\*error\_message**

Address of the pointer to the error message field. You provide this address to receive error or informational messages from IMS. If the post code returns a 20, then this field will contain data.

**\*retrsn**

Pointer to the return and reason code structure. If IMS OTMA rejects the input, the NAK code and its associated reason code are available in OTMA C/I reason codes 2 and 3. See *IMS Version 13 Messages and Codes, Volume 2: Non-DFS Messages* for an explanation of the NAK code.

## C-language function prototype

```
otma_send_async(
    otma_anchor_t *anchor,           [in]
    otma_retrsn_t *retrsn,         [out]
    ecb_t *ecb,                    [out]

    tpipe_name_t *tpipe_name,      [in]
    tran_name_t *transaction,      [in]
    racf_uid_t *username,          [in]
    racf_prf_t *prfname,          [in]
    lterm_name_t *lterm,           [in]
    mod_name_t *modname,           [in]
    otma_user_t *otma_userdata,    [in]
```

char	*send_buffer,	[in]
data_leng_t	*send_length,	[in]
data_leng_t	*send_segment_list[],	[in]
char	*error_message,	[out]
void	*special_options);	[in]

### Post codes

- 0 Normal completion.
- 8 Invalid input.
- 12 Input failed.
- 16 Input canceled (IMS is down or OTMA is stopped).
- 20 Error or information message from IMS.

### Return values (rc value)

The rc and reason are valid after ECB has been posted. For the complete description of each error, see *IMS Version 13 Communications and Connections*.

- 0 Normal completion.
- 8 No anchor/bad input.
- 12 Send failed.
- 16 Input canceled (IMS is down or OTMA is stopped).
- 20 Error or information message from IMS.

## otma\_receive\_async API

The `otma_receive_async` API is invoked to receive an IMS output message or an unsolicited message. The caller provides the buffer definitions to receive the IMS message. When the IMS message arrives, the ECB is posted.

### Invocation

Called by the client in TCB mode.

### Input

#### \*anchor

Pointer to anchor word that was set up by `otma_open`.

#### \*tpipe\_name

Pointer to OTMA tpipe name field. This name must be different from the tpipe name specified for the `otma_create` and `otma_open` APIs.

#### receive\_length

Length of buffer available to receive message.

### Output

#### \*ecb Event

Event control block to be posted when IMS receives the reply.

#### \*error\_message

Address of the pointer to the error message field. You provide this address to receive error or informational messages from IMS. If the post code returns a 20, then this field will contain data.

**\*lterm** Pointer to lterm name field. Can be updated with lterm value that is returned by IMS.

**\*modname**

Pointer to MODname name field. Can be updated with MODname value that is returned by IMS.

**\*otma\_user\_data**

Pointer to the OTMA user data. This 1022-byte field is optional. If the field is specified and IMS returns the OTMA user data, the data is passed back to the caller.

The OTMA user data received is either provided in the otma\_send\_async API or created by the IMS DRU exit DFSYDRU0.

**\*receive\_buffer**

Pointer to buffer to receive reply message from IMS.

**\*received\_length**

Field to receive length of data received to receive\_buffer. Should be equal to the sum of the segment lengths.

**\*receive\_segment\_list**

An array of lengths of message segments received from IMS. The client must set the first element to indicate the maximum number of message segments that can be received. After all the segments are received, the first array element indicates the actual number of segments received, and the rest of the array elements indicate the length of each segment received.

**\*retrsn**

Pointer to the return and reason code structure.

**\*special\_options**

Pointer to an area codifying non-standard options. Currently, no special are options supported. Specify a NULL for this parameter.

## C-language function prototype

```
otma_receive_async(  
    otma_anchor_t *anchor,           [in]  
    otma_retrsn_t *retrsn,          [out]  
    ecb_t *ecb,                      [out]  
  
    tpipe_name_t *tpipe_name,       [in]  
    lterm_name_t *lterm,            [out]  
    mod_name_t *modname,            [out]  
    otma_user_t *otma_userdata,     [out]  
  
    char *receive_buffer,           [out]  
    data_leng_t *receive_length,     [in]  
    data_leng_t *received_length,    [out]  
    data_leng_t *receive_segment_list[], [in/out]  
  
    void *special_options);         [in]
```

## Post codes

0 Normal completion.

12 Receive failed.

## Return values (rc value)

The rc and reason are valid after ECB has been posted. For the complete description of each error, see *IMS Version 13 Communications and Connections*.

- 0 Normal completion.
- 8 No anchor/bad session handle/segment too large.
- 12 Send failed.

## otma\_free API

The `otma_free` API is called to free an independent session created by `otma_alloc`.

### Invocation

Called by the client in TCB mode.

### Input

- \*anchor**  
Pointer to anchor word returned by `otma_open`.
- \*session\_handle**  
Pointer to session handle returned by `otma_alloc`.

### Output

- \*retrsn**  
Pointer to return code structure.
- \*session\_handle**  
Pointer to session handle will be nulled by `otma_free`.

### C-language function prototype

```
otma_free(  
    otma_anchor_t *anchor,           [in]  
    otma_retrsn_t *retrsn,          [out]  
    sess_handle_t *session_handle); [in/out]
```

## Return values (rc value)

The rc and reason are valid after ECB has been posted. For the complete description of each error, see *IMS Version 13 Communications and Connections*.

- 0 Success.
- 4 Not allocated session.
- 8 Incorrect anchor.

## otma\_close API

The `otma_close` API is called to free storages for communication and to leave the z/OS cross-system coupling facility (XCF) group. This function may be called when communications are in flight or an open is processing. In these cases all relevant ECBs will be posted with a canceled post code.

### Invocation

Called by the client in TCB mode.

## Input

### **\*anchor**

Pointer to anchor word returned by `otma_open`.

## Output

### **\*anchor**

Pointer to anchor word returned by `otma_open`.

### **\*retrsn**

Pointer to return code.

## C-language function prototype

```
otma_close(  
    otma_anchor_t *anchor, [in/out]  
    otma_retrsn_t *retrsn); [out]
```

## Return values (rc value)

The rc and reason are valid after ECB has been posted. For the complete description of each error, see *IMS Version 13 Communications and Connections*.

- 0 Success.
- 4 Null anchor.
- 8 Cannot leave the XCF group.

---

## OTMA C/I sample programs

The following two sample C programs are for display purposes only.

### Warranty and distribution for OTMA C/I sample programs

The OTMA C/I sample programs have warranty and distribution restrictions.

The code is provided "AS IS." IBM makes no warranties, express or implied, including but not limited to the implied warranties of merchantability and fitness for a particular purpose, regarding the function or performance of this code. IBM shall not be liable for any damages arising out of your use of the sample code, even if they have been advised of the possibility of such damages.

The sample code can be freely distributed, copied, altered, and incorporated into other software, provided that it bears the following Copyright notices and DISCLAIMER OF WARRANTIES intact.

```
(c) Copyright IBM Corp.  
2000 All Rights Reserved. Licensed Materials - Property of IBM  
DISCLAIMER OF WARRANTIES.
```

The following "enclosed" code is sample code created by IBM Corporation. This sample code is not part of any standard or IBM product and is provided to you solely for the purpose of assisting you in the development of your applications.

### OTMA C/I sample program for synchronous processing

The program below shows how to use the OTMA C/I for synchronous (one in-one out) processing.

In this sample program, the `otma_send_receive` API is used to send and receive IMS data.

```

#pragma langlvl(extended)
/*****/
/*
/* Callable Interface sample program using synchronous APIs
/*
/* Parameters:
/*     Server Name
/*     Client Name
/*     User Name
/*     Iterations
/*     Transaction
/*     User Group
/*     OTMA Data
/*
/* Note: The send buffer is sent as a file with a ddname of
/*     SENDBUFn in the invoking JCL.
/*
/* Example: //SENDBUF0 DD *,DLM=$$
/*          SEND OTMA TO SKS1
/*          $$
/*
/* Note: COMPARI is the DDNAME of an input file used to compare
/*     actual output with expected output. '?' is used to delimit
/*     the compare string and '|' is used to ignore a char compare
/*
/* Example: //COMPAR0 DD *,DLM=$$
/*          SEND OTMA TO SKS1?
/*          $$
/*
/*****/

/*****/
/* Entry...
/*
/* This test program is callable from JCL
/*
/* //NA10TMA JOB CLASS=A,MSGLEVEL=(1,1),MSGCLASS=H,REGION=2M
/* //*****
/* /** PARM=server_member_name tpipe_name client_member_name
/* /** iterations command groupid OTMA_Data
/* //MINISAMP EXEC PGM=NA10TMA,
/* // PARM='TRAP(OFF)/IMS61CR1 IMSTESR G214992 1 /DISP groupid
/* // OTMADData'
/* //STEPLIB DD DISP=SHR,DSN=OTMA.TEST.LOAD
/* //SYSUDUMP DD SYSOUT=*
/* //STDOUT DD SYSOUT=*
/* //STDERR DD SYSOUT=*
/* //CEEDUMP DD SYSOUT=*
/* //COMPAR1 DD *,DLM=$$
/* EXPECTED OUTPUT GOES HERE
/* $$
/* //SENDBUF0 DD *,DLM=$$
/* SEND DATA GOES HERE
/* $$
/*
/* Note: TRAP(OFF)/ Passes LE run-time option TRAP(OFF) which turns
/*     off LE condition handling. To get a LE dump on abend set
/*     TRAP ON and provide a CEEDUMP DDNAME.
/*
/* Note: COMPARI is the DDNAME of an input file used to compare
/*     actual output with expected output. '?' is used to delimit
/*     the compare string and '|' is used to ignore a char compare
/*
/*****/

/*****/
/* An example for using the OTMA Client API in C lang.
/*

```



```

/* This program is broken into the following parts:
/*   Declarations for special support
/*   Process invocation parameters
/*   Setup for C signal handling
/*   Do XCF open processing and analysis
/*   Do session allocate processing
/*   Execute a command or transaction per invocation parm
/*   Do session free processing
/*   Do close
/*   End
/*****

/*****
/* API's for non-authorized OTMA caller
/*****
#include "dfsyc0.h"          /* Non-authorized OTMA API's
#include <stdlib.h>          /* Standard C Header file
#include <stddef.h>         /* Standard C Header file
#include <stdio.h>          /* Standard C Header file

/*****
/* Internal functions
/*****
int memc(char *comp_buf, char *rec_buf1 );

/* macro to move string to blank filled left justified char field
#define splat(t,s) \
    {\
        memset((char*)&(t),' ',sizeof(t));\
        strncpy((char*)&(t), s ,strlen(s));}

/* standard math routines
#define min(a,b) ((a)<(b)?(a):(b))
#define max(a,b) ((a)>(b)?(a):(b))

main(int argc,char *argv[])
{
    /*          Following fields used by all Functions
    otma_anchor_t  anchor;          /* Handle returned by create
    /* and used by all others.
    otma_retrsn_t  retrsn;         /* Return code returned by all.
    long int      retsave;        /* Return code save area

    /*          Following fields used by several Functions
    sess_handle_t  sess_handle;    /* Handle returned by allocate
    /* used by send_receive and free.
    otma_grp_name_t  grp_name;     /* API XCF Group Member Name.
    otma_clt_name_t  clt_name;     /* API XCF Client Member Name.
    otma_srv_name_t  srv_name;     /* API XCF Server Member Name.
    /* (the IMS XCF member name).
    racf_uid_t      userid;        /* Our z/OS logon ID.
    racf_prf_t      groupid;       /* RACF Group ID
    otma_user_t     otma_data;     /* Otma Data

    lterm_name_t   lterm;          /* Lterm name
    mod_name_t     modname;        /* ModName

    unsigned char  error_message_text[120]; /* IMS error msg field
    /* A place to receive any IMS
    /* DFS error messages.
    unsigned char *error_message = (unsigned char*)&error_message_text;
    /* a pointer to which is parameter
    /* on send_receive.

```

```

char          *tran;          /* Transaction Name / IMS Command */
tran_name_t  tran_name;     /* Transaction Name / IMS Command */

#define BUFFER_LEN 4096      /* set our buffer sizes          */
#define NUM_BUFFER 60
#define COM_BUFFER 80
#define GROUP_NAME "HARRY"  /* Set XCF group name to join    */

char compare_buf[NUM_BUFFER + 1]; /* Compare buffer                */
int long      buffer_length = 0;
int long      rec_buffer_len = BUFFER_LEN;
char          rec_buf[BUFFER_LEN];
long int      rec_data_len = 0;
char          send_buf[BUFFER_LEN];
char          temp_buf[NUM_BUFFER];

context_t     context = {0x00000000000000000000000000000000};
/* This test is not distributed sync point. */
/* Too complicated for here.                */
/* Normally this is obtained from RRS       */

/*****
/* The callable interface makes use of z/OS Event Control Blocks. */
/* Any language which call the interface must deal with this.      */
*****/

unsigned long *(ecb_list[2]); /* z/OS pause stuff             */
unsigned long **pecb_list;

ecb_t         ecbOPEN = 0L; /* ecb to be posted by OTMA API */
ecb_t         ecbIO   = 0L; /* ecb to be posted by OTMA API */
ecb_t         signal  = 0L; /* ecb to be posted by C runtime */

ecb_t         temp_ecb = 0L; /* used by compare and swap      */
ecb_t         reset_ecb = 0L; /* used by compare and swap      */

/*****
/* Local variables
*****/

int           iterations;
int           loop_count;
int           compare_result;
long int      retcode;

signed long   sessions; /* number of sessions to support */
tpipe_prfx_t tpipe_prefix; /* first part of tpipe NAME      */

FILE * stream;
int num; /* number of characters read from stream */

/*****
/* To support test functions - names of parms
/* Print the parms out for documentation
*****/

char * argdefs[8]={ "pgm name", /* 1 */
                   "server name", /* 2 */
                   "client name", /* 3 */
                   "userid ", /* 4 */
                   "iterations ", /* 5 */
                   "transaction", /* 6 */
                   "group id ", /* 7 */
                   "otma data ", /* 8 */
                   };

```

```

/*****
/* Declare an array of compare file ddnames to          */
/* compare actual output received with expected output. */
/*****

char * infiledd[4]={ "DD:COMPAR0",          /* 1          */
                    "DD:COMPAR1" ,        /* 2          */
                    "DD:COMPAR2" ,        /* 3          */
                    "DD:COMPAR3" ,        /* 4          */
                    };

/*****
/* Declare an array of send file ddnames to          */
/* send application data to OTMA.                   */
/*****

char * sndfiledd[4]= { "DD:SENDBUF0",      /* 1          */
                      "DD:SENDBUF1" ,    /* 2          */
                      "DD:SENDBUF2" ,    /* 3          */
                      "DD:SENDBUF3" ,    /* 4          */
                      };

/* ----- */
/* Announce the startup of the test program.         */
/* ----- */
printf("Otmci01 Starting, version %s %s\n" ,__DATE__,__TIME__ );

/* ----- */
/* z/OS Pause Init - do this first, in case it fails bail out. */
/* This sets up a C environment for signaling from the API.     */
/* ----- */

ecb_list[0] = (unsigned long *) &(signal); /* post by C signal */
ecb_list[1] = (unsigned long *)          /* post by OTMA      */
              ((unsigned long)&(ecbOPEN) |
              (unsigned long)0x80000000); /* end of list      */
pecb_list = &ecb_list[0];              /* pointer to list  */
                                           /* define callable I/F */

/*****
/* Begin Test Case...                                  */
/* Announce the startup of the test program.           */
/*****
printf("OTMCI01 Run Date: %s Run Time: %s\n" ,__DATE__,__TIME__ );

/*****
/* Process parms/command line arguments.              */
/*****

/* First, print the parameters. */
printf("Invocation parameters = \n");
for (i=1 ; i<(min(8,argc));i++)
{
    printf("%d %s = ", i, argdefs[i]);
    printf("%s.\n", argv[i]);
}

if (argc>1) splat( srv_name, argv[1]) /* XCF memname of IMS */
else       splat( srv_name, "IMS61CR1"); /* hard coded default */
if (argc>2) splat( clt_name, argv[2]) /* Client name */
else       splat( clt_name, "XCFTTEST" ); /* hard coded default */
if (argc>3) splat( userid , argv[3]) /* ID to use */
else       splat( userid , "XCFTTEST" ); /* hard coded default */
if (argc>4) iterations = atoi(argv[4]); /* loop count */
else       iterations = 1; /* hard coded default */
if (argc>5) tran = argv[5]; /* Transaction/IMS CMD*/
else       tran = ""; /* hard coded default */

```

```

if (argc>6) splat( groupid, argv[6])      /* Group ID to use */
else       splat( groupid, " " ); /* hard coded default */
if (argc>7) splat( otma_data, argv[7])    /* OTMA Data */
else       splat( otma_data, "" ); /* hard coded default */

/* -----*/
/* Open the file with the ddname SENDBUF0 supplied in the */
/* JCL which invoked this C driver. Then read the file into */
/* temp_buf. */
/* -----*/

if ( ( stream = fopen("DD:SENBUFF0","rb")) != NULL )
{
    num = fread( temp_buf, sizeof( char ), NUM_BUFFER, stream );
    printf("BUFF SIZE = %d.\n", num);
    if (num == NUM_BUFFER) {
        printf( "Number of characters read = %i\n", num );
        fclose( stream );
    }
    else {
        if ( ferror(stream) )
            printf( "Error reading DDNAME sendbuf0/n" );
        else if ( feof(stream)) {
            printf( "EOF found\n" );
            printf( "Number of characters read %d\n", num );
            printf( "temp_buf = %.*s\n", num, temp_buf);
            fclose( stream );
        }
    }
}
else
    printf( "ERROR opening DDNAME sendbuf0/n" );

/* Initialize API parameters and buffers. */
splat( grp_name, GROUP_NAME ); /* XCF Group Name */
splat( tpipe_prefix, "TPAS" ); /* tpipe Prefix Name */
splat( tran_name, tran ); /* do scan here */
strncat(send_buf, temp_buf, num); /* Copy temp_buf into send_buf */
buffer_length = strlen(send_buf); /* Set send buffer length */

/*****
/* Example of setting up parms to Open the XCF Link
*****/

retrsn.ret = -1;
retrsn.rsn[0] = -1;
retrsn.rsn[1] = -1;
retrsn.rsn[2] = -1;
retrsn.rsn[3] = -1;

sessions = 10; /* OTMA supports multiple parallel */
/* sessions (TPIPES) How many do you want?*/

/*****
/*BEGIN:
/* We have a CREATE function to set up storage and
/* an OPEN function to start the protocol.
/* If you do not need to customize the environment you can start
/* with the OPEN function, the CREATE will be done by OPEN.
*****/

printf("-\n");
otma_create(&anchor, /* (out) ptr to addr to receive ancho*/
            &retrsn, /* (out) return code */
            (ecb_t *) &ecbOPEN, /* not posted by create but stored */
            &grp_name, /* (in) ptr to valid groupname */

```

```

        &clt_name,          /* (in) Our member name          */
        &srv_name,         /* (in) Our server name          */

        &sessions,        /* (in) number of sessions to support*/
        &tpipe_prefix     /* (in) first part of tpipe name   */
    );

    printf("OTMA_CREATE issued. ret = %d rsn = %.8x,%.8x,%.8x,%.8x\n"
        " anchor is at %.8x.\n",
        retrsns.ret,
        retrsns.rsn[0],
        retrsns.rsn[1],
        retrsns.rsn[2],
        retrsns.rsn[3],
        anchor);

    printf("-\n");

    /*****
    /* Connect to IMS
    *****/

    otma_open(&anchor,      /* out ptr to addr to receive anchor */
        &retrsns,         /* out return code
        (ecb_t *)&ecbOPEN, /* out posted by open if failure
        /* else posted by exit pgm
        &grp_name,         /* in ptr to valid XCF groupname
        &clt_name,         /* in Our member name
        &srv_name,         /* in Our server name

        &sessions,        /* in number of sessions to support
        &tpipe_prefix     /* in first part of tpipe name
    );

    printf("OTMA_OPEN issued. ret = %.8x rsn = %.8x,%.8x,%.8x,%.8x\n"
        " Waiting for ecb at %.8x.=%.8x.\n",
        retrsns.ret,
        retrsns.rsn[1],
        retrsns.rsn[2],
        retrsns.rsn[3],
        ecb_list[1],
        *ecb_list[1]
    );

    printf("-\n");

    /* -----
    /* Here we wait for Open to signal complete
    /* -----
    DFSYCWAT(ecb_list[1]); /* WAIT on ecb

    printf("OPEN_OTMA done. ret = %.8x rsn = %.8x,%.8x,%.8x,%.8x \n"
        "\nEcb at %.8x.=%.8x.\n",
        retrsns.ret,
        retrsns.rsn[0],
        retrsns.rsn[1],
        retrsns.rsn[2],
        retrsns.rsn[3],
        ecb_list[1], *ecb_list[1]
    );

    printf("Local Area Anchor at %8.8X = %8.8X\n",
        &anchor, anchor);

    printf("-\n");

```

```

/* -----*/
/* The post code from open indicates success or failure */
/* -----*/
if (0!=(0x00ffffff & ecbOPEN))
{
    printf("OPEN_OTMA ecb is posted failure.\n");
    return(retrsn.rsn[0]);
}

/* -----*/
/* Set userid to blanks if userid = bobdavis */
/* -----*/

printf(" Trans = %.8s,\n ", tran_name );
printf(" Userid = %.8s,\n ", userid );
printf("Groupid = %.8s,\n ", groupid );

/*****/
/* Like CREATE the ALLOC function just creates control blocks */
/* and stores data in them. Other functions may be invented */
/* to modify these structures before the command-of-execution,*/
/* SEND_RECEIVE is issued. */
/*****/

otma_alloc(
    &anchor,          /* in ptr to global word */
    &retrsn,         /* out rc,reason(1-4) */

    &sess_handle,    /* out session id */
    NULL,            /* in default overrides */

    &tran_name,      /* in IMS tp name or cmd */
    &userid,         /* in RACFid or blanks */
    &groupid         /* in RACF group id or blnk*/
);

printf("OTMA_ALLOC done. ret = %.8x rsn = %.8x,%.8x,%.8x,%.8x\n",
    retrsn.ret,
    retrsn.rsn[0],
    retrsn.rsn[1],
    retrsn.rsn[2],
    retrsn.rsn[3]
);

/*****/
/* Even if ALLOC fails we go on here just to prove the */
/* API will reject the call. */
/*****/

/*****/
/* This is the call that sends the data and prepares to */
/* receive the answer from IMS. */
/* */
/* This test program can iterate with multiple calls here. */
/*****/

/* __Send message wait for reply_____ */
for (loop_count = 0 ; loop_count<iterations ; loop_count++)
{
    /* __Change the environment to wait for ecbIO */
    ecbIO = 0; /* clear ecb for reuse */
    ecb_list[1] = (unsigned long *) /* posted by OTMA */
        ((unsigned long)&(ecbIO) |
        (unsigned long)0x80000000); /* end of list */

    if (loop_count != 0)
    {

```

```

/* -----*/
/* If looping more than once open the next file to send */
/* and read it into the send_buf. */
/* -----*/

if (( stream = fopen(sndfiledd[loop_count],"rb")) != NULL )
{
    num = fread( temp_buf, sizeof( char ), NUM_BUFFER, stream );
    printf("BUFF SIZE = %d.\n", num);
    if (num == NUM_BUFFER) {
        fclose( stream );
    }
    else {
        if ( ferror(stream) )
            printf( "Error opening file
else if ( feof(stream)) {
                printf( "EOF found\n" );
                printf( "Number of characters read %d\n", num );
                printf( "temp_buf = %.*s\n", num, temp_buf);
                fclose( stream );
            }
        }
    }
}
else
    printf( "Error opening file %s\n", sndfiledd[loop_count]);
/* Initialize send and receiving buffers. */
memset(rec_buf ,0, sizeof(rec_buf));
memset(send_buf ,0, sizeof(send_buf));
strcat(send_buf, temp_buf );
strcat(send_buf, " " );
buffer_length = strlen(send_buf);
printf("
printf( "buffer length = %d\n", buffer_length);
} /* end if loop_count != 0 */

/* Print otma_send_receive parms and start of API */
memset(error_message_text ,0, sizeof(error_message_text));
printf("Send buf at %.8x.\n", &send_buf);
printf("Send buf = %s.\n", send_buf);
printf("Receive buf at %.8x.\n", &rec_buf);
printf("Lterm = %.8s.\n", lterm );
printf("Modname = %.8s.\n", modname );

printf("-\n");
otma_send_receive(
    &anchor,          /* (in) anchor block */
    &retrsn,         /* (out) return status */
    &ecbI0,          /* (in) ecb address */

    &sess_handle,   /* (in) session handle */
    &lterm,         /* (in/out) logical terminal */
    &modname,       /* (in/out) module name */

(unsigned char *) &send_buf, /* (in) send buffer */
    &buffer_length, /* (in) size of send buffer */
    0,              /* (in) send_segment_list */

(unsigned char *) &rec_buf, /* (in) receive buffer */
    &rec_buffer_len, /* (in) size of buffer */
    &rec_data_len,  /* (out) received data length */
    0,              /* (in/out) receive seg list */

    &context,       /* (in) context id */
    &error_message, /* (out) ims message */
    &otma_data);   /* (in) Otma Data */

```

```

printf("OTMA_SEND done. ret = %.8x rsn = %.8x,%.8x,%.8x,%.8x\n",
      retrs_n.ret,
      retrs_n.rsn[0],
      retrs_n.rsn[1],
      retrs_n.rsn[2],
      retrs_n.rsn[3]);

/* ----- */
/* Here we wait for receive to signal complete */
/* An application can go do other thing while IMS is processing and */
/* while the XCF scheduled SRBs are returning data to the caller's */
/* buffers. DO NOT DEALLOCATE THE BUFERS WHILE THIS IS GOING ON! */
/* None of the output areas of the SEND_RECEIVE can be freed until */
/* the ECB is posted complete. */
/* ----- */

DFSVCWAT(ecb_list[1]);      /* WAIT on ecb */

retsave = retrs_n.ret;     /* Save Receive return code */

printf("OTMA_RECEIVE done. ret = %.8x rsn = %.8x,%.8x,%.8x,%.8x\n"
      "\nEcb at %.8x.= %.8x.\n",
      retrs_n.ret,
      retrs_n.rsn[0],
      retrs_n.rsn[1],
      retrs_n.rsn[2],
      retrs_n.rsn[3],
      ecb_list[1],
      *ecb_list[1]
      );

if (retrs_n.ret != 0)
{
    /* ___Error path Free allocated session _____ */
    printf("-error path retrs_n.ret="
          "\n");
    printf("Error message = %s\n", error_message );
    otma_free(
        & anchor,      /* (out) ptr to global word */
        & retrs_n,    /* (out) rc,reason (1-4) */
        & sess_handle /* (in) unique path id */
    );

    printf("OTMA_FREE done. ret = %.8x rsn = %.8x,%.8x,%.8x,%.8x \n",
          retrs_n.ret,
          retrs_n.rsn[0],
          retrs_n.rsn[1],
          retrs_n.rsn[2],
          retrs_n.rsn[3]
    );

    /* ___Sever IMS connection _____ */
    printf("-\n");
    otma_close(
        & anchor,      /* (in,out) tr to otma anchor */
        & retrs_n      /* (out) rc,reason (1-4) */
    );

    printf("OTMA_CLOSE done. ret = %.8x rsn = %.8x,%.8x,%.8x,%.8x\n",
          retrs_n.ret,
          retrs_n.rsn[0],
          retrs_n.rsn[1],
          retrs_n.rsn[2],
          retrs_n.rsn[3]
    );
}

```



```

        return (retsave);      /* EXIT with receive API return code */
    }

    /* -----*/
    /* If SEND_RECEIVE worked .. */
    /* -----*/

    /* -----*/
    /* Open the compare file containing the expected output */
    /* of the receive buffer. Compare the expected output */
    /* with the actual output and return the result. */
    /* -----*/

    rec_buf[0] = ' ';          /* Remove possible NL ie x'15' */
    printf( "infiledd = %s\n", infiledd[loop_count] );

    if (( stream = fopen(infiledd[loop_count],"rb")) != NULL )
    {
        num = fread( compare_buf, sizeof( char ), COM_BUFFER, stream );
        if (num == COM_BUFFER) { /* fread success */
            printf( "compare_buf = %s\n", compare_buf );
            printf( "    rec_buf = %s\n", rec_buf );
            fclose( stream );
            compare_result = memcmp( compare_buf, rec_buf );
            printf( "compare_result = "
                if (compare_result != 0)
                    return(compare_result);          /* Exit if NO COMPARE */
            }
        }
        else { /* fread() failed */
            if ( ferror(stream) )          /* possibility 1 */
                printf( "Error reading file %s\n", infiledd[loop_count]);
            else if ( feof(stream) ) {     /* possibility 2 */
                printf( "EOF found\n" );
                printf( "Number of characters read %d\n", num );
                printf( "compare_buf = %.*s\n", num, compare_buf);
            }
        }
    }
    else
        printf( "Error opening file %s\n", infiledd[loop_count]);
}
    /* end of loop */

    /*****
    /* Once a message is sent to IMS and the answer received it */
    /* is usual to release the tpipe for use by other transactions. */
    /* For conversational trans an application would keep using */
    /* the handle to continue a conversational transaction with IMS. */
    /* The Transaction name is specified in the ALLOC and it is */
    /* intended that a FREE be done at the end of each transaction */
    /* and a new ALLOC be done for the next one. This is not */
    /* expensive. */
    /*****/

    printf("-\n");
    otma_free(
        & anchor,          /* (out) ptr to global word */
        & retrsn,          /* (out) rc,reason (1-4) */
        & sess_handle     /* (in)  unique path id */
    );

    printf("OTMA_FREE done. ret = %.8x rsn = %.8x,%.8x,%.8x,%.8x \n",
        retrsn.ret,
        retrsn.rsn[0],
        retrsn.rsn[1],
        retrsn.rsn[2],
        retrsn.rsn[3]
    );

```

```

        );

    printf("-\n");

    /*
    /* Finally, CLOSE severs the connection with IMS and frees the
    /* Storage used by the OTMA API.
    /* This will be done at job-step termination but its untidy.
    /*
    /*

    otma_close(
                & anchor,      /* (in,out) ptr to otma anchor */
                & retrsn      /* (out) rc,reason (1-4) */
    );
    printf("OTMA_CLOSE done. ret = %.8x rsn = %.8x,%.8x,%.8x,%.8x \n",
        retrsn.ret,
        retrsn.rsn[0],
        retrsn.rsn[1],
        retrsn.rsn[2],
        retrsn.rsn[3]
    );

    return (compare_result);      /* Return return code */
} /* end of main */

/*=====*/
/* Subroutine to compare expected results(compare_buf)
/* with actual results(err_msg) the "|" is used to signify
/* an ignore compare and "?" is used to mark the end of string.
/* Note: Compare starts using an index i=1 ie. the 2nd character
/* because the 1st character was blanked out. ( NL x'15' )
/*=====*/

int memc(char *comp_buf, char *rec_buf1)
{

    int j;
    int i;

    j = 0;

    for (i=1;
        ( (j==0) && (comp_buf[i] != '?' ) );
        i++)
    {
        if( comp_buf[i] != '|' )          /* Ignore compare */
        {
            if( comp_buf[i] != rec_buf1[i] ) /* compare ok ? */
            {
                j++;                      /* No */
                printf( "MISCOMPARE !!! \n" );
                printf( "comp_buf[%d] = %c\n", i, comp_buf[i] );
                printf( "rec_buf1[%d] = %c\n", i, rec_buf1[i] );
            }
            else
                ;
        }
        else
            ;
        /* Else null */
    }

    return (j);
}

```

**Related reference:**

“OTMA C/I hints and tips” on page 553

## OTMA C/I sample program for asynchronous processing

The following program illustrates how to use OTMA C/I for asynchronous (unsolicited) processing.

In this sample program, one `otma_send_asynch` and one `otma_receive_asynch` call is issued per loop.

**Recommendation:** If you will be using synchronous (one in-one out) processing exclusively, use the `otma_send_receive` API. The `otma_send_receive` API provides the most efficient means of synchronous processing.

```
#pragma langlvl(extended)

/*****
/*
/* Callable Interface sample program using asynchronous APIs
/*
/* Parameters:
/* Server Name
/* Client Name
/* Transaction
/* User Name
/* User Group
/* Lterm
/* Mod Name
/* OTMA Data
/* Iterations
/*
/* Note: The send buffer is sent as a file with a ddname of
/* SENDBUFn in the invoking JCL.
/*
/* Example: //SENBUF0 DD *,DLM=$$
/* SEND OTMA TO SKS1
/* $$
/*
/* Note: COMPAR1 is the DDNAME of an input file used to compare
/* actual output with expected output. '?' is used to delimit
/* the compare string and '|' is used to ignore a char compare
/*
/* Example: //COMPAR0 DD *,DLM=$$
/* SEND OTMA TO SKS1?
/* $$
/*
/* Note: TPIPEBUFn is the DDNAME of an input file used to specify
/* the tpipe name to be used for each iteration.
/*
/* Example: //TPIPEBUF0 DD *,DLM=$$
/* TPIPE001
/* $$
/*
*****/

/*****
/* Entry...
/*
/* This test program is callable from JCL
/*
/* //NA10TMA JOB CLASS=A,MSGLEVEL=(1,1),MSGCLASS=H,REGION=2M
/* /**/
/* /** PARM=server_member_name client_member_name transaction
/* /** user_name group_name lterm_name ModName OTMA_Data
/* /** iterations
/* /**/
/* //MINISAMP EXEC PGM=NA10TMA,
/* // PARM='TRAP(OFF)/IMS61CR1 IMSTESR G214992 /DISP user01 groupid */
```

```

/* //          Lterm ModName OTMADData 1'          */
/* //STEPLIB DD DISP=SHR,DSN=OTMA.TEST.LOAD      */
/* //SYSUDUMP DD SYSOUT=*                        */
/* //STDOUT DD SYSOUT=*                         */
/* //STDERR DD SYSOUT=*                         */
/* //CEEDUMP DD SYSOUT=*                       */
/* //COMPAR1 DD *,DLM=$$                       */
/* EXPECTED OUTPUT GOES HERE                   */
/* $$                                           */
/* //SENDBUF0 DD *,DLM=$$                      */
/* SEND DATA GOES HERE                       */
/* $$                                           */
/* //TPIPBUF0 DD *,DLM=$$                     */
/* TPIPE NAME GOES HERE                       */
/* $$                                           */
/*
/* Note: TRAP(OFF)/ Passes LE run-time option TRAP(OFF) which turns
/* off LE condition handling. To get a LE dump on abend set
/* TRAP ON and provide a CEEDUMP DDNAME.
/*
/* Note: COMPAR1 is the DDNAME of an input file used to compare
/* actual output with expected output. '?' is used to delimit
/* the compare string and '|' is used to ignore a char compare
/*
/*****/
/*****/
/* An example for using the OTMA Client API in C lang.
/* This program is broken into the following parts:
/* Declarations for special support
/* Process invocation parameters
/* Setup for C signal handling
/* Do XCF open processing and analysis
/* Execute an API to send data per invocation parm
/* Execute an API to receive data per invocation parm
/* Do close
/* End
/*****/
/*****/
/* Header Definitions.
/*****/
#include "dfsyc0.h"          /* Non-authorized OTMA API's
#include <stdlib.h>         /* Standard C Header file
#include <stddef.h>        /* Standard C Header file
#include <stdio.h>         /* Standard C Header file

/*****/
/* Internal functions
/*****/
/* memory comparison macro.
/* int memc(char *comp_buf, char *rec_buf1 );
/*
/*
/* macro to move string to blank filled left justified char field
/* #define splat(t,s) \
/* { \
/* memset((char*)&(t),' ',sizeof(t));\
/* strncpy((char*)&(t), s ,strlen(s));\
/*
/* standard math routines
/* #define min(a,b) ((a)<(b)?(a):(b))
/* #define max(a,b) ((a)>(b)?(a):(b))

/*****/
/*
/* This OTMA C/I Program

```

```

/*
/* Note: TRAP(OFF)/ Passes LE run-time option TRAP(OFF) which turns
/* off LE condition handling. To get a LE dump on abend set
/* TRAP ON and provide a CEEDUMP DDNAME.
/*
/* Note: COMPARI is the DDNAME of an input file used to compare
/* actual output with expected output. '?' is used to delimit
/* the compare string and '|' is used to ignore a char compare
/*
/*****
main(int argc,char *argv[])
{

/*****
/* Fields used by OTMA C/I APIs.
/*****

/* The following fields used by all the OTMA C/I API's.

otma_anchor_t    anchor;        /* Handle returned by create
/* and used by all others.
otma_retrsn_t    retrsn;        /* Return code returned by all.

/* The following fields are used by the otma_create and
/* otma_open API's.

otma_grp_name_t  grp_name;      /* API XCF Group Member Name.
otma_clt_name_t  clt_name;      /* API XCF Client Member Name.
otma_srv_name_t  srv_name;      /* API XCF Server Member Name.
/* (IMS's XCF member name).
signed long      sessions;      /* number of sessions to support
tpipe_prfx_t     tpipe_prefix;  /* first part of tpipe NAME
/* The following fields are used by otma_send_async API.

tpipe_name_t     tpipe;         /* User tpipe Name.
tran_name_t      trans;         /* IMS Trancode or CMD.
racf_uid_t       user_name;     /* RACF UserID.
racf_prf_t       user_prf;      /* RACF Groupname.
lterm_name_t     lterm;         /* Input Lterm.
mod_name_t       modname;       /* Input Modname.
otma_user_t      otma_data;     /* OTMA Userdata.
char             send_buf[BUFFER_LEN];
int long         buffer_length = 0; /* Send Buffer length.
unsigned char error_message_text[120]; /* IMS error msg field -
/* A place to receive any IMS
/* DFS error messages.
unsigned char *error_message = (unsigned char*)&error_message_text;
/* a pointer to which is parameter
/* on send_receive.
otma_profile2_t  send_options;  /* Send Special Options.

/* The following fields are used by otma_receive_async API.

lterm_name_t     rec_lterm;     /* Output Lterm.
mod_name_t       rec_modname;   /* Output Modname.
otma_user_t      rec_otma_data; /* OTMA Userdata.
char             rec_buf[BUFFER_LEN];
int long         rec_buffer_len = BUFFER_LEN;
long int         rec_data_len = 0;
otma_profile3_t  rec_options;   /* Receive Special Options.

/*****
/* The callable interface makes use of z/OS Event Control Blocks.
/* Any language which call the interface must deal with this.
/*****

unsigned long *(ecb_list[2]);    /* z/OS pause ecb list

```

```

unsigned long **pecb_list;

ecb_t      ecbOPEN  = 0L;    /* ecb to be posted by OTMA API */
ecb_t      ecbIO    = 0L;    /* ecb to be posted by OTMA API */
ecb_t      signal   = 0L;    /* ecb to be posted by C runtime */

ecb_list[0] = (unsigned long *) &(signal); /* post by C signal */
ecb_list[1] = (unsigned long *)           /* post by OTMA */
              ((unsigned long)&(ecbOPEN) |
               (unsigned long)0x80000000); /* end of list */
pecb_list   = &ecb_list[0];           /* pointer to list */
                                                /* define callable I/F */

/*****
/* Local Variables */
*****/

long int    retsave;          /* Return code save area */
int         iterations;      /* Number of iterations to use */
int         loop_count;      /* Number of iterations used */
int         compare_result;  /* Return Code result of the
                             /* comparison for buffers.

/*****
/* Local Constants */
*****/

#define BUFFER_LEN 4096      /* Set our buffer sizes */
#define NUM_BUFFER 80       /* Set the number of buffers */
#define GROUP_NAME "HARRY"  /* Set XCF group name to join */
char temp_buf[NUM_BUFFER];  /* Swapping buffer */
char compare_buf[NUM_BUFFER + 1]; /* Compare buffer */
FILE * stream;
int num;                    /* number of characters read from stream */

/*****
/* To support test functions - names of parms in order to print
/* the parms out for documentation.
*****/

char * argdefs[10]={"Program Name", /* 1 */
                  "Server Name", /* 2 */
                  "Client Name", /* 3 */
                  "Transaction", /* 4 */
                  "User Name ", /* 5 */
                  "User Group ", /* 6 */
                  "Lterm ", /* 7 */
                  "Mod Name ", /* 8 */
                  "OTMA Data ", /* 9 */
                  "Iterations ", /* 10 */
                  };

/*****
/* Declare an array of compare file ddnames to
/* compare actual output received with expected output.
*****/

char * infiledd[4]={"DD:COMPAR0", /* 1 */
                   "DD:COMPAR1", /* 2 */
                   "DD:COMPAR2", /* 3 */
                   "DD:COMPAR3", /* 4 */
                   };

/*****
/* Declare an array of send file ddnames to
/* send application data to OTMA.
*****/

```

```

char * sndfiledd[4]= {"DD:SENDBUF0",      /* 1          */
                    "DD:SENDBUF1" ,    /* 2          */
                    "DD:SENDBUF2" ,    /* 3          */
                    "DD:SENDBUF3" ,    /* 4          */
                    };

/*****
/* Declare an array of tpipe names dnames for the
/* otma_send_async API.
*****/

char * tpipefiledd[4]= {"DD:TPIPBUF0",   /* 1          */
                      "DD:TPIPBUF1" ,  /* 2          */
                      "DD:TPIPBUF2" ,  /* 3          */
                      "DD:TPIPBUF3" ,  /* 4          */
                      };

/*****
/* Begin Test Case...
/* Announce the startup of the test program.
*****/
printf("OTMCI02 Run Date: %s Run Time: %s\n" ,__DATE__,__TIME__ );

/*****
/* Process parms/command line arguments.
/*
/* Note: If not a parameter is not used, then "NONE" is used in
/* its place.
/*
*****/

/* First, print the parameters. */
printf("Invocation parameters = \n");
for (i=1 ; i<(min(11,argc));i++)
{
    printf("%d %s = ", i, argdefs[i]);
    printf("%s.\n", argv[i]);
}

printf("\n");

if (argc>1 && strcmp(argv[1],"NONE") != 0)
    splat( srv_name, argv[1])          /* Server Name.      */
else
    splat( srv_name, "IMS61CR1");      /* Hard coded default */
if (argc>2 && strcmp(argv[2],"NONE") != 0)
    splat( clt_name, argv[2])          /* Client name       */
else
    splat( clt_name, "XCFTTEST" );    /* Hard coded default */
if (argc>3 && strcmp(argv[3],"NONE") != 0)
    splat( trans, argv[3])             /* IMS Tran/Cmd to use*/
else
    splat( trans, "");                 /* Hard coded default */
if (argc>4 && strcmp(argv[4],"NONE") != 0)
    splat( user_name, argv[4])         /* RACF Username     */
else
    splat( user_name, "");             /* Hard coded default */
if (argc>5 && strcmp(argv[5],"NONE") != 0)
    splat( user_prf, argv[5])          /* RACF Group ID     */
else
    splat( user_prf, "" );             /* Hard coded default */
if (argc>6 && strcmp(argv[6],"NONE") != 0)
    splat( lterm , argv[6])            /* Lterm to use      */
else
    splat( lterm , "" );               /* Hard coded default */

```

```

if (argc>7 && strcmp(argv[7],"NONE") != 0)
    splat( modname , argv[7])          /* ModName to use */
else
    splat( modname , "" );            /* Hard coded default */
if (argc>8 && strcmp(argv[8],"NONE") != 0)
    splat( otma_data, argv[8])        /* OTMAData to use */
else
    splat( otma_data, "" );           /* Hard coded default */
if (argc>9 && strcmp(argv[9],"NONE") != 0)
    iterations = atoi(argv[9]);       /* Loop count */
else
    iterations = 1;                   /* Hard coded default */

/* -----*/
/* Open the file with the ddname SENDBUF0 supplied in the */
/* JCL which invoked this C driver. Then read the file into */
/* temp_buf. */
/* -----*/

if ( ( stream = fopen("DD:SENBUFF0","rb")) != NULL )
{
    num = fread( temp_buf, sizeof( char ), NUM_BUFFER, stream );
    if (num == NUM_BUFFER) {
        printf( "Number of characters read = %i\n", num );
        fclose( stream );
    }
    else {
        if ( ferror(stream) )
            printf( "Error reading DDNAME sendbuf0/n");
        else if ( feof(stream)) {
            printf( "EOF found\n" );
            printf( "Number of characters read %d\n", num );
            printf( "temp_buf = %.*s\n", num, temp_buf);
            fclose( stream );
        }
    }
}
else
    printf( "ERROR opening DDNAME sendbuf0/n" );

/*-----*/
/* Initialize parameters for the otma_create and otma_open */
/* APIs. */
/*-----*/

splat( grp_name,GROUP_NAME );         /* XCF Group Name */
splat( tpipe_prefix,"TPAS" );         /* XCF Group Name */
strcat(send_buf, temp_buf );          /* Copy temp_buf into send_buf */
strcat(send_buf, " ");                /* add a blank for strlen */
buffer_length = strlen(send_buf);

/*****
/* Example of setting up parms to Open the XCF Link */
*****/

retrsn.ret    = -1;
retrsn.rsn[0] = -1;
retrsn.rsn[1] = -1;
retrsn.rsn[2] = -1;
retrsn.rsn[3] = -1;
r             = 0;
sessions      = 10; /* OTMA supports multiple parallel */
                /* sessions (TPIPES) How many do you want?*/

/*****
/*BEGIN: */
/* We have a CREATE function to set up storage and */

```



```

/* an OPEN function to start the protocol. */
/* If you don't need to customize the environment you can start */
/* with the OPEN function, the CREATE will be done by OPEN. */
/*****/

otma_create(&anchor, /* (out) ptr to addr to receive ancho*/
            &retrsn, /* (out) return code */
            (ecb_t *) &ecbOPEN, /* not posted by create but stored */

            &grp_name, /* (in) ptr to valid groupname */
            &clt_name, /* (in) Our member name */
            &srv_name, /* (in) Our server name */

            &sessions, /* (in) number of sessions to support*/
            &tpipe_prefix /* (in) first part of tpipe name */
            );

printf("OTMA_CREATE issued. ret = %d rsn = %.8x,%.8x,%.8x,%.8x\n"
      " anchor is at %.8x.\n",
      retrsn.ret,
      retrsn.rsn[0],
      retrsn.rsn[1],
      retrsn.rsn[2],
      retrsn.rsn[3],
      anchor);

printf("-\n");

/*****/
/* Time to try to connect to IMS */
/*****/

/* __start XCF connection_____ */

otma_open(&anchor, /* out ptr to addr to receive anchor */
          &retrsn, /* out return code */
          (ecb_t *)&ecbOPEN, /* out posted by open if failure */
          /* else posted by exit pgm */
          &grp_name, /* in ptr to valid XCF groupname */
          &clt_name, /* in Our member name */
          &srv_name, /* in Our server name */

          &sessions, /* in number of sessions to support */
          &tpipe_prefix /* in first part of tpipe name */
          );

printf("OTMA_OPEN issued. ret = %.8x rsn = %.8x,%.8x,%.8x,%.8x\n"
      " Waiting for ecb at %.8x.=%.8x.\n",
      retrsn.ret,
      retrsn.rsn[1],
      retrsn.rsn[2],
      retrsn.rsn[3],
      ecb_list[1],
      *ecb_list[1]
      );

printf("-\n");

/* ----- */
/* Here we wait for Open to signal complete */
/* ----- */
DFSYSWAT(ecb_list[1]); /* WAIT on ecb */

printf("OTMA_OPEN done. ret = %.8x rsn = %.8x,%.8x,%.8x,%.8x \n"
      "\nEcb at %.8x.=%.8x.\n",

```

```

        retrsн.ret,
        retrsн.rsn[0],
        retrsн.rsn[1],
        retrsн.rsn[2],
        retrsн.rsn[3],
        ecb_list[1], *ecb_list[1]
    );

    printf("Local Area Anchor at %8.8X = %8.8X\n",
           &anchor, anchor);

/* -----*/
/* The post code from open indicates success or failure */
/* -----*/
    if (0!=(0x00ffffff & ecbOPEN))
    {
        printf("OPEN_OTMA ecb is posted failure.\n");
        return(retrsн.rsn[0]);
    }

/*-----*/
/* This is the loop that sends and receives data. */
/* */
/* This test program can iterate with multiple calls here. */
/*-----*/

for (loop_count = 0 ; loop_count<iterations ; loop_count++)
{

    /* Change the environment to wait for ecbIO */
    ecbIO = 0; /* clear ecb for reuse */
    ecb_list[1] = (unsigned long *) /* posted by OTMA */
                  ((unsigned long)&(ecbIO) |
                   (unsigned long)0x80000000); /* end of list */

    if (loop_count != 0)
    {

        /* -----*/
        /* If looping more than once open the next file to send */
        /* and read it into the send_buf. */
        /* -----*/

        if (( stream = fopen(sndfiledd[loop_count],"rb")) != NULL )
        {
            num = fread( temp_buf, sizeof( char ), NUM_BUFFER, stream );
            if (num == NUM_BUFFER) {
                fclose( stream );
            }
            else {
                if ( ferror(stream) )
                    printf( "Error opening file
                    else if ( feof(stream)) {
                        printf( "EOF found\n" );
                        printf( "Number of characters read %d\n", num );
                        printf( "temp_buf = %.*s\n", temp_buf);
                        fclose( stream );
                    }
                }
            }
        }
        else
            printf( "Error opening file %s\n", sndfiledd[loop_count]);

        /* Put data in to Send Buffer. */
        memset(error_message_text ,0, sizeof(error_message_text));

```

```

memset(send_buf ,0, sizeof(send_buf));
strcat(send_buf, temp_buf );
strcat(send_buf, " " );
buffer_length = strlen(send_buf);

} /* end if loop_count != 0 */

/* -----*/
/* If looping more than once open the next tpipe to use */
/* and read it into the tpipe. */
/* -----*/

if (( stream = fopen(tpipefiledd[loop_count],"rb")) != NULL )
{
    num = fread( temp_buf, sizeof( char ), NUM_BUFFER, stream );
    if (num == NUM_BUFFER) {
        fclose( stream );
    }
    else {
        if ( ferror(stream) )
            printf( "Error opening file
else if ( feof(stream)) {
                printf( "EOF found\n" );
                printf( "Number of characters read %d\n", num );
                printf( "temp_buf = %.*s\n", temp_buf);
                fclose( stream );
            }
        }
    }
}
else
    printf( "Error opening file %s\n", sndfiledd[loop_count]);

memcpy(tpipe, temp_buf, 8);

/* Print announcement of send API. */
printf("-\n-\n- Iteration # %d Send API ----- \n-\n",
        loop_count+1);
printf("tpipe Name = %.8s.\n", tpipe);
printf("Transaction = %.8s.\n", trans);
printf("RACF UserID = %.8s.\n", user_name);
printf("RACF Group = %.8s.\n", user_prf);
printf("Lterm = %.8s.\n", lterm );
printf("Modname = %.8s.\n", modname );
printf("OTMA Data = %.50s.\n", otma_data );
printf("Send buf = %s.\n", send_buf);
printf("Send buf at %.8x.\n", &send_buf);
printf ("Buffer length = %d.\n", buffer_length);
printf ("Waiting for ecb at %.8x.=%.8x.\n", ecb_list[1],
        *ecb_list[1]);

otma_send_async(
    &anchor,          /* (in) anchor block */
    &retrsn,         /* (out) return status */
    &ecbIO,          /* (out) ecb address */

    &tpipe,         /* (in) user tpipe name */
    &trans,         /* (in) IMS tranocode or cmd */
    &user_name,     /* (in) RACF userid */
    &user_prf,     /* (in) RACF group name */
    &lterm,         /* (in) logical terminal */
    &modname,      /* (in) module name */
    &otma_data,    /* (in) OTMA user data */

(unsigned char *) &send_buf, /* (in) send buffer */
    &buffer_length, /* (in) size of send buffer */
    0,              /* (in) send_segment_list */

```

```

        &error_message, /* (out) IMS Error msg. */
        &send_options); /* (in) send special options */

DFSVCWAT(ecb_list[1]); /* WAIT on ecb */

/* Print results of send API. */
printf("OTMA_SEND_ASYNC done. ret = %.8x rsn = %.8x,%.8x,%.8x,%.8x\n"
      "Ecb at %.8x,%.8x.\n",
      retrsн.ret,
      retrsн.rsn[0],
      retrsн.rsn[1],
      retrsн.rsn[2],
      retrsн.rsn[3],
      ecb_list[1],
      *ecb_list[1]
);

retsave = retrsн.ret; /* Save otma_send_async Return Code. */

/* Error Processing for OTMA_SEND_ASYNC API. */
if (retrsн.ret != 0)
{
    /* ___Error path Free allocated session _____ */
    printf("-Error send_async API retrsн.ret="
          printf( "Error message = %s\n", error_message );

if (( stream = fopen(infiledd[loop_count],"rb")) != NULL )
{
    num = fread( compare_buf, sizeof( char ), NUM_BUFFER, stream );
    if (num == NUM_BUFFER) { /* fread success */
        printf( "Compare_buf = %.80s.\n", compare_buf );
        printf( "Error_buf = %.80s.\n", error_message );
        fclose( stream );
        compare_result = memc( compare_buf, error_message );
        printf("compare_result =
if (compare_result != 0)
return(compare_result); /* Exit if NO COMPARE */
}
else { /* fread() failed */
    if ( ferror(stream) ) /* possibility 1 */
        printf( "Error reading file %s\n", infiledd[loop_count]);
    else if ( feof(stream) ) { /* possibility 2 */
        printf( "EOF found\n" );
        printf( "Number of characters read %d\n", num );
        printf( "Receive compare_buf = %.*s\n", num, compare_buf);
    }
}
}
else
printf( "Error opening file %s\n", infiledd[loop_count]);

printf("-\n");

/* ___Sever IMS connection _____ */
printf("-\n");
otma_close(
    & anchor, /* (in,out) tr to otma anchor */
    & retrsн /* (out) rc,reason (1-4) */
);

printf("OTMA_CLOSE done. ret = %.8x rsn = %.8x,%.8x,%.8x,%.8x\n",
      retrsн.ret,
      retrsн.rsn[0],
      retrsн.rsn[1],
      retrsн.rsn[2],

```

```

        retrsn.rsn[3]
    );

    return (retsave);    /* EXIT with receive API return code */
}

/* Initialize otma_receive_async parameters.          */
splat( rec_lterm , "" );
splat( rec_modname , "" );
splat( rec_otma_data , "" );
ecbIO = 0;          /* clear ecb for reuse          */
ecb_list[1] = (unsigned long *) /* posted by OTMA          */
((unsigned long)&(ecbIO) |
 (unsigned long)0x80000000); /* end of list          */

/* Print announcement of receive API.          */
printf("-\n-\n- Iteration # %d Receive API ----- \n-\n",
    loop_count+1);
printf("tpipe Name = %.8s.\n", tpipe);
printf("Waiting for ecb at %.8x=%.8x.\n", ecb_list[1],
    *ecb_list[1]);

otma_receive_async(
    &anchor,          /* (in) anchor block          */
    &retrsn,         /* (out) return status        */
    &ecbIO,          /* (out) ecb address          */
    &tpipe,          /* (in) user tpipe name      */
    &rec_lterm,      /* (in) logical terminal      */
    &rec_modname,    /* (in) module name          */
    &rec_otma_data, /* (in) OTMA user data       */
    (unsigned char *) &rec_buf, /* (out) Receive buffer      */
    &rec_buffer_len, /* (in) size of rec buffer    */
    &rec_data_len,  /* (in) send_segment_list    */
    0,              /* (in/out) rec multiple seg */
    &rec_options); /* (in) rec special options  */

DFSYCWAT(ecb_list[1]); /* WAIT on ecb          */
/* Print results of receive API.          */
printf("OTMA_REC_ASYNC done. ret = %.8x rsn = %.8x,%.8x,%.8x,%.8x\n"
    "Ecb at %.8x=%.8x.\n",
    retrsn.ret,
    retrsn.rsn[0],
    retrsn.rsn[1],
    retrsn.rsn[2],
    retrsn.rsn[3],
    ecb_list[1],
    *ecb_list[1]);
printf("Lterm = %.8s.\n", rec_lterm );
printf("Modname = %.8s.\n", rec_modname );
printf("OTMA Data = %.50s.\n", rec_otma_data );
printf("Receive buf = %.80s.\n", rec_buf);
printf("Receive buf at %.8x.\n", &rec_buf);
printf("Data length = %d.\n", rec_data_len);
printf("Buffer length = %d.\n", rec_buffer_len);

retsave = retrsn.ret; /* Save otma_receive_async Return Code. */

/* Error Processing for OTMA_RECEIVE_ASYNC API.          */
if (retrsn.ret != 0)
{
    /* ___Error path Free allocated session _____ */
    printf("-error path retrsn.ret=");
    printf("-\n");
}

```

```

/* ___Sever IMS connection _____ */
printf("-\n");
otma_close(
    & anchor,      /* (in,out) tr to otma anchor */
    & retrsns     /* (out) rc,reason (1-4)      */
);

printf("OTMA_CLOSE done. ret = %.8x rsn = %.8x,%.8x,%.8x,%.8x\n",
    retrsns.ret,
    retrsns.rsn[0],
    retrsns.rsn[1],
    retrsns.rsn[2],
    retrsns.rsn[3]
);

return (retsave);    /* EXIT with receive API return code */
}

/* -----*/
/* Open the compare file containing the expected output      */
/* of the receive buffer. Compare the expected output      */
/* with the actual output and return the result.            */
/* -----*/

printf("-\n-\n- Iteration #d Data Validation ----- \n-\n",
    loop_count+1);

if (( stream = fopen(infiledd[loop_count],"rb")) != NULL )
{
    num = fread( compare_buf, sizeof( char ), NUM_BUFFER, stream );
    if (num == NUM_BUFFER) { /* fread success */
        printf( "compare_buf = %.80s.\n", compare_buf );
        printf( " rec_buf = %.80s.\n", rec_buf );
        fclose( stream );
        compare_result = memcmp( compare_buf, rec_buf );
        printf( "compare_result =
if (compare_result != 0)
return(compare_result);    /* Exit if NO COMPARE */
    }
    else { /* fread() failed */
        if ( ferror(stream) ) /* possibility 1 */
            printf( "Error reading file %s\n", infiledd[loop_count]);
        else if ( feof(stream)) { /* possibility 2 */
            printf( "EOF found\n" );
            printf( "Number of characters read %d\n", num );
            printf( "Receive compare_buf = %.*s\n", num, compare_buf);
        }
    }
}
else
    printf( "Error opening file %s\n", infiledd[loop_count]);

memset(rec_buf, ' ', sizeof(rec_buf));

printf( "End of loop \n" );
} /* end of loop */

printf("-\n");

/*****
/* Finally, CLOSE severs the connection with IMS and frees the */
/* Storage used by the OTMA API. */
/* This will be done at job-step termination but its untidy. */
*****/

```

```

otma_close(
    & anchor,      /* (in,out) ptr to otma anchor */
    & retrsns     /* (out) rc,reason (1-4) */
);
printf("OTMA_CLOSE done. ret = %.8x rsn = %.8x,%.8x,%.8x,%.8x \n",
    retrsns.ret,
    retrsns.rsn[0],
    retrsns.rsn[1],
    retrsns.rsn[2],
    retrsns.rsn[3]
);

return (compare_result); /* We're done */
} /* end of main */

/*=====*/
/* Subroutine to compare expected results(compare_buf) */
/* with actual results(err_msg) the "|" is used to signify */
/* an ignore compare and "?" is used to mark the end of string. */
/* Note: Compare starts using an index i=1 ie. the 2nd character */
/* because the 1st character was blanked out. ( NL x'15' ) */
/*=====*/

int memc(char *comp_buf, char *rec_buf1)
{
    int j;
    int i;

    j = 0;

    for (i=1;
        ( (j==0) && (comp_buf[i] != '?') );
        i++)
    {
        if( comp_buf[i] != '|' ) /* Ignore compare */
        {
            if( comp_buf[i] != rec_buf1[i]) /* compare ok ? */
            {
                j++; /* No */
                printf( "MISCOMPARE !!! \n" );
                printf( "comp_buf[%d] = %c\n", i, comp_buf[i] );
                printf( "rec_buf1[%d] = %c\n", i, rec_buf1[i] );
            }
            else
                ;
        }
        else
            ; /* Else null */
    }

    return (j);
}

```

**Related reference:**

“OTMA C/I hints and tips” on page 553





---

## Chapter 7. WSDL-to-PL/I segmentation APIs for web service development

The WSDL-to-PL/I segmentation APIs are used and referenced by the PL/I application templates that are generated by IBM Developer for System z® for the WSDL-to-PL/I top-down development scenario.

In general, each API operates on a @dfs\_async\_msg\_header instance to consume IMS messages that have been derived from XML or SOAP, or to produce IMS messages that need to be converted to XML or SOAP.

These APIs are implemented in a new module named DFSPWSIO that is provided with IMS. The DFSPWSIO module must be statically linked to PL/I-XML converters and service provider MPPs generated in the IBM Developer for System z WSDL2PLI scenario for web services to deploy on IMS Enterprise Suite SOAP Gateway. The DFSPWSHK exit routine can be used to inspect, modify, or replace the buffer that contains the current SOAP header, body, or fault data structure before it is sent or received.

**Important:** A minimum of IMS Enterprise Suite V3.1 SOAP Gateway and IBM Developer for System z V9.0.1.1 are required to use this set of APIs.

**Related concepts:**

➔ WSDL-to-PL/I segmentation APIs for adding business logic in generated PL/I templates (Application Programming)

**Related reference:**

➔ WSDL-to-PL/I segmentation APIs exit routine (DFSPWSHK) (Exit Routines)

---

### Include file DFSPWSH

The include file DFSPWSH defines the PL/I structures used with the WSDL-to-PL/I segmentation APIs DFSQGETS, DFSQSETS, DFSXGETS, and DFSXSETS.

The include file DFSPWSH is located on the z/OS server in the data set DFSSSAMP installed by IMS.

The following code shows the first part of DFSPWSH, before the definition of the segmentation APIs:

#### First part of the include file DFSPWSH

```
/*
 * IBM IMS Web service segmentation APIs
 * IMS Connect and IMS MPP
 * DFSPWSH
 *
 * This file must be included by all IMS service provider MPPs
 * developed using the IBM Rational Developer for System z WSDL2PLI
 * support for IMS Enterprise Suite SOAP Gateway.
 *
 * @since 1.0.0.0, 1F64F288-F037-469F-987B-60BF1FBE4B4B
 * @version 2.0.0.0, 2FFA2F75-8D4F-4951-80D5-D2444181745C
 */
```

```

%push;
%noprnt;
#include CEEIBMCT;
#include CEEIBMAW;
%pop;

/*****
 * Required, symmetric asynchronous message header segment for use
 * with DFSPWSIO APIs: DFSQGETS, DFSQSETS, DFSXGETS, DFSXSETS.
 * @version 2.0.0.0, 2FFA2F75-8D4F-4951-80D5-D2444181745C
 *****/
dcl 01 @dfs_async_msg_header_ptr pointer;
dcl 01 @dfs_async_msg_header unaligned
    based(@dfs_async_msg_header_ptr),
    02 ll          fixed bin (15) init(0),
    02 zz          fixed bin (15) init(0),
    02 trancode   char (08) init(''),
    02 header_guid char (36) init
        ('2FFA2F75-8D4F-4951-80D5-D2444181745C'),
    02 service_context,
    03 target_namespace wchar (1024) varying init(''),
    03 service_name     wchar (0512) varying init(''),
    03 port_name        wchar (0512) varying init(''),
    03 operation_name   wchar (0512) varying init(''),
    02 language_binding,
    03 struct_max_segment_size fixed bin(31) init(32767),
    03 soap_header_bit bit (1) aligned init('0'b),
    03 soap_header,
    04 header_struct_name     wchar (100) varying init(''),
    04 header_struct_segment_num fixed bin (31) init(0),
    04 header_struct_segment_cnt fixed bin (31) init(0),
    04 header_struct_size     fixed bin (31) init(0),
    04 header_struct_ptr      pointer,
    03 soap_body_bit bit (1) aligned init('0'b),
    03 soap_body,
    04 body_struct_name       wchar (100) varying init(''),
    04 body_struct_segment_num fixed bin (31) init(0),
    04 body_struct_segment_cnt fixed bin (31) init(0),
    04 body_struct_size       fixed bin (31) init(0),
    04 body_struct_ptr        pointer,
    03 soap_fault_bit bit (1) aligned init('0'b),
    03 soap_fault,
    04 fault_struct_name      wchar (100) varying init(''),
    04 fault_struct_segment_num fixed bin (31) init(0),
    04 fault_struct_segment_cnt fixed bin (31) init(0),
    04 fault_struct_size      fixed bin (31) init(0),
    04 fault_struct_ptr        pointer;

dcl @dfs_async_msg_header_size fixed bin(31)
    value(storage(@dfs_async_msg_header));

/*****
 * IMS I/O Program Communication Block (IOPCB) declarations and
 * constants.
 *****/
dcl 01 @dfs_iopcb_mask_ptr pointer;
dcl 01 @dfs_iopcb_mask unaligned based(@dfs_iopcb_mask_ptr),
    02 iopcb_lterm      char(8),
    02 resv              char(2),
    02 iopcb_status_code char(2),
    02 iopcb_date        decimal fixed(7,0),
    02 iopcb_time        decimal fixed(6,9),
    02 iopcb_msg_seq_number fixed bin(31),
    02 iopcb_mod_name    char(8),
    02 iopcb_user_id     char(8);

/*****

```

```

* @param @dfs_STRUCT_TYPE constants for use with DFSPWSIO APIs:
* DFSQGETS, DFSQSETS.
*****/
dcl @dfs_soap_header_struct fixed bin(31) value(1);
dcl @dfs_soap_body_struct  fixed bin(31) value(2);
dcl @dfs_soap_fault_struct fixed bin(31) value(3);

/*****
* Return code constants for use with DFSPWSIO APIs:
* DFSQGETS, DFSQSETS, DFSXGETS, DFSXSETS.
*****/
dcl @dfs_success           fixed bin(31) value(000);
dcl @dfs_omitted_parameter fixed bin(31) value(100);
dcl @dfs_invalid_pointer   fixed bin(31) value(101);
dcl @dfs_invalid_struct_type fixed bin(31) value(102);
dcl @dfs_struct_not_found  fixed bin(31) value(103);
dcl @dfs_struct_name_mismatch fixed bin(31) value(104);
dcl @dfs_invalid_struct_order fixed bin(31) value(105);
dcl @dfs_invalid_struct_size fixed bin(31) value(106);
dcl @dfs_invalid_struct_name fixed bin(31) value(107);
dcl @dfs_struct_already_set fixed bin(31) value(108);
dcl @dfs_invalid_segment_size fixed bin(31) value(109);

dcl @dfs_icon_buf_exhausted fixed bin(31) value(997);
dcl @dfs_cee_call_failure   fixed bin(31) value(998);
dcl @dfs_dli_call_failure   fixed bin(31) value(999);

/*****
* IMS CEETDLI interface declarations and constants.
*****/
dcl @dfs_dli_get_unique     char (4) value('GU ');
dcl @dfs_dli_get_next      char (4) value('GN ');
dcl @dfs_dli_insert        char (4) value('ISRT');
dcl @dfs_dli_message_exists char (2) value('CF');
dcl @dfs_dli_end_segments  char (2) value('QD');
dcl @dfs_dli_end_messages  char (2) value('QC');
dcl @dfs_dli_status_ok     char (2) value(' ');

dcl @dfs_message_max_data fixed bin(31) value(2147123205);
dcl @dfs_segment_max_data fixed bin(31) value(32763);

/*****
* Language Environment declarations and constants.
*****/
dcl 1 @dfs_cee_feedback feedback;

/*****
* Note: The remainder of this file contains declarations for
* the APIs that enable the XML Converters
* running in IMS Connect and the MPP running in an MPR to
* exchange messages that conform to a protocol that provides
* service invocation context and unique language bindings for
* each part of a SOAP message: header, body, fault.
*****/
dcl @dfs_icon_buf_ptr      pointer init(null());
dcl @dfs_icon_buf_size    fixed bin(31) init(0);
dcl @dfs_icon_buf_used    fixed bin(31) init(0);
dcl @dfs_struct_name      wchar(100) varying init('');
dcl @dfs_struct_ptr       pointer init(null());
dcl @dfs_struct_size      fixed bin(31) init(0);
dcl @dfs_cee_feedback_ptr pointer init(null());
dcl @dfs_commit_structs   bit(1) init('0'b);
dcl @dfs_debug            bit(1) init('0'b);
dcl @return_code          fixed bin(31) init(0);

/*****
* DFSQGETS,

```

```

*      Get a language structure that contains either a SOAP Header,
*      SOAP Body, or SOAP Fault. Language structures are retrieved
*      from the IMS Message Queue using the CEETDLI interface. All
*      language structures must be retrieved from the IMS Message Queue
*      prior to setting language structures using API DFSQSETS.
*
* @param @dfs_async_msg_header_ptr,
*      A pointer-by-value to the instance of @dfs_async_msg_header
*      that was retrieved from the IMS Message Queue by issuing
*      a GU using the CEETDLI interface prior to invoking the API.
*      This same instance must be passed on subsequent calls to
*      DFSQGETS and DFSQSETS.
*
* @param @dfs_iopcb_ptr,
*      A pointer-by-value to the I/O PCB that was passed to the
*      MPP by IMS. The I/O PCB will be used by the API when invoking
*      CEETDLI to interact with the IMS Message Queue. If a return
*      code of 999 is received from the API, inspect the I/O PCB
*      to determine the cause of the error.
*
* @param @dfs_struct_type,
*      An integer-by-value that specifies which language structure
*      to retrieve from the MPP's input message. The following
*      constants defined in include file DFSPWSH may be used:
*      @dfs_soap_header_struct, @dfs_soap_body_struct,
*      @dfs_soap_fault_struct.
*
* @param @dfs_struct_name,
*      A string-by-reference which contains the name of the
*      language structure that the API should retrieve from the
*      IMS Message Queue. This value of this parameter must
*      correspond to the value of parameter @dfs_struct_type.
*
* @param @dfs_struct_ptr,
*      A pointer-by-reference into which the API will write the
*      address of newly-allocated storage into which the requested
*      language structure has been copied from the IMS Message
*      Queue. The storage allocated by the API resides in the
*      same address space as the caller. Therefore, it is highly
*      recommended that the storage be explicitly freed by the
*      caller when no longer needed.
*
* @param @dfs_struct_size,
*      An integer-by-reference into which the API will write the
*      size in bytes of the language structure.
*
* @param @dfs_cee_feedback_ptr,
*      A pointer-by-value to an instance of @dfs_cee_feedback
*      which defines a Language Environment Condition Token.
*      The supplied instance is updated each time the API invokes
*      Language Environment Callable Services. If a return code of
*      998 is received from the API, use the publication Language
*      Environment Run-Time Messages (SA22-7566-10) to inspect
*      the contents of the condition token and determine the
*      cause of the error.
*
* @param @dfs_debug,
*      An optional bit that indicates whether or not
*      trace information should be displayed by the API.
*      Under normal circumstances trace information is written
*      to standard out and therefore can be found in the
*      job log of the Message Processing Region.
*
* @return One of the following codes will be returned by the API,
*      o @dfs_success
*      o @dfs_omitted_parameter
*      o @dfs_invalid_pointer

```

```

*    o @dfs_invalid_dfs_struct_type
*    o @dfs_struct_not_found
*    o @dfs_struct_name_mismatch
*    o @dfs_invalid_struct_order
*    o @dfs_cee_call_failure
*    o @dfs_dli_call_failure
*****/
decl DFSQGETS entry(pointer byvalue, pointer byvalue,
    fixed bin(31) byvalue, wchar(100) varying byaddr,
    pointer byaddr, fixed bin(31) byaddr, pointer byvalue,
    bit(1) optional) returns(fixed bin(31));

/*****
* DFSQSETS,
* Set a language structure that contains either the SOAP
* Header, SOAP Body, or SOAP Fault. This API does not
* insert language structures into the IMS Message Queue
* until instructed to do so via parameter @dfs_commit_structs.
* Therefore it is an error to deallocate or otherwise
* invalidate structure pointers passed to the API via parameter
* @dfs_struct_ptr before instructing the API to commit (insert)
* all structures to the IMS Message Queue.
*
* @param @dfs_async_msg_header_ptr,
* A pointer-by-value to the instance of @dfs_async_msg_header
* that was supplied on a previous call to DFSQGETS or DFSQSETS.
* Subsequent calls to this API must specify the same instance
* of @dfs_async_msg_header as it will be progressively updated.
*
* @param @dfs_iopcb_ptr,
* A pointer-by-value to the IOPCB that was passed to the
* MPP by IMS. The IOPCB will be used by the API when invoking
* CEETDLI to interact with the IMS Message Queue. If a return
* code of 999 is received from the API, inspect the I/O PCB
* to determine the cause of the error.
*
* @param @dfs_struct_type,
* An integer-by-value that specifies which language structure
* to set in the IMS Message Queue. The following
* constants defined in include file DFSPWSH may be used:
* @dfs_soap_header_struct, @dfs_soap_body_struct,
* @dfs_soap_fault_struct.
*
* @param @dfs_struct_name,
* A string-by-reference which contains the name of the
* language structure that corresponds to the supplied value of
* the @dfs_struct_type parameter.
*
* @param @dfs_struct_ptr,
* A pointer-by-value to the language structure that
* corresponds to the values specified for parameters
* @dfs_struct_type and @dfs_struct_name.
*
* @param @dfs_struct_size,
* An integer-by-value that specifies the size in bytes of the
* language structure supplied via parameter @dfs_struct_ptr.
*
* @param @dfs_commit_structs,
* A bit-by-value that indicates whether the API should
* insert the current and all previously supplied language
* structures into the IMS Message Queue.
*
* @param @dfs_cee_feedback_ptr,
* A pointer-by-value to an instance of @dfs_cee_feedback
* which defines a Language Environment Condition Token.
* The supplied instance is updated each time the API invokes
* Language Environment Callable Services. If a return code of

```

```

*      998 is received from the API, use the publication Language
*      Environment Run-Time Messages (SA22-7566-10) to inspect
*      the contents of the condition token and determine the
*      cause of the error.
*
* @param @dfs_debug,
*      An optional bit that indicates whether or not
*      trace information should be displayed by the API.
*      Under normal circumstances trace information is written
*      to standard out and therefore can be found in the
*      job log of the Message Processing Region.
*
* @return One of the following codes will be returned by the API,
*      o @dfs_success
*      o @dfs_omitted_parameter
*      o @dfs_invalid_pointer
*      o @dfs_invalid_dfs_struct_type
*      o @dfs_invalid_struct_order
*      o @dfs_invalid_struct_size
*      o @dfs_invalid_struct_name
*      o @dfs_struct_already_set
*      o @dfs_invalid_segment_size
*      o @dfs_cee_call_failure
*      o @dfs_dli_call_failure
*****/
dcl DFSQSETS entry(pointer byvalue, pointer byvalue,
fixed bin(31) byvalue, wchar(100) varying byaddr,
pointer byvalue, fixed bin(31) byvalue, bit(1) byvalue,
pointer byvalue, bit(1) optional) returns(fixed bin(31));

/*****
* DFSXGETS,
*      Get a language structure that contains either the SOAP
*      Header, SOAP Body, or SOAP Fault. Since the IMS Message
*      Queue is not available to XML Conversion in IMS Connect,
*      language structures are retrieved from the IMS Connect input
*      buffer. The expected format of the IMS Connect input buffer
*      is an [LLZZDATA]+ byte stream. This API is for use by PL/I
*      XML Converters running in IMS Connect. It is not to be used
*      by an MPP.
*
* @param @dfs_async_msg_header_ptr,
*      A pointer-by-value to the instance of @dfs_async_msg_header
*      that was retrieved from the first segment of the IMS Connect
*      input buffer prior to invoking the API.
*
* @param @dfs_icon_buf_ptr,
*      A pointer-by-value to the IMS Connect input message buffer.
*      The expected format of the buffer is an array of LLZZDATA.
*
* @param @dfs_icon_buf_size,
*      An integer-by-value that specifies the length in bytes of
*      the buffer supplied in parameter @dfs_icon_buf_ptr.
*
* @param @dfs_struct_type,
*      An integer-by-value that specifies which language structure
*      to retrieve from the MPP's input message. The following
*      constants defined in include file DFSPWSH may be used:
*      @dfs_soap_header_struct, @dfs_soap_body_struct,
*      @dfs_soap_fault_struct.
*
* @param @dfs_struct_name,
*      A string-by-reference which contains the name of the
*      language structure that the API should retrieve from the
*      IMS Connect input buffer. This value of this parameter
*      must correspond to the value of parameter @dfs_struct_type.
*

```

```

* @param @dfs_struct_ptr,
*   A pointer-by-reference into which the API will write the
*   address of a buffer that contains the bytes of the structure
*   that corresponds to the values specified for parameters
*   @dfs_struct_type and @dfs_struct_name. This buffer must be
*   freed by the XML Converter prior to returning to IMS Connect
*   because the Language Environment enclave in which the XML
*   Converters execute is persistent.
*
* @param @dfs_struct_size,
*   An integer-by-reference into which the API will write the
*   size in bytes of the structure that corresponds to the
*   specified values of parameters @dfs_struct_type
*   and @dfs_struct_name.
*
* @param @dfs_cee_feedback_ptr,
*   A pointer-by-value to an instance of @dfs_cee_feedback
*   which defines a Language Environment Condition Token.
*   The supplied instance is updated each time the API invokes
*   Language Environment Callable Services. If a return code of
*   998 is received from the API, use the publication Language
*   Environment Run-Time Messages (SA22-7566-10) to inspect
*   the contents of the condition token and determine the
*   cause of the error.
*
* @param @dfs_debug,
*   An optional bit that indicates whether or not
*   trace information should be displayed by the API.
*   Under normal circumstances trace information is written
*   to standard out and therefore can be found in the
*   IMS Connect job log.
*
* @return One of the following codes will be returned by the API,
*   o @dfs_success
*   o @dfs_omitted_parameter
*   o @dfs_invalid_pointer
*   o @dfs_invalid_dfs_struct_type
*   o @dfs_struct_not_found
*   o @dfs_struct_name_mismatch
*   o @dfs_invalid_struct_order
*   o @dfs_icon_buf_exhausted
*   o @dfs_cee_call_failure
*****/
dcl DFSXGETS entry(pointer byvalue, pointer byvalue,
  fixed bin(31) byvalue, fixed bin(31) byvalue,
  wchar(100) varying byaddr, pointer byaddr, fixed bin(31) byaddr,
  pointer byvalue, bit(1) optional) returns(fixed bin(31));

/*****
* DFSXSETS,
*   Set a language structure that contains either the SOAP
*   Header, SOAP Body, or SOAP Fault. This API does not
*   copy language structures into the IMS Connect output buffer
*   until instructed to do so via parameter @dfs_commit_structs.
*   Therefore it is an error to deallocate or otherwise
*   invalidate structure pointers passed to the API via parameter
*   @dfs_struct_ptr before instructing the API to commit (copy)
*   all structures to the IMS Connect output buffer.
*   This API is for use by PL/I XML Converters running in IMS
*   Connect. It is not to be used by an MPP.
*
* @param @dfs_async_msg_header_ptr,
*   A pointer-by-value to the instance of @dfs_async_msg_header
*   that will be sent as the first segment of the IMS message.
*
* @param @dfs_icon_buf_ptr,
*   A pointer-by-value to the IMS Connect output message buffer.

```

```

*     The expected format of the buffer is an array of LLZZDATA.
*
* @param @dfs_icon_buf_size,
*     An integer-by-value that specifies the length in bytes of
*     the buffer supplied in parameter @dfs_icon_buf_ptr.
*
* @param @dfs_icon_buf_used,
*     An integer-by-reference into which the API will write
*     the number of bytes that were required to format the
*     language structure as a multi-segment IMS message
*     in the IMS Connect output buffer. The value of this
*     parameter will always be greater than the actual size
*     of the language structure by at least 4 bytes.
*
* @param @dfs_struct_type,
*     An integer-by-value that specifies which language structure
*     to set in the IMS Connect output buffer. The following
*     constants defined in include file DFSPWSH may be used:
*     @dfs_soap_header_struct, @dfs_soap_body_struct,
*     @dfs_soap_fault_struct.
*
* @param @dfs_struct_name,
*     A string-by-reference which contains the name of the
*     language structure that corresponds to the supplied value of
*     the @dfs_struct_type parameter.
*
* @param @dfs_struct_ptr,
*     A pointer-by-value to the language structure that
*     corresponds to the values specified for parameters
*     @dfs_struct_type and @dfs_struct_name.
*
* @param @dfs_struct_size,
*     An integer-by-value that specifies the size in bytes of the
*     language structure.
*
* @param @dfs_commit_structs,
*     A bit-by-value that indicates whether the API should
*     copy the current and all previously supplied language
*     structures into the IMS Connect output buffer.
*
* @param @dfs_cee_feedback_ptr,
*     A pointer-by-value to a Language Environment Condition Token
*     (@dfs_cee_feedback) that is updated by the API after each
*     invocation of a Language Environment Callable Service. When
*     a RETURN_CODE of 998 is received from the API, use the
*     publication Language Environment Run-Time Messages
*     (SA22-7566-10) to inspect the contents of the condition
*     token and determine the cause of the error.
*
* @param @dfs_debug,
*     An optional bit that indicates whether or not
*     trace information should be displayed by the API.
*     Under normal circumstances trace information is written
*     to standard out and therefore can be found in the
*     IMS Connect job log.
*
* @return One of the following codes will be returned by the API,
*     o @dfs_success
*     o @dfs_omitted_parameter
*     o @dfs_invalid_pointer
*     o @dfs_invalid_dfs_struct_type
*     o @dfs_invalid_struct_order
*     o @dfs_invalid_struct_size
*     o @dfs_invalid_struct_name
*     o @dfs_struct_already_set
*     o @dfs_invalid_segment_size
*     o @dfs_icon_buf_exhausted

```



```

*      o @dfs_cee_call_failure
*****/
dcl DFSXSETS entry(pointer byvalue, pointer byvalue,
    fixed bin(31) byvalue, fixed bin(31) byaddr,
    fixed bin(31) byvalue, wchar(100) varying byaddr,
    pointer byvalue, fixed bin(31) byvalue, bit(1) byvalue,
    pointer byvalue, bit(1) optional) returns(fixed bin(31));

/*****
* DFSB64E,
*   This API encodes an input buffer using the base64 encoding
*   scheme specified by RFC 3548 available at
*   http://tools.ietf.org/html/rfc3548.
*
* @param @bin_input_buf_ptr (input),
*   A pointer-by-value to the binary buffer to encode in base64.
*   The base64 sequence will be encoded in UTF-16.
*
* @param @bin_input_buf_len (input),
*   An integer-by-value that specifies the length in bytes of
*   the binary buffer supplied in parameter @bin_input_buf_ptr.
*
* @param @b64_output_buf_ptr (input),
*   A pointer-by-value to a buffer in which to write the base64
*   representation of the supplied binary buffer
*   @bin_input_buf_ptr(1:@bin_input_buf_len). The buffer pointed
*   to by this parameter must have a minimum length in bytes of
*   [4 * floor( ( @bin_input_buf_len + 2 ) /3 )]. If this
*   parameter is set to null, the API will write the length
*   in bytes of the base64 result to parameter
*   @b64_output_buf_len but will not actually perform encoding.
*
* @param @b64_output_buf_len (input),
*   An integer-by-reference into which the API will write the
*   length in bytes of the base64 sequence that was written to
*   the buffer pointed to by parameter @b64_output_buf_ptr.
*   Recall that result will be encoded in UTF-16.
*
* @return This API does not return any codes.
*****/
dcl DFSB64E entry(pointer byvalue, fixed bin(31) byvalue,
    pointer byvalue, fixed bin(31) byaddr);

/*****
* DFSB64D,
*   This API decodes a base64 input buffer by reversing the
*   encoding scheme specified by RFC 3548
*   available at http://tools.ietf.org/html/rfc3548.
*
* @param @b64_input_buf_ptr (input),
*   A pointer-by-value to the base64 buffer to decode.
*   The base64 sequence must be encoded in UTF-16.
*
* @param @b64_input_buf_len (input),
*   An integer-by-value that specifies the length in bytes of
*   the base64 buffer supplied in parameter @b64_input_buf_ptr.
*
* @param @bin_output_buf_ptr (input),
*   A pointer-by-value to a buffer in which to write the decoded
*   representation of the supplied base64 buffer
*   @b64_input_buf_ptr(1:@b64_input_buf_len). If this
*   parameter is set to null, the API will write the length
*   in bytes of the decoded result to parameter
*   @bin_output_buf_len but will not actually perform decoding.
*
* @param @bin_output_buf_len (input),
*   An integer-by-reference into which the API will write the


```

```

*     length in bytes of the decoded result that was written
*     to the buffer pointed to by parameter @bin_output_buf_ptr).
*
* @return This API does not return any codes.
*****/
dcl DFSB64D entry(pointer byvalue, fixed bin(31) byvalue,
                 pointer byvalue, fixed bin(31) byaddr);

```

**Related concepts:**

 WSDL-to-PL/I segmentation APIs for adding business logic in generated PL/I templates (Application Programming)

## DFSQGETS

The DFSQGETS API retrieves a SOAP structure from the IMS Message Queue and returns the information to the caller in a high-level language structure.

The structures and variables referred to in this topic are defined in the include file DFSPWSH (see “Include file DFSPWSH” on page 595).

**Usage:**

- You must retrieve all structures in the IMS Message Queue using DFSQGETS before invoking DFSQSETS to put structures into it.

**Limitations:**

- DFSQGETS supports the retrieval of SOAP Body and SOAP Fault structures only. The SOAP Header structure is not supported.

**Parameters:**

*Table 149. Parameters for DFSQGETS*

Parameter	Type	Usage	Description
@dfs_async_msg_header_ptr	POINTER BYVALUE	Input	A pointer-by-value to the instance of @dfs_async_msg_header that the Message Processing Program (MPP) retrieves from the IMS Message Queue prior by issuing a Get Unique (GU) call using the CEETDLI interface. <b>Important:</b> This instance must be passed on all calls to DFSQGETS and DFSQSETS.
@dfs_iopcb_ptr	POINTER BYVALUE	Input	A pointer-by-value to the I/O PCB that was passed to the Message Processing Program (MPP) on entry by IMS. DFSQGETS uses this I/O PCB when invoking CEETDLI to interact with the IMS Message Queue. <b>Note:</b> If the return code from DFSQGETS is 999 then inspect the I/O PCB to determine the cause of the error.

Table 149. Parameters for DFSQGETS (continued)

Parameter	Type	Usage	Description
@dfs_struct_type	SIGNED FIXED BIN(31) BYVALUE	Input	An integer-by-value specifying the type of the language structure to retrieve from the IMS Message Queue. The following constants defined in the include file DFSPWSH can be used: @dfs_soap_body_struct.
@dfs_struct_name	WCHAR(100) VARYING BYADDR	Input	A string-by-reference containing the name of the language structure to retrieve from the IMS Message Queue. The value of this parameter must correspond to the value of the parameter @dfs_struct_type.
@dfs_struct_ptr	POINTER BYADDR	Output	A pointer-by-reference to a newly allocated storage block containing the returned language structure. <b>Important:</b> This storage block resides in the same address space as the caller. Therefore it is highly recommended that the caller free this storage block when it is no longer needed.
@dfs_struct_size	SIGNED FIXED BIN(31) BYADDR	Output	An integer-by-reference containing the size in bytes of the returned language structure.
@dfs_cee_feedback_ptr	POINTER BYVALUE	Input	A pointer-by-value to an instance of @dfs_cee_feedback defining a Language Environment® Condition Token. This instance is updated each time DFSQGETS invokes Language Environment Callable Services. <b>Note:</b> If the return code from DFSQGETS is 998 then use the publication <i>Language Environment Run-Time Messages (SA22-7566-10)</i> to inspect the contents of the condition token and determine the cause of the error.
@dfs_debug	BIT(1) OPTIONAL	Input	An optional bit indicating whether DFSQGETS should display trace information (see Trace output for WSDL-to-PL/I segmentation APIs (Application Programming)).

#### Return codes:

The return codes for DFSQGETS are constants defined in the DFSPWSH include file:

Table 150. Return codes for DFSQGETS

Type:	Name:	Value:
SIGNED FIXED BIN(31)	@dfs_success	000
	@dfs_omitted_parameter	100
	@dfs_invalid_pointer	101
	@dfs_invalid_struct_type	102
	@dfs_struct_not_found	103
	@dfs_struct_name_mismatch	104
	@dfs_invalid_struct_order	105
	@dfs_cee_call_failure	998
	@dfs_dli_call_failure	999

### Example invocation of DFSQGETS

```

01: /* Invoke API DFSQGETS to retrieve the SOAP body
02: * language structure from the IMS Message Queue.
03: */
04: @dfs_struct_name      = 'RequestBodyStruct';
05: @dfs_cee_feedback_ptr = addr(@dfs_cee_feedback);
06: @dfs_debug = '0'b;
07:
08: @return_code =
09:   DFSQGETS(@dfs_async_msg_header_ptr,
10:   @dfs_iopcb_mask_ptr, @dfs_soap_body_struct,
11:   @dfs_struct_name, @dfs_struct_ptr,
12:   @dfs_struct_size, @dfs_cee_feedback_ptr,
13:   @dfs_debug);
14:
15: if (@return_code != @dfs_success) then do;
16:   display('MYMPP#handle_myOperation(): '
17:   || 'ERROR, DFSQGETS @dfs_soap_body_struct, '
18:   || '@return_code: ' || trim(@return_code) || '.');
19:   return;
20: end; else do;
21:   RequestBodyStruct_ptr = @dfs_struct_ptr;
22: end;

```

## DFSQSETS

The DFSQSETS API creates a SOAP structure from the information in a language structure passed as input. Also, when specified, DFSQSETS copies the current SOAP structure and all previously supplied SOAP structures into the IMS Message Queue.

The structures and variables referred to in this topic are defined in DFSPWSH (see “Include file DFSPWSH” on page 595).

### Usage:

- Do not deallocate or otherwise invalidate structure pointers passed to DFSQSETS via the parameter @dfs\_struct\_ptr until you have committed the IMS multi-segment message to the IMS Message Queue by calling DFSQSETS with the @dfs\_commit\_structs bit set.
- You must retrieve all structures in the IMS Message Queue using DFSQGETS before invoking DFSQSETS to put structures into it.

**Limitations:**

- DFSQSETS does not support the storing of a SOAP Header structure.

**Parameters:***Table 151. Parameters for DFSQSETS*

Parameter	Type	Usage	Description
@dfs_async_msg_header_ptr	POINTER BYVALUE	Output	A pointer-by-value to the instance of @dfs_async_msg_header that the Message Processing Program (MPP) retrieves from the IMS Message Queue prior by issuing a Get Unique (GU) call using the CEETDLI interface. <b>Important:</b> This instance must be passed on all calls to DFSQSETS and DFSQSETS.
@dfs_iopcb_ptr	POINTER BYVALUE	Input	A pointer-by-value to the I/O PCB that was passed to the Message Processing Program (MPP) on entry by IMS. DFSQSETS uses this I/O PCB when invoking CEETDLI to interact with the IMS Message Queue. <b>Note:</b> If the return code from DFSQSETS is 999 then inspect the I/O PCB to determine the cause of the error.
@dfs_struct_type	SIGNED FIXED BIN(31) BYVALUE	Input	An integer-by-value specifying the type of the language structure to set in the IMS message. The following constants defined in include file DFSPWSH can be used: @dfs_soap_body_struct.
@dfs_struct_name	WCHAR(100) VARYING BYADDR	Input	A string-by-reference containing the name of the language structure to set in the IMS message. The value of this parameter must correspond to the value of the parameter @dfs_struct_type.
@dfs_struct_ptr	POINTER BYVALUE	Input	A pointer-by-reference to the language structure to set in the IMS message. <b>Note:</b> This value must correspond to the values specified for the parameters @dfs_struct_type and @dfs_struct_name.
@dfs_struct_size	SIGNED FIXED BIN(31) BYVALUE	Input	An integer-by-value specifying the size in bytes of the language structure pointed to by @dfs_struct_ptr.
@dfs_commit_structs	BIT(1) BYVALUE	Input	A bit-by-value that indicating whether DFSQSETS should insert the current language structure and all previously supplied language structures into the IMS Message Queue.

Table 151. Parameters for DFSQSETS (continued)

Parameter	Type	Usage	Description
@dfs_cee_feedback_ptr	POINTER BYVALUE	Input	A pointer-by-value to an instance of @dfs_cee_feedback defining a Language Environment Condition Token. This instance is updated each time DFSQSETS invokes Language Environment Callable Services. <b>Note:</b> If the return code from DFSQSETS is 998 then use the publication <i>Language Environment Run-Time Messages (SA22-7566-10)</i> to inspect the contents of the condition token and determine the cause of the error.
@dfs_debug	BIT(1) OPTIONAL	Input	An optional bit indicating whether DFSQSETS should display trace information (see Trace output for WSDL-to-PL/I segmentation APIs (Application Programming)).

#### Return codes:

The return codes for DFSQSETS are constants defined in the DFSPWSH include file:

Table 152. Return codes for DFSQSETS

Type:	Name:	Value:
SIGNED FIXED BIN(31)	@dfs_success	000
	@dfs_omitted_parameter	100
	@dfs_invalid_pointer	101
	@dfs_invalid_struct_type	102
	@dfs_struct_not_found	103
	@dfs_struct_name_mismatch	104
	@dfs_invalid_struct_order	105
	@dfs_invalid_segment_size	109
	@dfs_cee_call_failure	998
	@irz_dli_call_failure	999

#### Example invocation of DFSQSETS

```

01: /* Invoke API DFSQSETS to set the SOAP body language
02: * structure and commit it to the IMS Message Queue.
03: */
04: @irz_struct_name      = 'ResponseBodyStruct';
05: @irz_struct_ptr       = ResponseBodyStruct;
06: @dfs_struct_size      = storage(ResponseBodyStruct);
07: @dfs_commit_structs   = '1'b;
08: @dfs_cee_feedback_ptr = addr(@dfs_cee_feedback);
09: @dfs_debug            = '0'b;
10:
11: @return_code =
12:   DFSQSETS(@dfs_async_msg_header_ptr,

```

```

13:   @dfs_iopcb_mask_ptr, @dfs_soap_body_struct,
14:   @dfs_struct_name, @dfs_struct_ptr,
15:   @dfs_struct_size, @dfs_commit_structs,
16:   @dfs_cee_feedback_ptr, @dfs_debug);
17:
18: if (@return_code != @dfs_success) then do;
19:   display('MYMPP#handle_myOperation(): '
20:         || 'ERROR, DFSQSETS @dfs_soap_body_struct, '
21:         || '@return_code: ' || trim(@return_code) || '.');
22:   return;
23: end;

```

---

## DFSXGETS

The DFSXGETS API retrieves a SOAP structure from the IMS Connect input buffer and returns the information to the caller in a high-level language structure.

Because the IMS Message Queue is not available to XML Conversion in IMS Connect, DFSXGETS retrieves language structures from the IMS Connect input buffer. The expected format of the IMS Connect input buffer is an array of IMS message segments (LLZZDATA).

The structures and variables referred to in this topic are defined in the include file DFSPWSH (see “Include file DFSPWSH” on page 595).

**Note:** This API is for use by PL/I XML converters running in IMS Connect. It is not to be used by a message processing program (MPP).

### Restrictions:

- DFSXGETS supports the retrieval of SOAP Body and SOAP Fault structures only. The SOAP Header structure is not supported.

### Parameters:

Table 153. Parameters for DFSXGETS

Parameter	Type	Usage	Description
@dfs_async_msg_header_ptr	POINTER BYADDR	Output	A pointer-by-reference to the instance of @dfs_async_msg_header that was received in the IMS Connect input buffer.
@dfs_icon_buf_ptr	POINTER BYVALUE	Input	A pointer-by-value to the IMS Connect input message buffer. The expected format of the buffer is an array of LLZZDATA.
@dfs_icon_buf_len	SIGNED FIXED BIN(31) BYVALUE	Input	An integer-by-value specifying the length in bytes of the buffer pointed to by @dfs_icon_buf_ptr.
@dfs_struct_type	SIGNED FIXED BIN(31) BYVALUE	Input	An integer-by-value specifying the type of structure to retrieve from the IMS Connect input buffer. The following constants defined in the include file DFSPWSH can be used: @dfs_soap_body_struct.

Table 153. Parameters for DFSXGETS (continued)

Parameter	Type	Usage	Description
@dfs_struct_name	WCHAR(100) VARYING BYADDR	Input	A string-by-reference containing the name of the language structure to retrieve from the IMS Connect input buffer. The value of this parameter must correspond to the value of the parameter @dfs_struct_type.
@dfs_struct_ptr	POINTER BYADDR	Output	A pointer-by-reference into which DFSXGETS writes the address of a buffer containing the bytes of the structure requested in parameters @dfs_struct_type and @dfs_struct_name. <b>Important:</b> This buffer must be freed by the XML Converter prior to returning to IMS Connect because the Language Environment enclave in which the XML Converters execute is persistent.
@dfs_struct_size	SIGNED FIXED BIN(31) BYADDR	Output	An integer-by-reference into which DFSXGETS writes the size in bytes of the structure returned in the parameter @dfs_struct_ptr.
@dfs_cee_feedback_ptr	POINTER BYVALUE	Input	A pointer-by-value to an instance of @dfs_cee_feedback defining a Language Environment Condition Token. This instance is updated each time DFSXGETS invokes Language Environment Callable Services. <b>Note:</b> If the return code from DFSXGETS is 998 then use the publication <i>Language Environment Run-Time Messages (SA22-7566-10)</i> to inspect the contents of the condition token and determine the cause of the error.
@dfs_debug	BIT(1) OPTIONAL	Input	An optional bit indicating whether DFSXGETS should display trace information (see Trace output for WSDL-to-PL/I segmentation APIs (Application Programming)).

#### Return codes:

The return codes for DFSXGETS are constants defined in the DFSPWSH include file:



Table 154. Return codes for DFSXGETS

Type:	Name:	Value:
SIGNED FIXED BIN(31)	@dfs_success	000
	@dfs_omitted_parameter	100
	@dfs_invalid_pointer	101
	@dfs_invalid_struct_type	102
	@dfs_struct_not_found	103
	@dfs_struct_name_mismatch	104
	@dfs_invalid_struct_order	105
	@dfs_icon_buf_exhausted	997
	@dfs_cee_call_failure	998

## DFSXSETS

The DFSXSETS API creates a SOAP structure from the information in a language structure passed as input. Also, when specified, DFSXSETS copies the current SOAP structure and all previously supplied SOAP structures into the IMS Connect output buffer.

Because the IMS Message Queue is not available to XML Conversion in IMS Connect, DFSXSETS inserts language structures into the IMS Connect output buffer. The format of the IMS Connect output buffer is an array of IMS message segments (LLZZDATA).

The structures and variables referred to in this topic are defined in DFSPWSH (see “Include file DFSPWSH” on page 595).

**Note:** This API is for use by PL/I XML converters running in IMS Connect. It is not to be used by a message processing program (MPP).

### Limitations:

- DFSXSETS does not support the storing of a SOAP Header structure.

### Parameters:

Table 155. Parameters for DFSXSETS

Parameter	Type	Usage	Description
@dfs_async_msg_header_ptr	POINTER BYVALUE	Input	A pointer-by-value to the instance of @dfs_async_msg_header that is to be sent as the first segment of the IMS message.
@dfs_icon_buf_ptr	POINTER BYVALUE	Input	A pointer-by-value to the IMS Connect output message buffer. The expected format of the buffer is an array of IMS message segments (LLZZDATA).

Table 155. Parameters for DFSXSETS (continued)

Parameter	Type	Usage	Description
@dfs_icon_buf_len	SIGNED FIXED BIN(31) BYVALUE	Input	An integer-by-value specifying the length in bytes of the buffer pointed to by @dfs_icon_buf_ptr.
@dfs_icon_buf_used	SIGNED FIXED BIN(31) BYADDR	Output	An integer-by-reference into which DFSXSETS writes the number of bytes that are required to format the language structure as a multisegment IMS message in the IMS Connect output buffer. The value of this parameter is always greater than the actual size of the language structure by at least 4 bytes.
@dfs_struct_type	SIGNED FIXED BIN(31) BYVALUE	Input	An integer-by-value specifying the type of language structure to set in the IMS Connect output buffer. The following constants defined in include file DFSPWSH can be used: @dfs_soap_body_struct.
@dfs_struct_name	WCHAR(128) VARYING BYADDR	Input	A string-by-reference containing the name of the language structure that corresponds to the value of the parameter @dfs_struct_type.
@dfs_struct_ptr	POINTER BYVALUE	Input	A pointer-by-value to a structure corresponding to the structure specified in parameters @dfs_struct_type and @dfs_struct_name.
@dfs_struct_size	SIGNED FIXED BIN(31) BYVALUE	Input	An integer-by-value specifying the size in bytes of the structure pointed to by parameter @dfs_struct_ptr.
@dfs_commit_structs	BIT(1) BYVALUE	Input	A bit-by-value indicating whether DFSXSETS should copy the current language structure and all previously supplied language structures into the IMS Connect output buffer.

Table 155. Parameters for DFSXSETS (continued)

Parameter	Type	Usage	Description
@dfs_cee_feedback_ptr	POINTER BYVALUE	Input	A pointer-by-value to an instance of @dfs_cee_feedback defining a Language Environment Condition Token. This instance is updated each time DFSXSETS invokes Language Environment Callable Services. <b>Note:</b> If the return code from DFSXSETS is 998 then use the publication <i>Language Environment Run-Time Messages (SA22-7566-10)</i> to inspect the contents of the condition token and determine the cause of the error.
@dfs_debug	BIT(1) OPTIONAL	Input	An optional bit indicating whether DFSXSETS should display trace information (see Trace output for WSDL-to-PL/I segmentation APIs (Application Programming)).

**Return codes:**

The return codes for DFSXSETS are constants defined in the DFSPWSH include file:

Table 156. Return codes for DFSXSETS

Type:	Name:	Value:
SIGNED FIXED BIN(31)	@dfs_success	000
	@dfs_omitted_parameter	100
	@dfs_invalid_pointer	101
	@dfs_invalid_struct_type	102
	@dfs_invalid_struct_order	105
	@dfs_invalid_struct_size	106
	@dfs_invalid_struct_name	107
	@dfs_struct_already_set	108
	@dfs_invalid_segment_size	109
	@dfs_icon_buf_exhausted	997
	@dfs_cee_call_failure	998

## Return codes from the DFSPWSIO APIs

This topic describes the return codes from the DFSPWSIO APIs.

The following table describes the return codes:

*Table 157. Return codes from the DFSPWSIO APIs*

Value	DFSPWSH constant	Description
000	@dfs_success	The API completed successfully.
100	@dfs_omitted_parameter	A required parameter was not specified to the API.
101	@dfs_invalid_pointer	The value of a pointer supplied to the API specified an invalid memory address.
102	@dfs_invalid_struct_type	The language structure type specified to the API was not one of DFSPWSH.@dfs_soap_header_struct, DFSPWSH.@dfs_soap_body_struct, or DFSPWSH.@dfs_soap_fault_struct.
103	@dfs_struct_not_found	A language structure with the specified type was not found in the IMS message.
104	@dfs_struct_name_mismatch	A language structure of the specified type was found in the IMS message, but the specified name does not match.
105	@dfs_invalid_struct_order	An attempt to get or set a language structure out of order was detected. For example, it is an error to attempt to get the SOAP body structure before getting the SOAP header structure, if a SOAP header is present in the IMS message.
106	@dfs_invalid_struct_size	The size of the language structure specified to the API was invalid ( $\leq 0$ ) or exceeded the maximum (see DFSPWSH.@dfs_message_max_data).
107	@dfs_invalid_struct_name	The specified language structure name was not a valid PL/I identifier.
108	@dfs_struct_already_set	The specified language structure type already exists in the IMS message.
109	@dfs_invalid_segment_size	The segment size specified in IMS Connect parameter XMPAMAXS is invalid ( $\leq 5$ or $\geq 32767$ ).
995	@dfs_fetch_failure	The API was unable to fetch a required load module from the available libraries. An instance of Enterprise PL/I runtime message IBM0590S was generated prior to receiving this return code.
997	@dfs_icon_buf_exhausted	The API was unable to get or set a language structure because it encountered the end of the IMS Connect input or output buffer. This error may only be raised when Compiled XML Conversion invokes the API.

Table 157. Return codes from the DFSPWSIO APIs (continued)

Value	DFSPWSH constant	Description
998	@dfs_cee_call_failure	An error was encountered by the API when it invoked a Language Environment callable service. Inspect the Language Environment condition token supplied in parameter @dfs_cee_feedback_ptr for more information.
999	@dfs_dli_call_failure	An error was encountered by the API when it invoked the CEETDLI interface. Inspect the IOPCB supplied in parameter @dfs_iopcb_ptr for more information.

The following table shows the return codes used by each API:

Table 158. Return codes used by each API

Value	DFSPWSH constant	DFSQGETS	DFSQSETS	DFSXGETS	DFSXSETS
000	@dfs_success	X	X	X	X
100	@dfs_omitted_parameter	X	X	X	X
101	@dfs_invalid_pointer	X	X	X	X
102	@dfs_invalid_struct_type	X	X	X	X
103	@dfs_struct_not_found	X			
104	@dfs_struct_name_mismatch	X		X	
105	@dfs_invalid_struct_order	X	X	X	X
106	@dfs_invalid_struct_size		X		X
107	@dfs_invalid_struct_name		X		X
108	@dfs_struct_already_set		X		X
109	@dfs_invalid_segment_size		X		X
995	@dfs_fetch_failure	X	X	X	X
997	@dfs_icon_buf_exhausted			X	X
998	@dfs_cee_call_failure	X	X	X	
999	@dfs_dli_call_failure	X	X		X

<sup>1</sup>At runtime these error codes are cited by messages IRZ0500S and IRZ0501S, even though the APIs are internal to compiled XML conversion.

**Related information:**

 IBM0590S and PL/I run-time messages



---

## Chapter 8. SQL programming reference

These topics provide the reference information for Structured Query Language (SQL) for IMS.

---

### SQL concepts for IMS

Certain IMS concepts are important to understand when using Structured Query Language (SQL).

#### Structured query language

One language that you use to access the data in IMS is SQL. SQL is a standardized language for defining and manipulating data in a relational database.

The language consists of SQL statements. SQL statements let you retrieve, insert, update, or delete data in IMS databases.

When you write an SQL statement, you specify what you want done, not how to do it. To access data, for example, you need only to name the segment and fields that contain the data. You do not need to describe how to get to the data.

In accordance with the relational model of data:

- The database is perceived as a set of tables.
- Relationships are represented by values in tables.
- Data is retrieved by using SQL to specify a *result table* that can be derived from one or more tables.

IMS transforms each SQL statement, that is, the specification of a result table, into a sequence of operations for data retrieval or modifications.

All executable SQL statements must be prepared before they can run.

#### Static SQL

The source form of a *static* SQL statement is embedded within an application program that is written in a host language such as COBOL. The statement is prepared before the program is executed and the operational form of the statement persists beyond the execution of the program.

In IMS Version 13 and later, static SQL is not supported for COBOL.

#### Dynamic SQL

Programs that contain embedded *dynamic* SQL statements must be precompiled like those that contain static SQL, but unlike static SQL, the dynamic statements are constructed and prepared at run time.

In IMS Version 13, only dynamic SQL is supported for COBOL.

The source form of a dynamic statement is a character string that is passed to IMS by the program using the SQL PREPARE statement. A statement that is prepared using the PREPARE statement can be referenced in a DECLARE CURSOR, DESCRIBE, or EXECUTE statement.

## Interactive SQL

*Interactive SQL* refers to SQL statements submitted using IMS Enterprise Suite Explorer for Development.

## IMS data structures for SQL

SQL support allows you to issue standard SQL query to access IMS data instead of using DL/I calls. To use SQL calls in IMS application, you need to understand the difference between the hierarchical model for IMS databases and the standard relational database model since SQL calls are commonly used for relational databases. You also need to understand how IMS database elements are being mapped to relational database elements.

A database segment definition defines the fields for a set of segment instances similar to the way that a relational table defines columns for a set of rows in a table. In this regard, segments relate to tables, and fields in a segment relate to columns in a table. An occurrence of a segment in a database corresponds to a row in a table.

The following table summarizes the mapping between IMS database elements and relational database elements.

*Table 159. Mapping between IMS database elements and relational database elements.*

Hierarchical database elements in IMS	Equivalent relational database elements
Segment name	Table name
Segment instance	Table row
Segment field name	Column name
Segment unique key	Table primary key
Foreign key field	Table foreign key
PCB	Schema
Segment	Table
Field	Column
Record	Row
Data set group or Area	Tablespace

### Related tasks:

 Database design and implementation (Database Administration)

## Hierarchical and relational databases

There are differences between the hierarchical model for IMS databases and the standard relational database model.

IMS presents a relational model of a hierarchical database. In addition to the one-to-one mappings of terms, IMS can also show a hierarchical parentage through the primary or foreign key constraints.

For a comparison of the hierarchical and relational database models, see Comparison of hierarchical and relational databases (Application Programming).



---

## Language elements

An understanding of the basic syntax of SQL and language elements that are common to many SQL statements can be helpful in using SQL with IMS.

The following topics provide information about these language elements:

### Characters

The basic symbols of keywords and operators in the SQL language are *characters* that are classified as letters, digits, or special characters.

- A *letter* is any of the 26 uppercase (A through Z) and 26 lowercase (a through z) letters of the English alphabet.
- A *digit* is any one of the characters 0 through 9.
- A *special character* is any character other than a letter or a digit.

### Tokens

The basic syntactical units of the SQL language are called *tokens*. A token consists of one or more characters of which none are blanks, control characters, or characters within a string constant or delimited identifier.

Tokens are classified as *ordinary* or *delimiter* tokens:

- An *ordinary token* is a numeric constant, an ordinary identifier, a host identifier, or a keyword.

*Examples:*

```
1      .1      +2      SELECT      E      3
```

- A *delimiter token* is a string constant, a delimited identifier, an operator symbol, or any of the special characters shown in the syntax diagrams. A question mark (?) is also a delimiter token when it serves as a parameter marker, as explained in "PREPARE" on page 653.

*Examples:*

```
,      'string'      "fld1"      =      .
```

### Spaces

A *space* is a sequence of one or more blank characters.

### Uppercase and lowercase

A token in an SQL statement can include lowercase letters, but lowercase letters in an ordinary token are folded to uppercase. Delimiter tokens are never folded to uppercase.

*Example:* The following two statements, after folding, are equivalent:

```
select * from PCB01.HOSPITAL where hospname = 'Alexandria';  
SELECT * FROM PCB01.HOSPITAL WHERE HOSPNAME = 'Alexandria';
```

### Identifiers

An *identifier* is a token used to form a name. An identifier in an SQL statement is an SQL identifier or a host identifier.

#### SQL identifiers

SQL identifiers can be *ordinary identifiers* or *delimited identifiers*.

### Ordinary identifiers:

An *ordinary identifier* is an uppercase letter followed by zero or more characters, each of which is an uppercase letter, a digit, or the underscore character.

An ordinary identifier should not be a reserved word. If a reserved word is used as an identifier in SQL, it must be specified in uppercase and must be a delimited identifier or specified in a host variable.

*Example:* The following example is an ordinary identifier:

```
HOSPITAL
```

### Host identifiers

A *host identifier* is a name declared in the host program.

## Naming conventions

The rules for forming a name depend on the type of the object designated by the name.

The syntax diagrams use different terms for different types of names. The following list defines these terms.

#### column-name

A qualified or unqualified name that designates a column of a table.

A qualified column name is a qualifier followed by a period and an SQL identifier. The qualifier is a table name, a view name, a synonym, an alias, or a correlation name. The unqualified column name is an SQL identifier.

#### cursor-name

An SQL identifier that designates an SQL cursor.

#### descriptor-name

A host identifier that designates an SQL descriptor area (SQLIMSDA). See "References to host variables" on page 629 for a description of a host identifier. A descriptor name never includes an indicator variable.

#### host-variable

A sequence of tokens that designates a host variable. A host variable includes at least one host identifier, as explained in "References to host variables" on page 629.

#### statement-name

An SQL identifier that designates a prepared SQL statement.

#### table-name

A qualified or unqualified name that designates a table (IMS segment).

A one-part or unqualified table name is an SQL identifier with two implicit qualifiers.

## Data types

IMS supports data types from SQL.

The smallest unit of data that can be manipulated in SQL is called a *value*. How values are interpreted depends on the data type of their source. The sources of values are:

- Columns

- Constants
- Expressions
- Variables (such as host variables and parameter markers)

The following image shows the built-in data types that IMS supports.

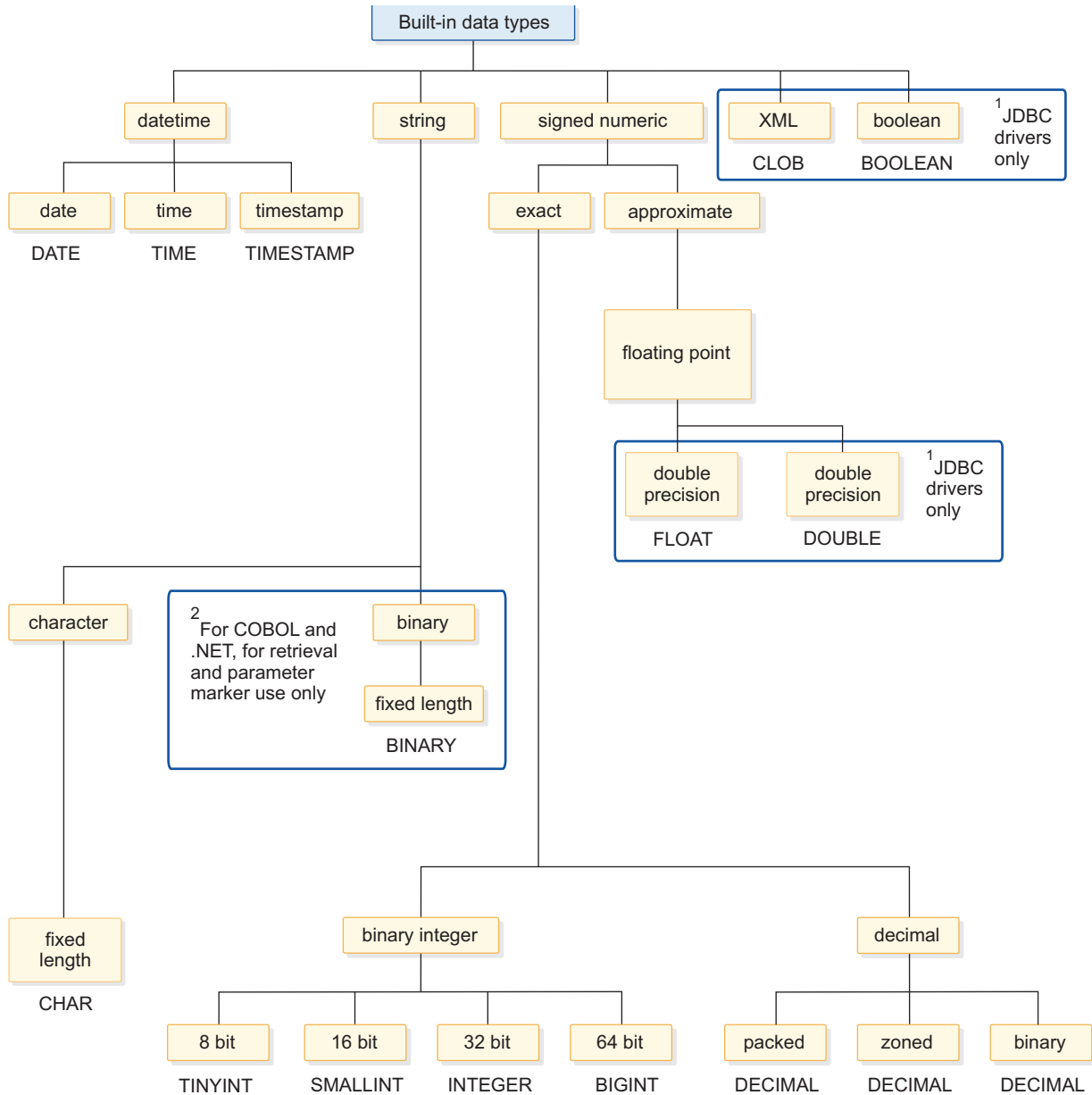


Figure 36. Built-in data types supported by IMS

**Important:**

1. XML, CLOB, BOOLEAN, FLOAT, DOUBLE, and fixed-length binary data types are supported only for Java application programs.
2. Fixed-length binary data type is supported for COBOL and .NET applications for retrieval and parameter marker use only.

**Related concepts:**

➡ Data transformation support for JDBC (Application Programming)

| **Related reference:**

| [🔗 COBOL copybook types that map to Java data types \(Application Programming\)](#)

| [🔗 XML to JDBC data type mapping \(Database Administration\)](#)

| **Nulls**

| All data types include the null value. Distinct from all non-null values, the null value is a special value that denotes the absence of a (non-null) value.

| IMS does not support setting columns to null. However, IMS interprets certain cases as null. For instance, variable-length segments where a field lies outside the segment boundary and segments with multiple mappings where a specific mapping does not apply are interpreted as null.

| **Numbers**

| The numeric data types are categorized as exact numerics: binary integer and decimal; and approximate numerics: floating-point

| Binary integer includes small integer, large integer, and big integer. Binary numbers are exact representations of integers. Decimal numbers are exact representations of real numbers. Binary and decimal numbers are considered exact numeric types. Floating-point includes double precision. Floating-point numbers are approximations of real numbers and are considered approximate numeric types.

| All numbers have a sign, a precision, and a scale. If a column value is zero, the sign is positive. Decimal floating point has distinct values for a number and the same number with various exponents (for example: 0.0, 0.00, 0.0E5, 1.0, 1.00, 1.0000). The precision is the total number of binary or decimal digits excluding the sign. The scale is the total number of binary or decimal digits to the right of the decimal point. If there is no decimal point, the scale is zero.

| **Small integer (SMALLINT):**

| A *small integer* is a binary integer that occupies 2 bytes. The range of small integers is -32768 to +32767.

| **Large integer (INTEGER):**

| A *large integer* is a binary integer that occupies 4 bytes.

| The range of large integers is -2147483648 to +2147483647.

| **Big integer (BIGINT):**

| A *big integer* is a binary integer that occupies 8 bytes.

| The range of big integers is -9223372036854775808 to +9223372036854775807.

| **Double precision floating-point (DOUBLE or FLOAT):**

| A *double precision floating-point* number is a long (64 bits) floating-point number.

| DOUBLE and FLOAT are not supported by SQL for COBOL.

The range of double precision floating-point numbers is about  $-7.2E+75$  to  $7.2E+75$ . In this range, the largest negative value is about  $-5.4E-79$ , and the smallest positive value is about  $5.4E-079$ .

### **Decimal:**

A *decimal* number is a packed decimal number with an implicit decimal point.

The position of the decimal point is determined by the precision and the scale of the number. The scale, which is the number of digits in the fractional part of the number, cannot be negative or greater than the precision. The maximum precision is 31 digits.

All values of a decimal column have the same precision and scale. The range of a decimal variable or the numbers in a decimal column is  $-n$  to  $+n$ , where  $n$  is the largest positive number that can be represented with the applicable precision and scale. The maximum range is  $1 - 10^{31}$  to  $10^{31} - 1$ .

### **Numeric host variables:**

The COBOL format for binary numeric data is USAGE BINARY. BINARY, COMP, and COMP-4 are synonyms. Binary-format numbers occupy 2, 4, or 8 bytes of storage.

For COBOL, COMP-1 refers to short floating-point format and COMP-2 refers to long floating-point format, which occupy 4 and 8 bytes of storage, respectively.

In COBOL, decimal numbers can be represented in the following formats:

- Packed decimal format, denoted by USAGE PACKED-DECIMAL or COMP-3
- External decimal format, denoted by USAGE DISPLAY with SIGN LEADING SEPARATE

### **Character strings**

A *character string* is a sequence of bytes. The length of the string is the number of bytes in the sequence. If the length is zero, the value is called the *empty string*. The empty string should not be confused with the null value.

### **Fixed-length character strings:**

When fixed-length character strings, columns, and variables are defined, the length attribute is specified, and all values have the same length. For a fixed-length character string, the length attribute must be between 1 and 255 inclusive.

### **Datetime values**

Datetime values are neither strings nor numbers. Nevertheless, datetime values can be used in certain arithmetic and string operations and are compatible with certain strings.

Moreover, strings can represent datetime values.

### **Date:**

A *date* is a three-part value (year, month, and day) designating a point in time using the Gregorian calendar, which is assumed to have been in effect from the year 1 A.D.

The range of the year part is 0001 to 9999. The range of the month part is 1 to 12. The range of the day part is 1 to 28, 29, 30, or 31, depending on the month and year.<sup>1</sup>

The internal representation of a date is a string of 4 bytes. Each byte consists of two packed decimal digits. The first 2 bytes represent the year, the third byte the month, and the last byte the day.

A character-string representation must have an actual length that is not greater than 255 bytes.

#### **Time:**

A *time* is a three-part value (hour, minute, and second) designating a time of day using a 24-hour clock. The range of the hour part is 0 to 24. The range of the minute and second parts is 0 to 59. If the hour is 24, the minute and second parts are both zero.

The internal representation of a time is a string of 3 bytes. Each byte consists of two packed decimal digits. The first byte represents the hour, the second byte the minute, and the last byte the second.

A character-string representation must have an actual length that is not greater than 255 bytes.

#### **Timestamp:**

A *timestamp* is a six-part or seven-part value (year, month, day, hour, minute, second, and optional fractional second) with an optional time zone specification, that represents a date and time.

In Java, timestamp maps with the type `java.sql.Timestamp`.

In COBOL, timestamp is a character string with an application-defined length,

#### **Datetime host variables:**

Character-string host variables are normally used to contain date, time, and timestamp values.

## **Assignment and comparison**

The basic operations of SQL are assignment and comparison.

Assignment operations are performed during the execution of statements such as INSERT and UPDATE statements. In addition, when a function is invoked or a stored procedure is called, the arguments of the function or stored procedure are assigned. Comparison operations are performed during the execution of statements that include predicates and other language elements such as ORDER BY.

The basic rule for both operations is that data types of the operands must be compatible.

---

1. Historical dates do not always follow the Gregorian calendar. Dates between 1582-10-04 and 1582-10-15 are accepted as valid dates although they never existed in the Gregorian calendar.

The following table shows the compatibility of data types for assignments and comparisons.

Table 160. Data type compatibility for assignments and comparisons

Operand	BYTES	SHORT	INT	LONG	DOUBLE	BIT	CHAR	PACKED	ZONED	DATE	TIME	FLOAT	TIMESTAMP
BYTES	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	No
SHORT	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	No
INT	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	No
LONG	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	No
DOUBLE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	No
BIT	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	No
CHAR	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
PACKED	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	No
ZONED	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	No
DATE	No	No	No	No	No	No	Yes	No	No	Yes	Yes	No	Yes
TIME	No	No	No	No	No	No	Yes	No	No	Yes	Yes	No	Yes
FLOAT	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	No
TIMESTAMP	No	No	No	No	No	No	Yes	No	No	Yes	Yes	No	Yes

**Notes:**

- LOBs and bit data are not supported.
- The compatibility of datetime values is limited to assignment and comparison:
  - Datetime values can be assigned to string columns and to string variables.
  - A valid string representation of a date can be assigned to a date column or compared to a date.
  - A valid string representation of a time can be assigned to a time column or compared to a time.
  - A valid string representation of a timestamp can be assigned to a timestamp column or compared to a timestamp.
- Character strings can be assigned to XML columns.

**String assignments**

There are two types of string assignments; storage assignment and retrieval assignment.

- Storage assignment* is when a value is assigned to a column.
- Retrieval assignment* is when a value is assigned to a variable.

The rules differ for storage and retrieval assignment.

**Character string assignment:**

The rules for storage and retrieval assignment apply when both the source and the target are strings.

*Storage assignment:*

The basic rule for character storage assignment is that the length of a string that is assigned to a column or parameter of a function must not be greater than the length attribute of the column or the parameter.

Trailing blanks are included in the length of the string. When the length of the string is greater than the length attribute of the column or the parameter, the following actions occur:

- If all of the trailing characters that must be truncated to make a string fit the target are blanks and the string is a character or graphic string, the string is truncated and assigned without warning.
- Otherwise, the string is not assigned and an error occurs to indicate that at least one of the excess characters is non-blank.

When a string is assigned to a fixed-length column or parameter and the length of the string is less than the length attribute of the target, the string is padded to the right with the necessary number of blanks.

*Retrieval assignment:*

In COBOL, the length of a string that is assigned to a host variable can be greater than the length attribute of the variable. When the length of the string is greater than the length of the variable, the string is truncated on the right by the necessary number of characters.

When truncation occurs, the value W is assigned to the SQLIMSWARN1 field of the SQLIMSCA.

When a character string is assigned to a fixed-length variable and the length of the string is less than the length attribute of the target, the string is padded to the right with the necessary number of blanks.

## **String comparisons**

String comparisons can occur with binary string, character strings.

### **Character string comparisons:**

Two strings are compared by comparing the corresponding bytes of each string. If the strings do not have the same length, the comparison is made with a temporary copy of the shorter string that has been padded on the right with blanks so that it has the same length as the other string.

Two strings are equal if they are both empty or if all corresponding bytes are equal. An empty string is equal to a blank string. If two strings are not equal, their relationship (that is, which has the greater value) is determined by the comparison of the first pair of unequal bytes from the left end of the strings.

## **Constants**

A *constant* (also called a *literal*) specifies a value. Constants are classified as string constants or numeric constants. Numeric constants are further classified as integer, floating-point, decimal, or decimal floating-point. String constants are classified as character or binary.



All constants have the attribute NOT NULL. A negative sign in a numeric constant with a value of zero is ignored, except for a decimal floating-point constant.

### Integer constants

An *integer constant* specifies an integer as a signed or unsigned number with a maximum of 19 digits that does not include a decimal point.

The data type of an integer constant is large integer if its value is within the range of a large integer. The data type of an integer constant is big integer if its value is outside the range of a large integer, but within the range of a big integer. A constant that is defined outside the range of big integer values is considered a decimal constant.

*Examples:*

64    -15    +100    32767    720176

In syntax diagrams, the term *integer* is used for a large integer constant that must not include a sign.

### Floating-point constants

A *floating-point constant* specifies a double-precision floating-point number as two numbers separated by an E.

The first number can include a sign and a decimal point. The second number can include a sign but not a decimal point. The value of the constant is the product of the first number and the power of 10 specified by the second number. It must be within the range of floating-point numbers. The number of characters in the constant must not exceed 30. Excluding leading zeros, the number of digits in the first number must not exceed 17 and the number of digits in the second must not exceed 2.

*Examples:* The following floating-point constants represent the numbers '150', '200000', -0.22, and '500':

15E1    2.E5    -2.2E-1    +5.E+2

### Decimal constants

A *decimal constant* is a signed or unsigned number of no more than 31 digits and either includes a decimal point or is not within the range of binary integers.

The precision is the total number of digits, including those, if any, to the right of the decimal point. The total includes all leading and trailing zeros. The scale is the number of digits to the right of the decimal point, including trailing zeros.

*Examples:* The following decimal constants have, respectively, precisions and scales of 5 and 2; 4 and 0; 2 and 0; and 23 and 2:

025.50    1000.    -15.    +3758933333333333333333.33

### Character string constants

A *character string constant* specifies a varying-length character string. There is one form of character string constant.

- A sequence of characters that starts and ends with a string delimiter.

*Examples:*

'10/14/2013'    '32'    'DON''T CHANGE'    ''

The right most string in the example ("") represents an empty character string constant, which is a string of zero length.

For COBOL, only character string EBCDIC 037 is supported.

## Field names

The meaning of a field name depends on its context.

A field name can be used to:

- Identify a field.
- Specify values of the field, as in the following contexts:
  - In an *ORDER BY* clause, a field name specifies all values in the intermediate result table to which the clause is applied. For example, *ORDER BY HOSPNAME* orders an intermediate result table by the values of the field *HOSPNAME*.
  - In a *search condition*, a field name specifies a value for each row or group to which the construct is applied. For example, when the search condition *CODE = 20* is applied to some row, the value specified by the field name *CODE* is the value of the field *CODE* in that row.
- Provide a field name for an expression to temporarily rename a field, or as in the *AS* clause in the *select-clause*.

### Qualified field names

A qualifier for a field name is a segment name.

Where a qualifier is optional, it can serve two purposes. See “Field name qualifiers to avoid ambiguity” and for details.

### Field name qualifiers to avoid ambiguity

In the context of an *ORDER BY* clause, an expression, or a search condition, a field name refers to values of a field in some segment or view in a *DELETE* or *UPDATE* statement or *table-reference* in a *FROM* clause.

One reason for qualifying a field name is to designate the object from which the field comes.

**Table designators:** A qualifier that designates a specific object table is called a *table designator*. The clause that identifies the object tables also establishes the table designators for them. For example, the object tables of an expression in a *SELECT* statement are named in the *FROM* clause that follows it, as in the following statement:

```
SELECT Z.HOSPCODE, WARDNO, WARDLL
FROM PCB01.HOSPITAL Z, PCB01.WARD
WHERE Z.HOSPNAME = 'ALEXANDRIA'
AND Z.HOSPCODE = 'R1210010000A'
```

Table designators in the *FROM* clause are established as follows:

- A name that follows a table or view name is both a correlation name and a table designator. Thus, *Z* is a table designator and qualifies the first column name in the select list.
- An exposed table or view name is a table designator. Thus, the qualified table name, *PCB01.WARD* is a table designator and qualifies the second column name in the select list.

## References to variables

A *variable* in an SQL statement specifies a value that can be changed when the SQL statement is executed. There are several types of variables used in SQL statements.

### host variable

Host variables are defined by statements of a host language. For more information about how to refer to host variables, see “References to host variables.”

### parameter marker

Parameter markers are specified in an SQL statement that is dynamically prepared instead of host variables. For more information about parameter markers, see Parameter markers in the PREPARE statement.

Unless otherwise noted, the term *host variable* in syntax diagrams is used to describe where a host variable or parameter marker can be used.

## References to host variables

Host variables are defined directly by statements of the host language. A *host-variable* in an SQL statement must identify a host variable that is described in the program according to the rules for declaring host variables. Host variables cannot be referenced in dynamic SQL statements; parameter markers must be used instead.

A *host variable* is a data element in COBOL.

A host-variable in an SQL statement must identify a host variable that is described in the program according to the rules for declaring host variables.

The term *host-variable*, as used in the syntax diagrams, shows a reference to a host variable. In the INTO clause of a FETCH statement, a host variable is an output variable to which a value is assigned by IMS. A host variable can also be an input variable which provides a value to IMS.

### variable references

The general form of a host variable reference is:

►► *:host-identifier* ◀◀

Each host identifier must be declared in the source program.

An SQL statement that refers to host variables must be within the scope of the declaration of those host variables. For host variables referred to in the SELECT statement of a cursor, the OPEN statement, and the DECLARE CURSOR statement have to be in the same scope.

All references to host variables must be preceded by a colon. If an SQL statement references a host variable without a preceding colon, the coprocessor issues an error for the missing colon or interprets the host variable as an unqualified column name, which might lead to unintended results. The interpretation of a host variable without a colon as a column name occurs when the host variable is referenced in a context in which a column name can also be referenced.

### Related concepts:

“Host variables in dynamic SQL” on page 630

## Host variables in dynamic SQL

In dynamic SQL statements, parameter markers are used instead of host variables.

A *parameter marker* is a question mark (?) that represents a position in a dynamic SQL statement where the application will provide a value.

```
INSERT INTO PCB01.DOCTOR (hospital_HOSPCODE, patient_patnum, ward_wardno, doctno,docname) VALUES (?,?,?,?,?)
```

### Parameter marker replacement:

Before the prepared statement is executed, each parameter marker in the statement is effectively replaced by its corresponding host variable. The replacement is an assignment operation in which the source is the value of the host variable and the target is a variable. The assignment rules are those described for assignment to a column in Assignment and comparison (Application Programming APIs).

## Host structures in COBOL

A *host structure* is a COBOL group that is referred to in an SQL statement.

As used here, the term *host structure* does not include an SQLIMSCA or SQIMSLDA.

The form of a host structure reference is identical to the form of a host variable reference. The reference :S1 is a host structure reference if S1 names a host structure.

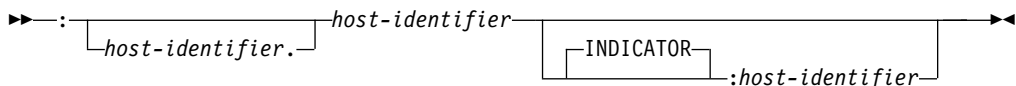
A host structure can be referred to in any context where a list of host variables can be referenced. A host structure reference is equivalent to a reference to each of the host variables contained within the structure in the order which they are defined in the host language structure declaration. The *n*th variable of the indicator array is the indicator variable for the *n*th variable of the host structure.

For example, if V1, V2, and V3 are declared as the variables within the structure S1, the following two statements are equivalent:

```
EXEC SQLIMS FETCH CURSOR1 INTO :S1;  
EXEC SQLIMS FETCH CURSOR1 INTO :V1, :V2, :V3;
```

In addition to structure references, individual host variables in COBOL can be referred to by qualified names. The qualified form is a host identifier followed by a period and another host identifier. The first host identifier must name a structure, and the second host identifier must name a host variable at the next level within that structure.

In COBOL, the syntax of *host-variable* is:



## Predicates

A *predicate* specifies a condition that is true, false, or unknown about a given row or group.

The types of predicates are:

►► *basic predicate* ◀◀

The following rules apply to predicates of any type:

- All values that are specified in the same predicate must be compatible.

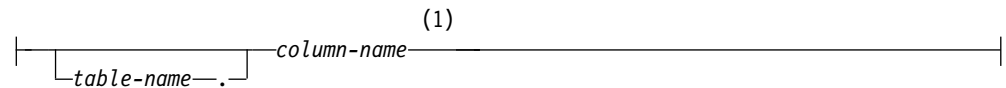
### Basic predicate

A *basic predicate* compares two values or compares a set of values with another set of values.

This statement can be embedded only in a COBOL application program.



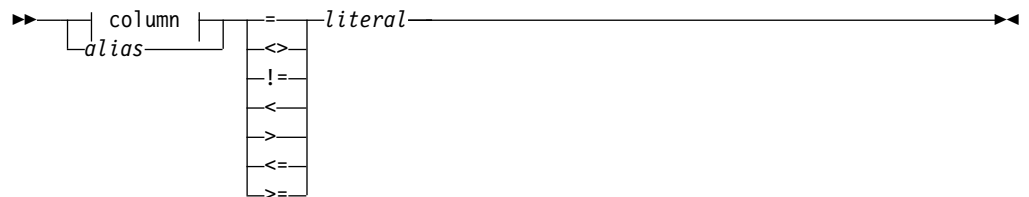
#### column:



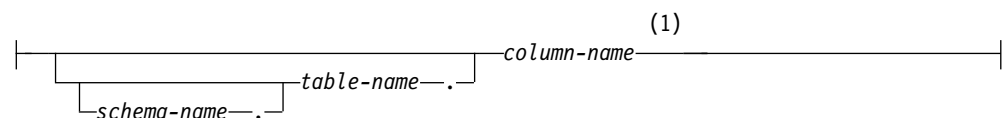
#### Notes:

- 1 You can have the same column name in multiple tables, but if the table is not qualified, each table must be searched for the column.

This statement is supported only for Java application programs.



#### column:



#### Notes:

- 1 You can have the same column name in multiple tables, but if the table is not qualified, each table must be searched for the column.

The result of the predicate depends on the operator, as in the following two cases:

- If the operator is =, the result of the predicate is:

- True if all pairs of corresponding value expressions evaluate to true.
- False if any one pair of corresponding value expressions evaluates to false.
- If the operator is <>, the result of the predicate (x1,x2,...,xn) <> (y1,y2,...,yn) is:
  - True, if and only if xi=yi evaluates to false for some value of i. (that is, there is at least one pair of non-null values, xi and yi, that are not equal to each other).
  - False, if and only if xi=yi evaluates to true for every value of i. (that is, (x1,x2,...,xn)=(y1,y2,...,yn) is true).

Table 161. For values *x* and *y*

Predicate	Is true only if ...
$x = y$	$x$ is equal to $y$
$x \neq y$	$x$ is not equal to $y$
$x < y$	$x$ is less than $y$
$x > y$	$x$ is greater than $y$
$x \leq y$	$x$ is less than or equal to $y$
$x \geq y$	$x$ is greater than or equal to $y$

Examples for values *x* and *y*:

```
HOSPCODE = '528671'
XINTEGER < 20000
HOSPNAME <> :VAR1
```

Example: List the hospital code and hospital name from the HOSPITAL segment where the hospital code is H5140070000H.

```
SELECT HOSPCODE, HOSPNAME
FROM PCB01.HOSPITAL
WHERE HOSPCODE = 'H5140070000H'
```

## BETWEEN predicate

The BETWEEN predicate determines whether a given value lies between two other given values that are specified in ascending order.

The BETWEEN predicate is supported only for Java application programs.

→  $\boxed{\text{column}}$   $\boxed{\text{alias}}$   $\boxed{\text{NOT}}$  BETWEEN  $\text{literal}$  AND  $\text{literal}$  →

### column:

$\boxed{\text{table-name}}$   $\boxed{\text{schema-name}}$   $\text{column-name}$  (1)

### Notes:

- 1 You can have the same column name in multiple tables, but if the table is not qualified, each table must be searched for the column.

Each of the predicate's two forms has an equivalent search condition, as shown in the following table:

Table 162. BETWEEN predicate and equivalent search conditions

BETWEEN predicate	Equivalent search condition
<i>column1</i> BETWEEN <i>value1</i> AND <i>value2</i>	<i>column1</i> >= <i>value1</i> AND <i>column1</i> <= <i>value2</i>
<i>column1</i> NOT BETWEEN <i>value1</i> AND <i>value2</i>	<i>column1</i> < <i>value1</i> OR <i>column1</i> > <i>value2</i>

Search conditions are discussed in “Search conditions” on page 634.

If the operands include a mixture of datetime values and valid string representations of datetime values, all values are converted to the data type of the datetime operand.

Example: Consider the following predicate:

*A* BETWEEN *B* AND *C*

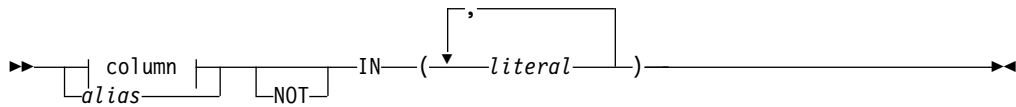
The following table shows the value of the predicate for various values of *A*, *B*, and *C*.

Value of <i>A</i>	Value of <i>B</i>	Value of <i>C</i>	Value of predicate
1,2, or 3	1	3	true
0 or 4	1	3	false
null	any value	any value	false

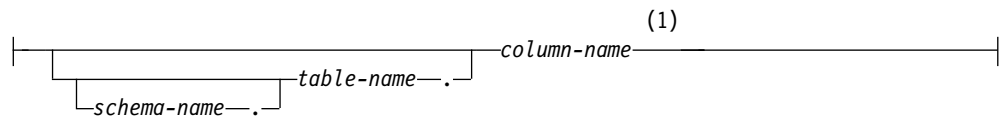
### IN predicate

The IN predicate compares a value or values with a set of values.

The IN predicate is supported only for Java application programs.



#### column:



#### Notes:

- 1 You can have the same column name in multiple tables, but if the table is not qualified, each table must be searched for the column.

The IN predicate is equivalent to the quantified predicate as follows:

Table 163. IN predicate and equivalent quantified predicates

IN predicate	Equivalent quantified predicate
<i>column1</i> IN ( <i>value1</i> , <i>value2</i> , <i>valuen</i> )	<i>column1</i> = <i>value1</i> or <i>column1</i> = <i>value2</i> or <i>column1</i> = <i>valuen</i>

Table 163. IN predicate and equivalent quantified predicates (continued)

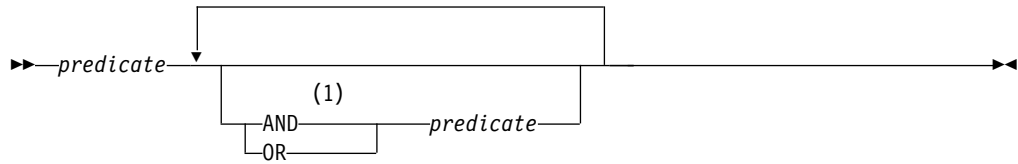
IN predicate	Equivalent quantified predicate
<code>column1 NOT IN (value1, value2, valuen)</code>	<code>column1 &lt;&gt; value1 and column1 &lt;&gt; value2 and column1 &lt;&gt; valuen</code>

Example: The following predicate is true for any row whose employee is in department D11, B01, or C01.

`WORKDEPT IN ('D11', 'B01', 'C01')`

## Search conditions

A *search condition* specifies a condition that is true or false about a given row or group. When the condition is true, the row or group qualifies for the results. When the condition is false or unknown, the row or group does not qualify.



### Notes:

- 1 Predicates across different tables must be connected by AND.

## Description

The result of a search condition is derived by application of the specified *logical operators* (AND, OR) to the result of each specified predicate. If logical operators are not specified, the result of the search condition is the result of the specified predicate.

AND and OR are defined in the following table, in which *P* and *Q* are any predicates:

Table 164. Truth table for AND and OR

<i>P</i>	<i>Q</i>	<i>P and Q</i>	<i>P or Q</i>
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

Search conditions within parentheses are evaluated first. The order in which operators at the same precedence level are evaluated is undefined to allow for optimization of search conditions.

## Example

For the following search condition, AND is applied first. After the application of AND, the OR operators could be applied in either order without changing the result. IMS can therefore select the order of applying the OR operators.



PATNUM > ? AND AGE > ? OR HOSPCODE = ? OR HOSPNAME = ?

*For COBOL only:*

PATNUM>:VAR1 AND AGE>:VAR2 OR HOSPCODE=:VAR3 OR HOSPNAME=:VAR4

**Related concepts:**

“Predicates” on page 630

---

## SQL statements

This section contains syntax diagrams, semantic descriptions, rules, and examples of the use of the SQL statements.

The SQL support for COBOL provides the underlying SQL functions for the IBM IMS Data Provider for Microsoft .NET. All SQL statements supported for COBOL application programs and their syntax and rules also apply to .NET applications.

*Table 165. SQL statements*

SQL statement	Function	Supported application program type
“CLOSE” on page 638	Closes a cursor	COBOL, .NET
“DECLARE CURSOR” on page 639	Defines an SQL cursor	COBOL, .NET
“DECLARE STATEMENT” on page 640	Declares names used to identify prepared SQL statements	COBOL, .NET
“DELETE” on page 641	Deletes one or more rows from a table	COBOL, .NET, Java
“DESCRIBE OUTPUT” on page 642	Describes the result columns of a prepared statement	COBOL, .NET
“EXECUTE” on page 643	Executes a prepared SQL statement	COBOL, .NET
“FETCH” on page 645	Positions the cursor, returns data, or both positions the cursor and returns data	COBOL, .NET
“INCLUDE” on page 646	Inserts declarations into a source program	COBOL, .NET
“INSERT” on page 647	Inserts one or more rows into a table	COBOL, .NET, Java
“OPEN” on page 651	Opens a cursor	COBOL, .NET
“PREPARE” on page 653	Prepares an SQL statement (with optional parameters) for execution	COBOL, .NET
“SELECT” on page 655	Specifies the SELECT statement of the cursor	COBOL, .NET, Java
“UPDATE” on page 665	Updates the values of one or more columns in one or more rows of a table	COBOL, .NET, Java
“WHENEVER” on page 668	Defines actions to be taken on the basis of SQL return codes	COBOL, .NET

## How SQL statements are invoked

SQL statements are invoked in different ways depending on whether the statement is an executable or nonexecutable statement or the *select-statement*.

The SQL statements are classified as *executable* or *nonexecutable*. The description of each statement includes a heading on invocation that indicates whether or not the statement is executable.

*Executable statements* can be invoked in the following ways:

- Dynamically prepared and executed in an application program
- Issued interactively

A *nonexecutable statement* can only be embedded in an application program.

### Using an SQL statement in an application program

You can include SQL statements in a source program that will be submitted to the IMS coprocessor. An SQL statement can be placed anywhere in the application program where a host language statement is allowed. Each statement must be preceded by a keyword (or keywords) to indicate that the statement is an SQL statement.

- In COBOL, each SQL statement must be preceded by the keywords EXEC SQLIMS.

***Executable statements:*** An executable SQL statement in an application program is executed every time a statement of the host language would be executed if specified in the same place. (Thus, for example, a statement within a loop is executed every time the loop is executed, and a statement within a conditional construct is executed only when the condition is satisfied.)

An SQL statement can contain references to host variables. A host variable referred to in this way can be used in one of two ways:

#### ***As input***

The current value of the host variable is used in the execution of the statement.

#### ***As output***

The variable is assigned a new value as a result of executing the statement.

In particular, all references to host variables in predicates are effectively replaced by current values of the variables; that is, the variables are used as input. The treatment of other references is described individually for each statement.

The successful or unsuccessful execution of the statement is indicated by setting the SQLIMSCODE and SQLIMSSTATE fields in the included SQLIMSCA. You must therefore follow all executable statements by a test of SQLIMSCODE or SQLIMSSTATE. Alternatively, you can use the WHENEVER statement (which is itself nonexecutable) to change the flow of control immediately after the execution of an SQL statement.

***Nonexecutable statements:*** A nonexecutable statement is processed only by the coprocessor. The coprocessor reports any errors encountered in the statement. The statement is never executed, and acts as a no-operation if placed among executable statements of the application program. Therefore, do not follow such statements with a test of an SQL return code.

## Dynamic preparation and execution

Your application program can dynamically build an SQL statement in the form of a character string placed in a host variable.

The statement so constructed can be prepared for execution by means of the (embedded) statement PREPARE and executed by means of the (embedded) statement EXECUTE.

A statement that is going to be prepared must not contain references to host variables. It can instead contain parameter markers. (See Parameter markers in the description of the PREPARE statement for rules concerning parameter markers.) When the prepared statement is executed, the parameter markers are effectively replaced by current values of the host variables specified in the EXECUTE statement. (See the EXECUTE statement for rules concerning this replacement.) After it is prepared, a statement can be executed several times with different values of host variables.

The successful or unsuccessful execution of the statement is indicated by the values returned in the SQLIMSCODE and SQLIMSSTATE fields in the SQLIMSCA after the EXECUTE (or EXECUTE IMMEDIATE) statement. You should check the fields as described above for embedded statements.

## Dynamic invocation of a SELECT statement

Your application program can dynamically build a SELECT statement in the form of a character string placed in a host variable.

The statement so constructed can be prepared for execution by means of the (embedded) statement PREPARE, and referred to by a (nonexecutable) statement DECLARE CURSOR. The statement is then executed every time you open the cursor by means of the (embedded) statement OPEN. After the cursor is open, you can retrieve the result table a row at a time by successive executions of the (embedded) SQL FETCH statement.

The SELECT statement used in that way must not contain references to host variables. It can instead contain parameter markers. (See "Notes" in "PREPARE" on page 653 for rules concerning parameter markers.) The parameter markers are effectively replaced by the values of the host variables specified in the OPEN statement. (See "OPEN" on page 651 for rules concerning this replacement.)

The successful or unsuccessful execution of the SELECT statement is indicated by the values returned in the SQLIMSCODE and SQLIMSSTATE fields in the SQLIMSCA after the OPEN. You should check the fields as described above for embedded statements.

## Detecting and processing error and warning conditions in host language applications

Errors and warnings conditions in host language applications can be checked by using the SQLIMSCODE or SQLIMSSTATE host variables or by using the SQLIMSCA.

Each host language provides a mechanism for handling diagnostic information.

In COBOL, an application program that contains executable SQL statements must provide a stand-alone integer variable named *SQLIMSCODE*.

## SQLIMSSTATE:

IMS sets SQLIMSSTATE after each SQL statement is executed. IMS returns values that conform to the error specification in the SQL standard. Thus, application programs can check the execution of SQL statements by testing SQLIMSSTATE instead of SQLIMSCODE.

SQLIMSSTATE provides application programs with common codes for common error conditions (the values of SQLIMSSTATE are product-specific if the error or warning is product-specific). Furthermore, SQLIMSSTATE is designed so that application programs can test for specific errors or classes of errors.

In the case of a LOOP statement, the SQLIMSSTATE is set after the END LOOP portion of the LOOP statement completes. With the REPEAT statement, the SQLIMSSTATE is set after the UNTIL and END REPEAT portions of the REPEAT statement completes.

## SQLIMSCODE:

The SQLIMSCODE is also set by IMS after each SQL statement is executed.

IMS conforms to the SQL standard as follows:

- If SQLIMSCODE = 0 and SQLIMSWARN0 is blank, execution was successful.
- If SQLIMSCODE = 100, "no data" was found. For example, a FETCH statement returned no data because the cursor was positioned after the last row of the result table.
- If SQLIMSCODE > 0 and not = 100, execution was successful with a warning.
- If SQLIMSCODE = 0 and SQLIMSWARN0 = 'W', execution was successful with a warning.
- If SQLIMSCODE < 0, execution was not successful.

In the case of a LOOP statement, the SQLIMSSTATE is set after the END LOOP portion of the LOOP statement completes. With the REPEAT statement, the SQLIMSSTATE is set after the UNTIL and END REPEAT portions of the REPEAT statement completes.

The SQL standard does not define the meaning of any other specific positive or negative values of SQLIMSCODE, and the meaning of these values is product specific.

## SQLIMSERRM:

The SQLIMSERRM is a variable length character string set by IMS after each SQL statement which contains the error message and length.

In COBOL, SQLIMSERRM includes SQLIMSERRML and SQLIMSERRMC. SQLIMSERRMC contains the SQL error message returned by IMS. It can be up to 255 characters. SQLIMSERRML is the length of the SQL error message.

### Related reference:

 [SQL codes \(Messages and Codes\)](#)

## CLOSE

The CLOSE statement closes a cursor.

## Invocation

This statement can be embedded only in a COBOL application program. It is an executable statement that cannot be dynamically prepared.

## Syntax

►►—CLOSE—*cursor-name*—————►►

## Description

The following keyword parameters are defined for the CLOSE statement:

*cursor-name*

Identifies the cursor to be closed. The cursor name must identify a declared cursor as explained in “DECLARE CURSOR.”

## Example

A cursor C1 is used to fetch one row at a time into the application program variables HOSPCODE, HOSPNAME, WARDNAME, and PATNAME. Finally, the cursor is closed. If the cursor is reopened, it is again located at the beginning of the rows to be fetched.

```
EXEC SQLIMS
  DECLARE C1 CURSOR FOR DYSQL
END-EXEC.

EXEC SQLIMS
  PREPARE DYSQL FROM :SELECT-STATEMENT
END-EXEC

EXEC SQLIMS OPEN C1 END-EXEC.

EXEC SQLIMS
  FETCH C1 INTO :HOSPCODE, :HOSPNAME, :WARDNAME, :PATNAME
END-EXEC.

IF SQLIMSCODE = 100
  PERFORM DATA-NOT-FOUND
ELSE
  PERFORM GET-REST-OF-HOSP
  UNTIL SQLIMSCODE IS NOT EQUAL TO ZERO.

EXEC SQLIMS CLOSE C1 END-EXEC.
```

## DECLARE CURSOR

The DECLARE CURSOR statement defines a cursor.

## Invocation

This statement can be embedded only in a COBOL application program. It is not an executable statement.

## Syntax

►►—DECLARE—*cursor-name*—NO SCROLL—FOR—*statement-name*—————►►

## Description

The following keyword parameters are defined for the DECLARE CURSOR statement:

### *cursor-name*

Names the cursor. The name must not identify a cursor that has already been declared in the source program.

### *statement-name*

Identifies the prepared *select-statement* that specifies the result table of the cursor whenever the cursor is opened. The *statement-name* must not be identical to a statement name specified in another DECLARE CURSOR statement of the source program. For an explanation of prepared SELECT statements, see "PREPARE" on page 653.

## Notes

*Cursors in COBOL programs:* In COBOL source programs, the DECLARE CURSOR statement must precede all statements that explicitly refer to the cursor by name.

## Examples

This example declares a cursor named C1 for statement named DYSQL.

```
EXEC SQLIMS
DECLARE C1 CURSOR FOR DYSQL
END-EXEC.

EXEC SQLIMS
PREPARE DYSQL FROM :SELECT-STATEMENT
END-EXEC

EXEC SQLIMS OPEN C1 END-EXEC.

EXEC SQLIMS
FETCH C1 INTO :HOSPCODE, :HOSPNAME, :WARDNAME, :PATNAME
END-EXEC.

IF SQLIMSCODE = 100
PERFORM DATA-NOT-FOUND
ELSE
PERFORM GET-REST-OF-HOSP
UNTIL SQLIMSCODE IS NOT EQUAL TO ZERO.

EXEC SQLIMS CLOSE C1 END-EXEC.
```

## DECLARE STATEMENT

The DECLARE STATEMENT statement is used for application program documentation. It declares names that are used to identify prepared SQL statements.

## Invocation

This statement can be embedded only in a COBOL application program. It is not an executable statement.

## Syntax

» DECLARE *statement-name* STATEMENT

## Description

The following keyword parameters are defined for the DECLARE statement:

*statement-name* **STATEMENT**

Lists one or more names that are used in your application program to identify prepared SQL statements.

## Example

This example shows the use of the DECLARE STATEMENT statement in a COBOL program. It declares a statement named UPD.

```
EXEC SQLIMS
  DECLARE UPD STATEMENT
END-EXEC.

EXEC SQLIMS
  PREPARE UPD FROM :SQLSTMT
END-EXEC.
IF SQLIMSCODE < 0
  MOVE '**** PREPARE ERROR ****' TO ERR-MSG1
  PERFORM 100-ERROR
ELSE
  PERFORM EXECUTE-STMT
END-IF
```

## DELETE

The DELETE statement deletes rows from a table.

The *searched* DELETE form is used to delete one or more rows, optionally determined by a search condition.

## Invocation

This statement can be embedded in a COBOL or Java application program or issued interactively.

- “DELETE syntax”
- “table syntax”

## DELETE syntax

» DELETE FROM *table* [WHERE *search-condition*]

## table syntax

» [*schema-name*.] *table-name*

## Description

The following keyword parameters are defined for the DELETE statement:

### DELETE FROM

Identifies the table from which rows are to be deleted.

#### *table-name*

The *table-name* defines the name of the table in your SQL query. The name must identify a segment in IMS.

#### *schema-name*

The *schema-name* defines the schema in your SQL query. In IMS, the schema name is the PCB name.

### WHERE

Specifies the rows to be deleted. You can omit the clause or give a search condition. When the clause is omitted, all the rows of the table are deleted.

#### *search-condition*

Is any search condition as described in “Search conditions” on page 634. Each *column-name* in the search condition must identify a column of the table.

The search condition is applied to each row of the table and the rows are those for which the result of the search condition is true are deleted.

## Example

From the table PCB01.HOSPITAL, delete all rows for hospitals Alexandria and Santa Teresa.

```
DELETE FROM PCB01.HOSPITAL WHERE HOSPNAME = 'Alexandria' OR HOSPNAME = 'Santa Teresa';
```

## DESCRIBE OUTPUT

The DESCRIBE OUTPUT statement obtains information about a prepared statement.

### Invocation

This statement can be embedded only in a COBOL application program. It is an executable statement that can be dynamically prepared.

### Syntax

►►—DESCRIBE—OUTPUT—*statement-name*—INTO—*descriptor-name*—◄◄

## Description

The following keyword parameters are defined for the DESCRIBE OUTPUT statement:

### OUTPUT

When a *statement-name* is specified, optional keyword to indicate that the describe will return information about the select list columns in the prepared SELECT statement.



*statement-name*

Identifies the prepared statement. When the DESCRIBE statement is executed, the name must identify a statement that has been prepared.

**INTO** *descriptor-name*

Identifies an SQL descriptor area (SQLIMSDA), which is described in “SQL descriptor area (SQLIMSDA)” on page 671. Use the INCLUDE SQLIMSDA statement to declare the SQLIMSDA in the application.

After the DESCRIBE statement is executed, all the fields in the SQLIMSDA except SQLN are either set by IMS or ignored.

## Example

Execute a DESCRIBE statement with the included SQLIMSDA. After DESCRIBE, SQLIMSD specifies the number of result fields returned. IF SQLIMSD equals 0, the statement is a non-SELECT statement such as INSERT, UPDATE, or DELTEE. If SQLIMSD is greater than zero, the statement is a SELECT statement and allocates storage for each result field and specify its address to the SQLIMSDATA field in the SQLIMSDA. Finally, FETCH the result dataset into the SQLIMSDA.

```
EXEC SQLIMS
  INCLUDE SQLIMSDA
END-EXEC

EXEC SQLIMS
  DECLARE C1 CURSOR FOR DYSQL
END-EXEC.

EXEC SQLIMS
  PREPARE DYSQL FROM :SELECT-STATEMENT
END-EXEC

EXEC SQLIMS
  DESCRIBE DYSQL INTO :SQLIMSDA
END-EXEC

IF SQLIMSD > 0
  EXEC SQLIMS OPEN C1 END-EXEC.
  .... /* Code to allocate the storage for each result field */
  .... /* Set the storage address to each SQLIMSDATA variable */
  EXEC SQLIMS FETCH C1 INTO :SQLIMSDA END-EXEC.

  IF SQLIMSCODE = 100
    PERFORM DATA-NOT-FOUND
  ELSE
    PERFORM GET-REST-OF-HOSP
    UNTIL SQLIMSCODE IS NOT EQUAL TO ZERO.

EXEC SQLIMS CLOSE C1 END-EXEC.
```

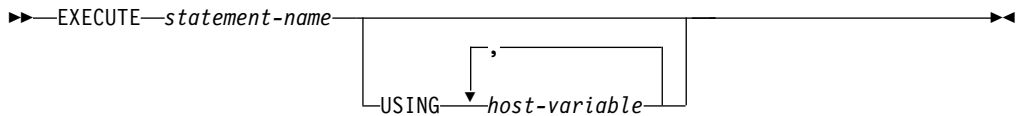
## EXECUTE

The EXECUTE statement executes a prepared SQL statement.

### Invocation

This statement can be embedded only in a COBOL application program. It is an executable statement that cannot be dynamically prepared.

## Syntax



## Description

The following keyword parameters are defined for the EXECUTE statement:

### *statement-name*

Identifies the prepared statement to be executed. *statement-name* must identify a statement that was previously prepared within the unit of work and the prepared statement must not be a SELECT statement.

### USING

Introduces a list of variables whose values are substituted for the parameter markers (question marks) in the prepared statement. (For an explanation of parameter markers, see “PREPARE” on page 653.) If the prepared statement includes parameter markers, you must include USING in the EXECUTE statement. USING is ignored if there are no parameter markers.

For more on the substitution of values for parameter markers, see Parameter marker replacement.

### *host-variable*

Identifies structures or variables that must be described in the application program in accordance with the rules for declaring host structures and variables. A reference to a structure is replaced by a reference to each of its variables. The number of variables must be the same as the number of parameter markers in the prepared statement. The *n*th variable supplies the value for the *n*th parameter marker in the prepared statement.

## Notes

### Parameter marker replacement:

Before the prepared statement is executed, each parameter marker in the statement is effectively replaced by its corresponding host variable. The replacement is an assignment operation in which the source is the value of the host variable and the target is a variable. The assignment rules are those described for assignment to a column in Assignment and comparison (Application Programming APIs).

## Example

In this example, an UPDATE statement is prepared from the variable SQLSTMT and executed.

```
EXEC SQLIMS  
  DELCARE UPD STATEMENT  
END-EXEC.  
  
EXEC SQLIMS  
  PREPARE UPD FROM :SQLSTMT  
END-EXEC.  
IF SQLIMSCODE < 0  
  MOVE '**** PREPARE ERROR ****' TO ERR-MSG1  
  PERFORM 100-ERROR  
ELSE
```

```

EXEC SQLIMS
EXECUTE UPD
END-EXEC
END-IF.

```

## FETCH

The FETCH statement positions a cursor on a row of its result table. It can return zero or one and assigns the values of the rows to host variables if there is a target specification.

### Invocation

This statement can be embedded only in a COBOL application program. It is an executable statement that cannot be dynamically prepared.

- "FETCH syntax"
- "single-row-fetch syntax"

### FETCH syntax

```

>> FETCH cursor-name
      |-----|
      | single-row-fetch |

```

### single-row-fetch syntax

```

>>
      |-----|
      | INTO host-variable |
      | INTO DESCRIPTOR descriptor-name |

```

### Description

The following keyword parameters are defined for the FETCH statement:

#### **INTO *host-variable***

Specifies a list of host variables. Each *host-variable* must identify a structure or variable that is described in the application program in accordance with the rules for declaring host structures and variables. A reference to a structure is replaced by a reference to each of its variables. The first value in the result row is assigned to the first host variable, the second value to the second host variable, and so on.

#### **INTO DESCRIPTOR *descriptor-name***

Identifies an SQLIMSDA that contains a valid description of the host output variables. Result values from the associated SELECT statement are returned to the application program in the output host variables.

Before the FETCH statement is processed, you must set the following fields in the SQLIMSDA:

- SQLIMSN to indicate the number of SQLIMSVAR occurrences provided in the SQIMSLDA
- SQLIMSABC to indicate the number of bytes of storage allocated in the SQLIMSDA
- SQLIMSD to indicate the number of variables used in the SQLIMSDA when processing the statement

- SQLIMSVAR occurrences to indicate the attributes of the variables

The SQLIMSDA must have enough storage to contain all SQLIMSVAR occurrences. Each SQLIMSVAR occurrence describes a host variable or buffer into which a value in the result table is to be assigned. For more information on the SQLIMSDA, which includes a description of the SQLIMSVAR and an explanation on how to determine the number of SQLIMSVAR occurrences, see “SQL descriptor area (SQLIMSDA)” on page 671.

SQLIMSD must be set to a value greater than or equal to zero and less than or equal to SQLIMSN.

*cursor-name*

Identifies the cursor to be used in the fetch operation. The cursor name must identify a declared cursor or an allocated cursor. When the FETCH statement is executed, the cursor must be in the open state.

### Example

*Example 1:* The FETCH statement fetches the results of the SELECT statement into the application program variables HOSPCODE and HOSPNAME. When no more rows remain to be fetched, the not found condition is returned.

```
EXEC SQLIMS
  DECLARE C1 CURSOR FOR DYSQL
END-EXEC.

EXEC SQLIMS
  PREPARE DYSQL FROM :SELECT-STATEMENT
END-EXEC

EXEC SQLIMS OPEN C1 END-EXEC.

EXEC SQLIMS FETCH C1 INTO :HOSPCODE, :HOSPNAME END-EXEC.

  IF SQLIMSCODE = 100
    PERFORM DATA-NOT-FOUND
  ELSE
    PERFORM GET-REST-OF-HOSP
    UNTIL SQLIMSCODE IS NOT EQUAL TO ZERO.

EXEC SQLIMS CLOSE C1 END-EXEC.
```

## INCLUDE

The INCLUDE statement inserts application code, including declarations and statements, into a source program.

### Invocation

This statement can be embedded only in a COBOL application program. It is not an executable statement.

### Syntax



## Description

The following keyword parameters are defined for the INCLUDE statement:

### SQLIMSCA

Indicates that the description of an SQL communication area (SQLIMSCA) is to be included. INCLUDE SQLIMSCA must not be specified more than once in the same application program. In COBOL, INCLUDE SQLIMSCA must be specified in the Working-Storage Section or the Linkage Section.

For a description of the SQLIMSCA, see “SQL communication area (SQLIMSCA)” on page 669.

### SQLIMSDA

Indicates that the description of an SQL descriptor area (SQLIMSDA) is to be included. For a description of the SQLIMSDA, see “SQL descriptor area (SQLIMSDA)” on page 671.

### *member-name*

Names a member of the partitioned data set to be the library input when your application program is prepared (with the IMS coprocessor). It must be an SQL identifier.

The member can contain any host language source statements and any SQL statements other than an INCLUDE statement. In COBOL, INCLUDE *member-name* must not be specified in other than the Data Division or the Procedure Division.

## Notes

When your application program is prepared (with the IMS coprocessor), the INCLUDE statement is replaced by source statements. Thus, the INCLUDE statement must be specified at a point in your application program where the resulting source statements are acceptable to the compiler.

The INCLUDE statement cannot refer to source statements that themselves contain INCLUDE statements.

## Example

Include an SQL communications area in a COBOL program.

```
EXEC SQLIMS INCLUDE SQLIMSCA END-EXEC.
```

## INSERT

The INSERT statement inserts rows into a table.

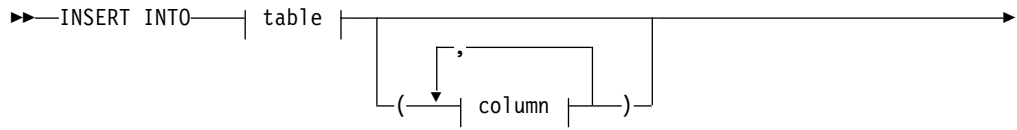
The INSERT via VALUES form is used to insert a single row into the table using the values provided or referenced.

## Invocation

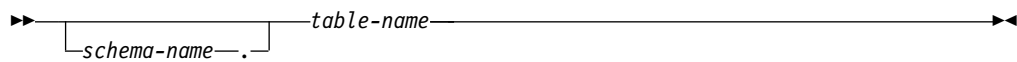
This statement can be embedded in a COBOL or Java application program or issued interactively. An INSERT can be embedded in an application program. It is an executable statement that can be dynamically prepared.

- “Syntax for COBOL” on page 648
- “Syntax for Java” on page 648

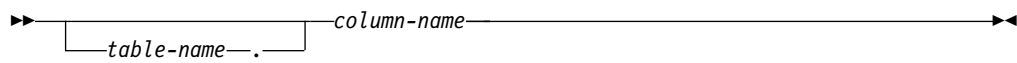
## Syntax for COBOL



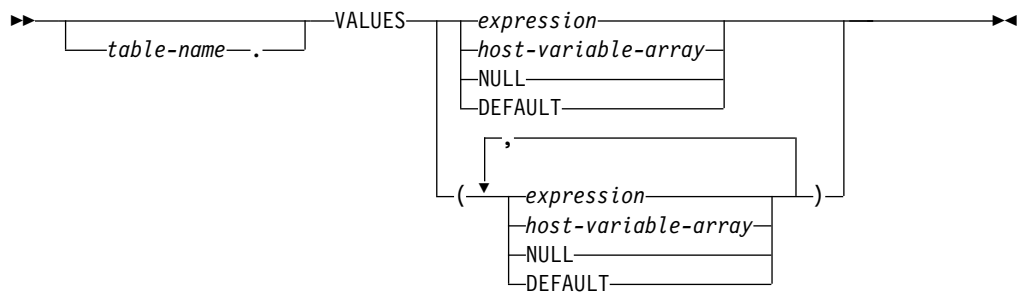
### table syntax



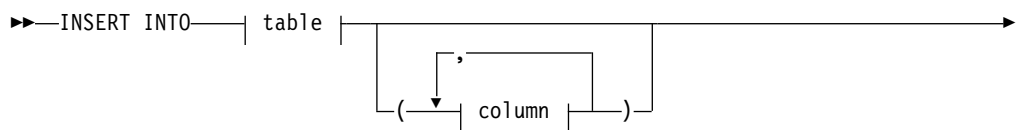
### column syntax



### multi-row-insert syntax



## Syntax for Java



## table:

`[schema-name.] table-name`

## column:

`[schema-name.] table-name. column-name (1)`

## Notes:

- 1 You can have the same column name in multiple tables, but if the table is not qualified, each table must be searched for the column.

## Description

The following keyword parameters are defined for the INSERT statement:

### INSERT INTO

Identifies the object of the INSERT statement.

#### **table-name**

The *table-name* defines the name of the table in your SQL query. The name must identify a segment in IMS.

#### **schema-name**

The *schema-name* defines the schema in your SQL query. In IMS, the schema name is the PCB name.

#### *column-name*

Specifies the columns for which insert values are provided. Each name must identify a field of the segment. The columns can be identified in any order, but the same column must not be identified more than one time.

Omission of the column list is an implicit specification of a list in which every column in the table is identified in the order identified by the metadata.

### VALUES

Specifies one new row in the form of a list of values. The number of values in the VALUES clause must be equal to the number of names in the column list. The first value is inserted in the first column in the list, the second value in the second column, and so on. The list of values must be enclosed in parentheses.

## Notes

### Insert rules:

Insert values must satisfy the following rules. If they do not, or if any other errors occur during the execution of the INSERT statement, no rows are inserted and the position of the cursors are not changed.

- *Length*. If the insert value of a column is a number, the column must be a numeric column with the capacity to represent the integral part of the number. If the insert value of a column is a string, the column must be either a string column with a length attribute at least as great as the length of the string, or a datetime column if the string represents a date, time, or timestamp.

- *Referential constraints.* When inserting a record in a table at a non-root level, you must specify values for all the foreign key fields of the table. Foreign key fields properly position the new record (or segment instance) to be inserted in the hierarchic path using standard SQL processing, similar to foreign keys in a relational database.
- *Omitting the column list.* When you omit the column list, you must specify a value for every column that was present in the table when the INSERT statement was bound or (for dynamic execution) prepared.

**Number of rows inserted:**

For COBOL, the value of SQLIMSERRD(3) in the SQLIMSCA is the number of rows inserted after an INSERT statement completes execution. For a complete description of the SQLIMSCA, see “SQL communication area (SQLIMSCA)” on page 669.

**Inserting binary fields:**

For COBOL, when inserting a binary field, you must use a parameter marker to specify the binary value. Not using a parameter marker would result in a 408 (data type not compatible) error.

**Examples**

**Inserting data at the root**

The following statement inserts a new HOSPITAL record:

```
INSERT INTO PCB01.HOSPITAL (HOSPCODE, HOSPNAME)
VALUES ('R1210050000A', 'O'MALLEY CLINIC')
```

**Inserting data into a specified table in a hierarchic path**

When inserting a record in a table at a non-root level, you must specify values for all the virtual foreign key fields of the table. The following statement inserts a new ILLNESS record under a specific HOSPITAL, WARD, and PATIENT table. In this example, the ILLNESS table has three virtual foreign keys HOSPITAL\_HOSPCODE, WARD\_WARDNO, and PATIENT\_PATNUM. The new record will be inserted if and only if there is a HOSPCODE in the HOSPITAL table with the value of 'H5140070000H', a WARD table with a WARDNO value of '01', and a PATIENT table with PATNUM value of 'R1210050000A'.

```
INSERT INTO PCB01.ILLNESS (HOSPITAL_HOSPCODE, WARD_WARDNO,
ILLNAME, PATIENT_PATNUM)
VALUES ('H5140070000H', '01', 'COLD', 'R1210050000A')
```

The following statement inserts a new WARD record under a specific HOSPITAL table. In this example, the WARD table has the virtual foreign key HOSPITAL\_HOSPCODE. The new record will be inserted if and only if there is a HOSPCODE in the HOSPITAL table with the value of 'H5140070000H'.

```
INSERT INTO PCB01.WARD (WARDNO, HOSPITAL_HOSPCODE, WARDNAME)
VALUES ('0001', 'H5140070000H', 'EMGY')
```

**Inserting data in a searchable field with subfields**

If a searchable field consists of subfields, you can insert data by setting all the subfield values such that the searchable field is completely populated.

**Inserting a record at a non-root level without specifying virtual foreign key fields**

In this statement, the WARD\_WARDNO virtual foreign key field is missing. The query will fail because it violates the referential integrity constraint that all foreign keys must be provided with legal values.

```
INSERT INTO PCB01.PATIENT (HOSPITAL_HOSPCODE, PATNAME, PATNUM)
VALUES ('HW3201', 'JOHN O'CONNOR', 'Z800')
```



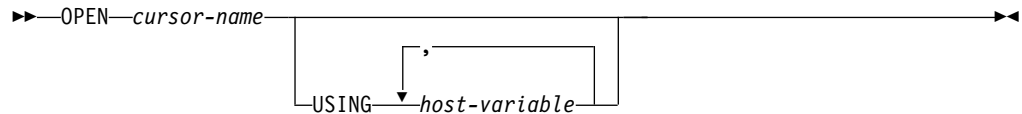
## OPEN

The OPEN statement opens a cursor so that it can be used to process rows from its result table.

### Invocation

This statement can be embedded only in a COBOL application program. It is an executable statement that cannot be dynamically prepared.

### Syntax



### Description

The following keyword parameters are defined for the OPEN statement:

#### *cursor-name*

Identifies the cursor to be opened. The *cursor-name* must identify a declared cursor as explained in “DECLARE CURSOR” on page 639. When the OPEN statement is executed, the cursor must be in the closed state.

The SELECT statement of the cursor is either one of the following types of SELECT statements:

- The prepared SELECT statement that is identified by the *statement-name* that is specified in the DECLARE CURSOR statement.

If the statement has not been successfully prepared, or is not a SELECT statement, the cursor cannot be successfully opened.

The result table of the cursor is derived by evaluating the SELECT statement. The evaluation uses the current values of any host variables that are specified in the USING clause of the OPEN statement. The rows of the result table can be derived during the execution of the OPEN statement. The cursor is placed in the open state and positioned before the first row of its result table.

#### USING

Introduces a list of host variables whose values are substituted for the parameter markers (question marks):

- If the DECLARE CURSOR statement included *statement-name*, the statement was prepared with a PREPARE statement. The host variables specified in the USING clause of the OPEN statement replace any parameter markers in the prepared statement. This reflects the typical use of the USING clause of the OPEN statement. For an explanation of parameter marker replacement, see “PREPARE” on page 653.

If the prepared statement includes parameter markers, you must use USING. If the prepared statement does not include parameter markers, USING is ignored.

#### *host-variable*

Identifies host structures or variables that must be described in the application program in accordance with the rules for declaring host structures and variables. When the statement is executed, a reference to a structure is replaced by a reference to each of its variables. The number of

variables must be the same as the number of parameter markers in the prepared statement. The *n*th variable corresponds to the *n*th parameter marker in the prepared statement. Where appropriate, locator variables can be provided as the source of values for parameter markers.

## Notes

**Closed state of cursors:** All cursors in an application process are in the closed state when:

- The application process is started.
- A new unit of work is started for the application process.

A cursor can also be in the closed state because:

- A CLOSE statement was executed.
- An error was detected that made the position of the cursor unpredictable.

To retrieve rows from the result table of a cursor, you must execute a FETCH statement when the cursor is open. The only way to change the state of a cursor from closed to open is to execute an OPEN statement.

**Parameter marker replacement:** Before the OPEN statement is executed, each parameter marker in the query is effectively replaced by its corresponding host variable. The replacement is an assignment operation in which the source is the value of the host variable and the target is a variable within IMS. The assignment rules are those described for assignment to a column in “Assignment and comparison” on page 624.

When the SELECT statement of the cursor is evaluated, each parameter marker in the statement is effectively replaced by the value of its corresponding host variable. For more on the process of replacement, see Parameter marker replacement.

## Examples

*Example 1:* Execute an OPEN statement, which places the cursor at the beginning of the rows to be fetched.

```
EXEC SQLIMS
  DECLARE C1 CURSOR FOR DYSQL
END-EXEC.

EXEC SQLIMS
  PREPARE DYSQL FROM :SELECT-STATEMENT
END-EXEC

EXEC SQLIMS OPEN C1 END-EXEC.

EXEC SQLIMS FETCH C1 INTO :HOSPCODE, :HOSPNAME, :WARDNAME, :PATNAME END-EXEC.

  IF SQLIMSCODE = 100
    PERFORM DATA-NOT-FOUND
  ELSE
    PERFORM GET-REST-OF-HOSP
    UNTIL SQLIMSCODE IS NOT EQUAL TO ZERO.

EXEC SQLIMS CLOSE C1 END-EXEC.
```

## PREPARE

The PREPARE statement creates an executable SQL statement from a string form of the statement. The character-string form is called a *statement string*. The executable form is called a *prepared statement*.

### Invocation

This statement can be embedded only in a COBOL application program. It is an executable statement that can be dynamically prepared.

### Syntax

►►—PREPARE—*statement-name*—FROM—*host-variable*—►►

### Description

The following keyword parameters are defined for the PREPARE statement:

#### *statement-name*

Names the prepared statement. If the name identifies an existing prepared statement, that prepared statement is destroyed. The name must not identify a prepared statement that is the SELECT statement of an open cursor.

#### FROM

Specifies the statement string. The statement string is the value of the identified *host-variable*.

#### *host-variable*

Must identify a host variable that is described in the application program in accordance with the rules for declaring variable-length string variables for statement string. The length of the SQL statement cannot be over 32767.

### Notes

**Rules for statement strings:** The value of the specified *statement-name* is called the *statement string*. The statement string must be declared with a varying-length character host variable. The first two bytes must contain the length of the SQL statement. The maximum length of the SQL statement is 32,767. For example:

```
01 STMTSTR.  
   49 STMTSTR-LEN PIC S9(4) COMP VALUE +180.  
   49 STMTSTR-TXT PIC X(180) VALUE SPACES.
```

The statement string must be one of the following SQL statements:

- DELETE
- INSERT
- SELECT
- UPDATE

The statement string must not:

- Begin with EXEC SQLIMS
- End with END-EXEC or a semicolon
- Include references to host variables

**Parameter markers:** Although a statement string cannot include references to host variables, it can include *parameter markers*. The parameter markers are replaced by the values of host variables when the prepared statement is executed. A parameter marker is a question mark (?) that appears where a host variable could appear if the statement string were a static SQL statement. For an explanation of how parameter markers are replaced by values, see “EXECUTE” on page 643 and “OPEN” on page 651.

**Error checking:** When a PREPARE statement is executed, the statement string is parsed and checked for errors. If the statement string is invalid, a prepared statement is not created and the error condition that prevents its creation is reported in the SQLIMSCA.

**Reference and execution rules:** Prepared statements can be referred to in the following kinds of statements, with the following restrictions shown:

**DESCRIBE**

No restrictions

**DECLARE CURSOR**

Must be SELECT when the cursor is opened

**EXECUTE**

Must not be SELECT

**Scope of a statement name:** The scope of a *statement-name* is the same as the scope of a *cursor-name*. See “DECLARE CURSOR” on page 639 for more information about the scope of a *cursor-name*.

## Examples

*Example 1:* Prepare a dynamic SELECT statement with a host variable on the PREPARE statement. The text of the SELECT statement is in a variable named SELECT-STATEMENT.

In the example, the statement text in host variable SELECT-STATEMENT is  
SELECT HOSPCODE, HOSPNAME, WARDNAME, PATNAME FROM  
PCB01.HOSPITAL, WARD,PATIENT.

```
EXEC SQLIMS  
DECLARE C1 CURSOR FOR DYSQL  
END-EXEC.
```

```
EXEC SQLIMS  
PREPARE DYSQL FROM :SELECT-STATEMENT  
END-EXEC
```

```
EXEC SQLIMS OPEN C1 END-EXEC.
```

```
EXEC SQLIMS FETCH C1 INTO :HOSPCODE, :HOSPNAME, :WARDNAME, :PATNAME END-EXEC.
```

```
IF SQLIMSCODE = 100  
PERFORM DATA-NOT-FOUND  
ELSE  
PERFORM GET-REST-OF-HOSP  
UNTIL SQLIMSCODE IS NOT EQUAL TO ZERO.
```

```
EXEC SQLIMS CLOSE C1 END-EXEC.
```

*Example 2:* Prepare a dynamic INSERT statement with parameter markers and is executed.

For the INSERT statement:

```
INSERT INTO PCB01.HOSPITAL HOSPCODE, HOSPNAME VALUES(?,?)
```

The following statement prepares and executes the INSERT statement with parameter marker. Before execution, the values for the parameter markers are read into the host variables S1, S2.

```
EXEC SQLIMS  
PREPARE DYSQL FROM :INSERT-STATEMENT  
END-EXEC
```

```
EXEC SQLIMS  
EXECUTE USING :S1, :S2  
END-EXEC.
```

## SELECT

The SELECT statement is used to retrieve data from one or more tables. The result is returned in a tabular result set.

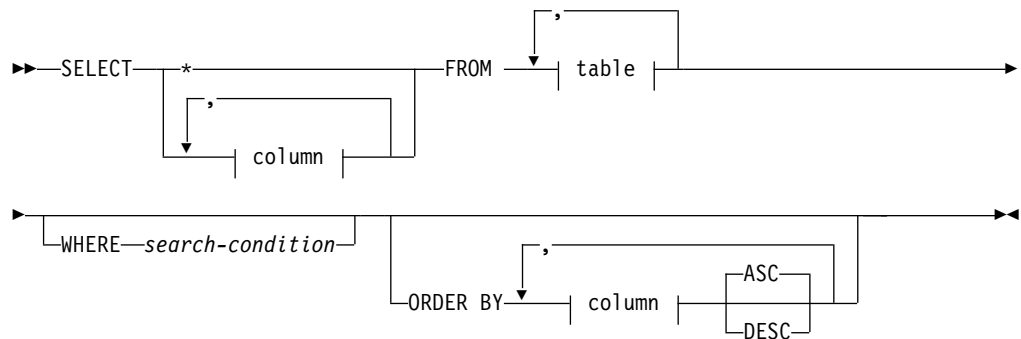
### Invocation

This statement can be used in COBOL or Java application programs, but the syntax is different.

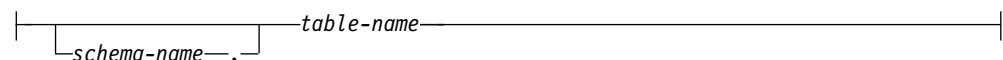
For COBOL application programs, this is an executable statement that cannot be dynamically prepared.

- Syntax for COBOL
- Syntax for Java

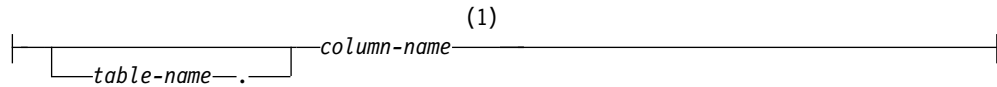
### Syntax for COBOL



### table:



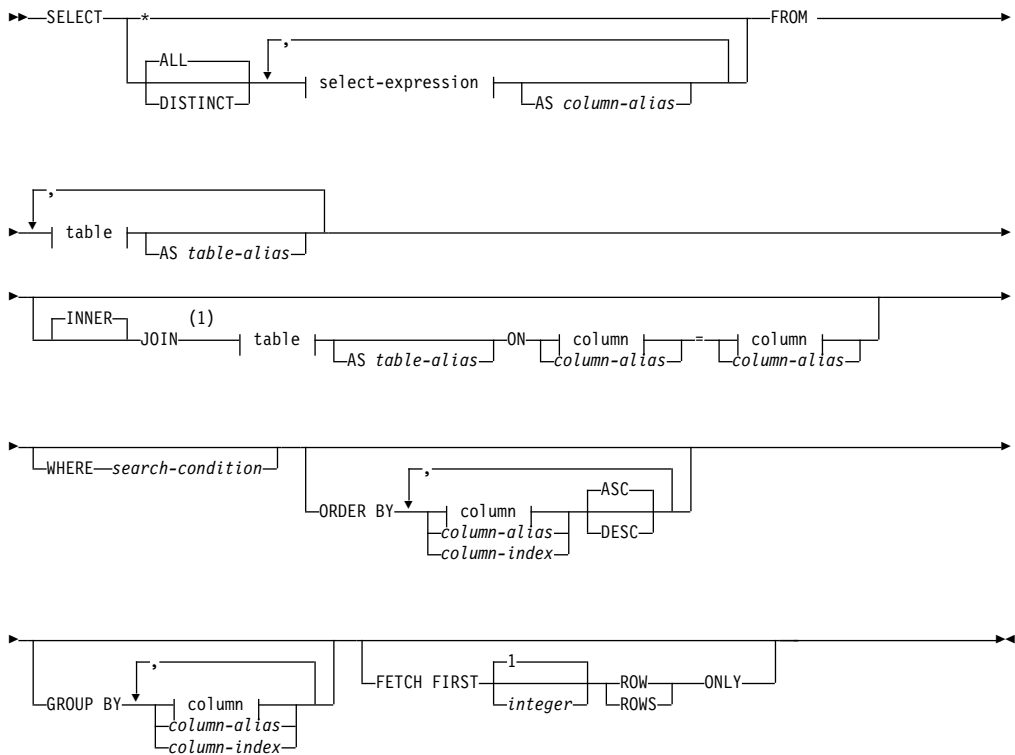
**column:**



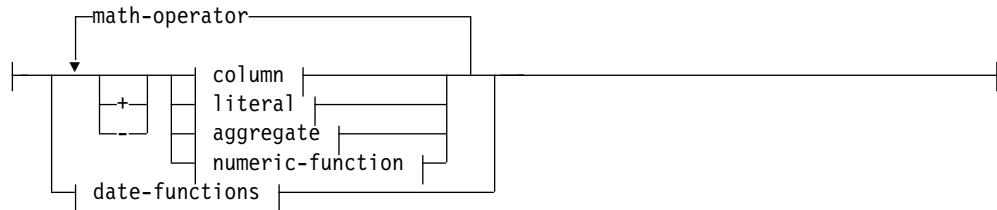
**Notes:**

- 1 You can have the same column name in multiple tables, but if the table is not qualified, an ambiguity check is performed to determine the table that the column belongs to.

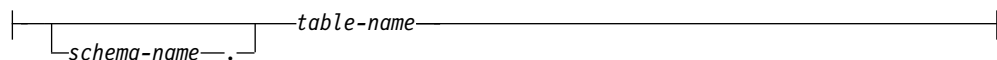
**Syntax for Java**



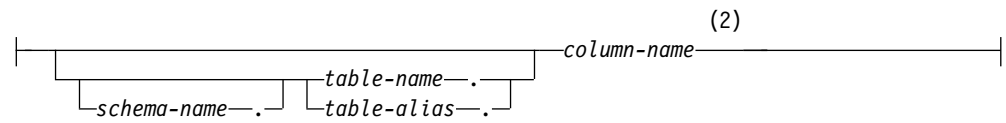
**select-expression:**



**table:**



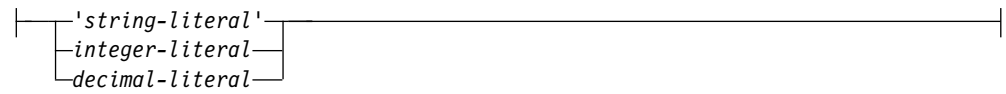
**column:**



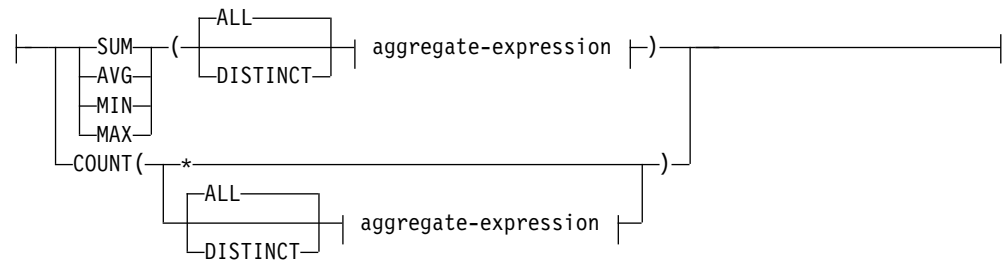
**math-operator:**



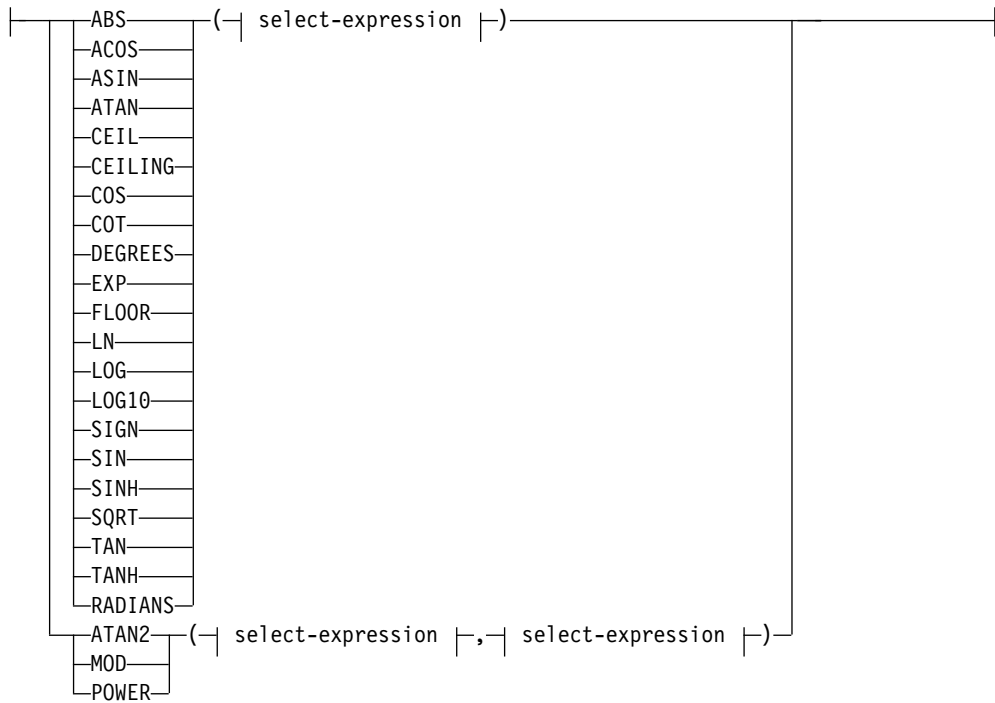
**literal:**



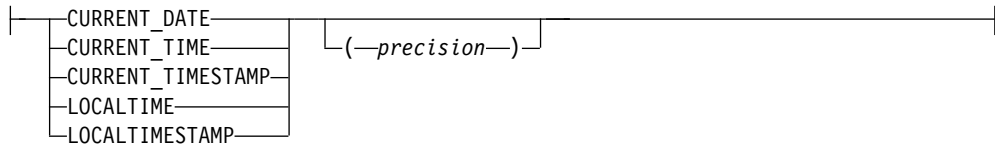
**aggregate:**



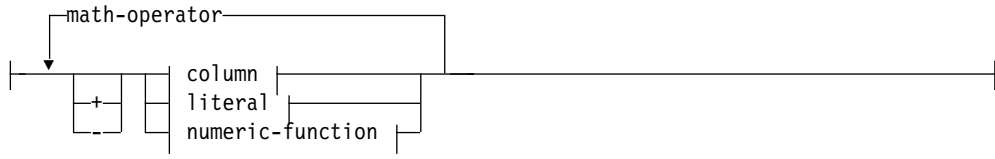
**numeric-function:**



**date-functions:**



**aggregate-expression:**



**Notes:**

- 1 JOIN tables must have referential integrity, expressed by the key field of a parent segment and the virtual foreign key of the dependent segment. You cannot specify both a comma separated list of tables and a JOIN statement.
- 2 You can have the same column name in multiple tables, but if the table is not qualified, each table must be searched for the column.

**Description**

The following keyword parameters are defined for the SELECT statement:

**ALL**

Retains all rows of the final result table and does not eliminate redundant duplicates. This is the default.



| **DISTINCT**

| Eliminates all but one of each set of duplicate rows of the final result table.  
| This keyword is valid only for Java application programs.

| Two rows are duplicates of one another only if each value in the first row is  
| equal to the corresponding value in the second row. For determining duplicate  
| rows, two null values are considered equal.

| **AS *column-alias***

| Names or renames the result column. The name must be unique. The AS  
| clause is not supported in COBOL or .NET application programs.

| **FROM *table-name***

| Identifies the table from which rows are to be retrieved. The name must  
| identify a segment in IMS.

| **AS *table-alias***

| Renames the table. The name must be unique. The AS clause is not supported  
| in COBOL or .NET application programs.

| **INNER JOIN**

| **JOIN**

| If a join operator is not specified, INNER is the default. The INNER JOIN  
| keyword selects all rows from both tables as long as there is a match between  
| the columns in both tables.

| **WHERE**

| Specifies the rows to be retrieved. You can omit the clause or give a search  
| condition. When the clause is omitted, all the rows of the table are retrieved.

| ***search-condition***

| Is any search condition as described in “Search conditions” on page 634.  
| Each *column-name* in the search condition must identify a column of the  
| table.

| The search condition is applied to each row of the table and the retrieved  
| rows are those for which the result of the search condition is true.

| **ORDER BY**

| The ORDER BY clause specifies an ordering of the rows of the result table.

| A column, *column-alias*, or a *column-index* that specifies the value that is to be  
| used to order the rows of the result of the table.

| The *column-index n* identifies the *n*th column of the result table.

| **ASC**

| Uses the values of the column, *column-alias*, or *column-index* in ascending  
| order. ASC is the default.

| **DESC**

| Uses the values of the column, *column-alias*, or *column-index* in descending  
| order.

| **GROUP BY**

| The GROUP BY clause specifies a result table that consists of a grouping of the  
| rows of intermediate result table that is the result of the previous clause.

| The result of GROUP BY is a set of groups of rows. In each group of more  
| than one row, all values of each column, *column-alias*, or *column-index* are equal,  
| and all rows with the same set of values of the column, *column-alias*, or  
| *column-index* are in the same group. For grouping, all null values for a  
| column, *column-alias*, or *column-index* are considered equal.

If your SELECT statement contains both aggregate and non-aggregate select-expressions, all of the non-aggregate select-expressions need to be in a GROUP BY clause.

***schema-name***

The *schema-name* defines the schema in your SQL query. In IMS, the schema name is the PCB name.

***table-name***

The *table-name* defines the name of the table in your SQL query. The name must identify a segment in IMS.

***table-alias***

The *table-alias* defines the alias that is defined in your SQL query that can be used in place of the *table-name*.

***column-name***

The *column-name* defines the name of the column in your SQL query.

***column-name***

The *column-name* defines the name of the column in your SQL query.

**'string-literal'**

A *'string-literal'* defines a static character string that is UTF-8 encoded.

***integer-literal***

An *integer-literal* defines an integer value within the range of -2,147,483,648 to 2,147,483,647.

***decimal-literal***

A *decimal-literal* defines a decimal value of double point precision.

**SUM**

The SUM function returns the sum of a set of numbers.

**AVG**

The AVG function returns the average of a set of numbers.

**MIN**

The MIN function returns the minimum value in a set of values.

**MAX**

The MAX function returns the maximum value in a set of values.

**COUNT**

The COUNT function returns the number of rows or values in a set of rows or values.

**numeric functions:**

**ABS**

The ABS function returns the absolute value of a number.

**ACOS**

The ACOS function returns the arc cosine of the argument as an angle, expressed in radians. The ACOS and COS functions are inverse operations.

**ASIN**

The ASIN function returns the arc sine of the argument as an angle, expressed in radians. The ASIN and SIN functions are inverse operations.

**ATAN**

The ATAN function returns the arc tangent of the argument as an angle, expressed in radians. The ATAN and TAN functions are inverse operations.

|  
|  
| **CEIL**

| **CEILING**

| The CEILING function returns the smallest integer value that is greater  
| than or equal to the argument.

| **COS**

| The COS function returns the cosine of the argument, where the argument  
| is an angle, expressed in radians. The COS and ACOS functions are inverse  
| operations.

| **COT**

| The COT function returns the cotangent of the argument, where the  
| argument is an angle, expressed in radians. The COT and TAN functions  
| are reciprocal operations.

| **DEGREES**

| The DEGREES function returns the number of degrees of the argument,  
| which is an angle, expressed in radians.

| **EXP**

| The EXP function returns a value that is the base of the natural logarithm  
| (e), raised to a power that is specified by the argument. The EXP and LN  
| functions are inverse operations.

| **FLOOR**

| The FLOOR function returns the largest integer value that is less than or  
| equal to the argument.

| **LN**

| **LOG**

| The LN and LOG function returns the natural logarithm of the argument.  
| The LN and EXP functions are inverse operations.

| **LOG10**

| The LOG10 function returns the common logarithm (base 10) of a number.

| **SIGN**

| The SIGN function returns an indicator of the sign of the argument.

| **SIN**

| The SIN function returns the sine of the argument, where the argument is  
| an angle, expressed in radians.

| **SINH**

| The SINH function returns the hyperbolic sine of the argument, where the  
| argument is an angle, expressed in radians.

| **SQRT**

| The SQRT function returns the square root of the argument.

| **TAN**

| The TAN function returns the tangent of the argument, where the  
| argument is an angle, expressed in radians.

| **TANH**

| The TANH function returns the hyperbolic tangent of the argument, where  
| the argument is an angle, expressed in radians.

| **RADIANS**

| The RADIANS function returns the number of radians for an argument  
| that is expressed in degrees.

### **ATAN2**

The ATAN2 function returns the arc tangent of x and y coordinates as an angle, expressed in radians.

### **MOD**

The MOD function divides the first argument by the second argument and returns the remainder.

### **POWER**

The POWER function returns the value of the first argument to the power of the second argument.

### **CURRENT\_DATE**

The CURRENT\_DATE special register specifies a date that is based on a reading of the time-of-day clock when the SQL statement is executed at the application.

### **CURRENT\_TIME**

The CURRENT\_TIME special register specifies a time that is based on a reading of the time-of-day clock when the SQL statement is executed at the application.

#### ***precision***

The *precision* specifies the fractions of a second. *precision* can range from 0 to 12. The default precision is 3.

### **CURRENT\_TIMESTAMP**

The CURRENT\_TIMESTAMP special register specifies a timestamp that is based on a reading of the time-of-day clock when the SQL statement is executed at the application.

#### ***precision***

The *precision* specifies the fractions of a second. *precision* can range from 0 to 12. The default precision is 6.

### **LOCALTIME**

The LOCALTIME special register specifies a time that is based on a reading of the time-of-day clock when the SQL statement is executed at the application.

#### ***precision***

The *precision* specifies the fractions of a second. *precision* can range from 0 to 12. The default precision is 3.

### **LOCALTIMESTAMP**

The LOCALTIMESTAMP special register specifies a timestamp that is based on a reading of the time-of-day clock when the SQL statement is executed at the application.

#### ***precision***

The *precision* specifies the fractions of a second. *precision* can range from 0 to 12. The default precision is 6.

## **Note**

- If you are selecting from multiple tables and the same column name exists in one or more of these tables, you must table-qualify the column or an ambiguity error will occur.
- The FROM clause must list all the tables you are selecting data from. The tables listed in the FROM clause must be in the same hierarchic path in the IMS database.
- Because there are multiple database PCBs in a PSB, queries must specify which PCB in a PSB to use. To specify which PCB to use, always qualify segments that

are referenced in the FROM clause of an SQL statement by prefixing the segment name with the PCB name. You can omit the PCB name only if the PSB contains only one PCB.

## Examples

### Selecting all fields with \* symbol

The following statement retrieves all fields for the PATIENT table:

```
SELECT *  
FROM PCB01.PATIENT
```

The following statement retrieves the hospital name from the HOSPITAL table and all fields from the WARD table:

```
SELECT HOSPITAL.HOSPNAME, WARD.*  
FROM PCB01.HOSPITAL, PCB01.WARD
```

### Selecting specified columns

The following statement retrieves the ward names and patient names from the WARD and PATIENT tables, respectively:

```
SELECT WARD.WARDNAME, PATIENT.PATNAME  
FROM PCB01.WARD, PATIENT
```

### Selecting with ORDER BY

The ORDER BY clause is used to sort the rows. By default, results are sorted by ascending numerical or alphabetical order. The following statement retrieves all distinct hospital names, sorted in alphabetical order:

```
SELECT DISTINCT HOSPNAME FROM PCB01.HOSPITAL  
ORDER BY HOSPNAME
```

The following statement retrieves all ward names sorted in alphabetical order, and the number of patients in each ward sorted in ascending numerical order. If two WARDNAME values in the ORDER BY compare are equal, the tiebreaker will be their corresponding PATCOUNT values (in this case, the row with the numerically smaller corresponding PATCOUNT value is displayed first).

```
SELECT WARDNAME, PATCOUNT FROM PCB01.WARD  
ORDER BY WARDNAME, PATCOUNT
```

Use the DESC qualifier to sort the query result in descending numerical or reverse alphabetical order. The following statement retrieves all patient names in reverse alphabetical order:

```
SELECT PATNAME FROM PCB01.PATIENT  
ORDER BY PATNAME DESC
```

Use the ASC qualifier to explicitly sort the query result in ascending numerical or reverse alphabetical order. The following statement retrieves all ward names sorted in ascending alphabetical order, and the number of patients in each ward sorted in descending numerical order:

```
SELECT WARDNAME, PATCOUNT FROM PCB01.WARD  
ORDER BY WARDNAME ASC, PATCOUNT DESC
```

### Selecting with GROUP BY

For SQL for the IMS Universal JDBC driver, the GROUP BY clause is used to return results for aggregate functions, grouped by distinct column values. The following statement returns the aggregated sum of all doctors in every ward in a hospital, grouped by distinct ward names :

```

SELECT WARDNAME, SUM(DOCCOUNT)
FROM PCB01.WARD
WHERE HOSPITAL_HOSPCODE = 'H5140070000H
GROUP BY WARDNAME

```

The following statement returns the hospital name, ward name, and the count of all patients in each ward in each hospital, grouped by distinct hospital names and sub-grouped by ward names:

```

SELECT HOSPNAME, WARDNAME, COUNT(PATNAME)
FROM PCB01.HOSPITAL, WARD, PATIENT
GROUP BY HOSPNAME, WARDNAME

```

### Selecting with DISTINCT

For SQL for the IMS Universal JDBC driver, the DISTINCT keyword is supported. The following statement retrieves all distinct patient names from the PATIENT table for SQL:

```

SELECT DISTINCT PATNAME
FROM PCB01.PATIENT

```

### Using the AS clause

For SQL for the IMS Universal JDBC driver, use the AS clause to rename the aggregate function column in the result set or any other field in the SELECT statement. The AS clause is not supported for SQL for COBOL application programs.

For SQL for the IMS Universal JDBC driver, the DISTINCT keyword is supported. The following statement returns the aggregate count of distinct patients in the PATIENT table with the alias of "PATIENTCOUNT":

```

SELECT COUNT(DISTINCT PATNAME)
AS PATIENTCOUNT
FROM PCB01.PATIENT

```

The following statement returns the aggregate count of distinct wards in all hospitals with the alias of "WARDCOUNT", sorted by the hospital names in alphabetical order, and grouped by distinct hospital names (under a renamed column alias "HOSPITALNAME"):

```

SELECT HOSPNAME AS HOSPITALNAME, COUNT(DISTINCT WARDNAME)
AS WARDCOUNT
FROM PCB01.HOSPITAL, WARD
GROUP BY HOSPNAME
ORDER BY HOSPNAME

```

### Example of SELECT using a parameter marker:

The following statement retrieves data based on the value that is supplied for the parameter for HOSPNAME:

```

SELECT * FROM PCB01.HOSPITAL WHERE HOSPNAME = ?

```

### Example of using the FETCH FIRST clause:

The following statement fetches the first *n* number of rows returned:

```

SELECT HOSPNAME FROM PCB01.HOSPITAL FETCH FIRST 3 ROWS ONLY

```

### Examples of invalid SELECT queries:


The following statement is invalid because the FROM clause is missing the WARD table:

```

SELECT WARD.WARDNAME, PATIENT.PATNAME
FROM PCB01.PATIENT

```

### Related reference:

 SQL aggregate functions supported by the IMS JDBC drivers (Application Programming)

## UPDATE

The UPDATE statement updates the values of specified columns in rows of a table.

The *searched* UPDATE form is used to update one or more rows optionally determined by a search condition.

### Invocation

This statement can be embedded in a COBOL or Java application program or issued interactively. An UPDATE can be embedded in an application program. It is an executable statement that can be dynamically prepared.

- "Syntax for COBOL"
- "Syntax for Java"

### Syntax for COBOL

#### update

```
►► UPDATE table SET assignment-clause  
└── WHERE search-condition ─┘
```

#### table:

```
└── schema-name ─┘ table-name
```

#### assignment clause:

```
└── column ─┘ = value
```

#### column:

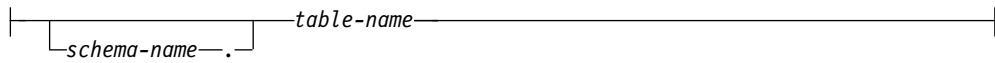
```
└── table-name ─┘ column-name
```

### Syntax for Java

#### update

```
►► UPDATE table SET assignment-clause  
└── WHERE search-condition ─┘
```

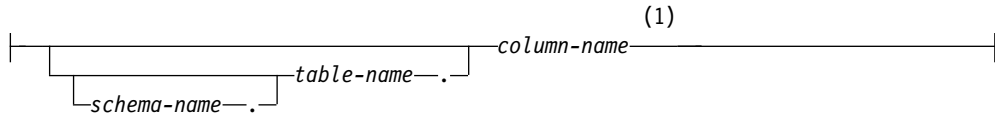
#### table:



**assignment clause:**



**column:**



**Notes:**

- 1 You can have the same column name in multiple tables, but if the table is not qualified, each table must be searched for the column.

**Description**

The following keyword parameters are defined for the UPDATE statement:

**UPDATE**

Identifies the object of the UPDATE statement.

***table-name***

The *table-name* defines the name of the table in your SQL query. The name must identify a segment in IMS.

***schema-name***

The *schema-name* defines the schema in your SQL query. In IMS, the schema name is the PCB name.

**SET**

Introduces the assignment of values to column names.

***column-name***

Identifies a column that is to be updated. *column-name* must identify a field of the specified segment.

**WHERE**

Specifies the rows to be updated. You can omit the clause or give a search condition. When the clause is omitted, all the rows of the table are updated.

***search-condition***

Is any search condition as described in "Search conditions" on page 634. Each *column-name* in the search condition must identify a column of the table.

The search condition is applied to each row of the table and the rows are those for which the result of the search condition is true are updated.

***value***

Indicates the new value of the column.



| **schema-name**

| The *schema-name* defines the schema in your SQL query. In IMS, the schema  
| name is the PCB name.

| **table-name**

| The *table-name* defines the name of the table in your SQL query.

| **column-name**

| The *column-name* defines the name of the column in your SQL query.

| **Notes**

| **Update rules:**

| Update values must satisfy the following rules. If they do not, or if other  
| errors occur during the execution of the UPDATE statement, no rows are  
| updated and the position of the cursors are not changed.

- | • *Assignment.* Update values are assigned to columns using the assignment  
| rules described in “Language elements” on page 619.
- | • When updating a record in a table at a non-root level, you must specify  
| values for all the foreign key fields of the table to identify the exact  
| record (or segment instance) to update.
- | • Making an UPDATE on a foreign key field is invalid.

| **Number of rows updated:**

| For COBOL, the value of SQLIMSERRD(3) in the SQLIMSCA is the  
| number of rows updated after an UPDATE statement completes execution.  
| For a complete description of the SQLIMSCA, including exceptions to the  
| preceding sentence, see “SQL communication area (SQLIMSCA)” on page  
| 669.

| **Examples**

| **Updating one column in a record**

| The following statement updates the root:

| UPDATE HOSPITAL SET HOSPNAME = 'MISSION CREEK'  
| WHERE HOSPITAL.HOSPCODE = 'H001007'

| **Updating multiple fields in a specified record in a hierarchic path**

| Foreign keys are used to maintain referential integrity by identifying the  
| exact record (or segment instance) to update. The following statement  
| updates a WARD record under a specific HOSPITAL. In this example, the  
| WARD table has the virtual foreign key HOSPITAL\_HOSPCODE. The  
| record will be updated if and only if there is a HOSPCODE in the  
| HOSPITAL table with the value of 'H5140070000H'.

| UPDATE WARD SET WARDNAME = 'EMGY',  
| DOCCOUNT = '2', NURCOUNT = '4'  
| WHERE HOSPITAL\_HOSPCODE = 'H5140070000H'  
| AND WARDNO = '01'

| **Example of an invalid UPDATE query**

| This statement is invalid because it does not use the correct syntax to  
| specify a legal value for the virtual foreign key field  
| (HOSPITAL\_HOSPCODE).

| UPDATE WARD SET WARDNAME = 'EMGY',  
| DOCCOUNT = '2', NURCOUNT = '4'  
| WHERE HOSPITAL.HOSPCODE = 'H5140070000H'  
| AND WARDNO = '01'

### Example of an invalid foreign key field UPDATE query

Making an UPDATE query on a foreign key field is invalid. For example, the following UPDATE query will fail:

```
UPDATE WARD SET WARDNAME = 'EMGY',  
        HOSPITAL_HOSPCODE = 'H5140070000H'  
WHERE WARDNO = '01'
```

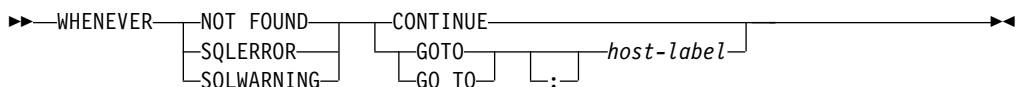
## WHENEVER

The WHENEVER statement specifies the host language statement to be executed when a specified exception condition occurs.

### Invocation

This statement can be embedded only in a COBOL application program. It is not an executable statement.

### Syntax



### Description

The following keyword parameters are defined for the WHENEVER statement:

The **NOT FOUND**, **SQLERROR**, or **SQLWARNING** clause is used to identify the type of exception condition.

#### **NOT FOUND**

Identifies any condition that results in an SQLIMSCODE of +100 (equivalently, an SQLIMSSTATE code of '02000').

#### **SQLERROR**

Identifies any condition that results in a negative SQLIMSCODE.

#### **SQLWARNING**

Identifies any condition that results in a warning condition (SQLIMSWARN0 is W), or that results in a positive SQLIMSCODE other than +100.

The CONTINUE or GO TO clause specifies the next statement to be executed when the identified type of exception condition exists.

#### **CONTINUE**

Specifies the next sequential statement of the source program.

#### **GOTO or GO TO *host-label***

Specifies the statement identified by *host-label*. For *host-label*, substitute a single token, optionally preceded by a colon. The form of the token depends on the host language. In COBOL, for example, it can be *section-name* or an unqualified *paragraph-name*.

### Notes

There are three types of WHENEVER statements:

- WHENEVER NOT FOUND
- WHENEVER SQLERROR

- WHENEVER SQLWARNING

Every executable SQL statement in an application program is within the scope of one implicit or explicit WHENEVER statement of each type. The scope of a WHENEVER statement is related to the listing sequence of the statements in the application program, not their execution sequence.

An SQL statement is within the scope of the last WHENEVER statement of each type that is specified before that SQL statement in the source program. If a WHENEVER statement of some type is not specified before an SQL statement, that SQL statement is within the scope of an implicit WHENEVER statement of that type in which CONTINUE is specified.

### Examples

The following statements can be embedded in a COBOL program.

*Example 1:* Go to the label HANDLER for any statement that produces an error.

```
EXEC SQLIMS WHENEVER SQLERROR GOTO HANDLER END-EXEC.
```

*Example 2:* Continue processing for any statement that produces a warning.

```
EXEC SQLIMS WHENEVER SQLWARNING CONTINUE END-EXEC.
```

*Example 3:* Go to the label ENDDATA for any statement that does not return.

```
EXEC SQLIMS WHENEVER NOT FOUND GO TO ENDDATA END-EXEC.
```

---

## SQL communication area (SQLIMSCA)

An SQLIMSCA is a structure or collection of variables that is updated after each SQL statement executes. An application program that contains executable SQL statements must provide exactly one SQLIMSCA.

In COBOL, the INCLUDE statement can be used to provide the declaration of the SQLIMSCA.

### Description of SQLIMSCA fields

The SQLIMS INCLUDE statement provides SQLIMSCA field.

The names in the following table are those provided by the SQLIMS INCLUDE statement.

*Table 166. Fields of SQLIMSCA*

COBOL name	Data type	Purpose
SQLIMSCAID	CHAR(8)	An “eye catcher” for storage dumps, containing the text 'SQLIMSCA'.
SQLIMSCABC	INTEGER	Contains the length of the SQLIMSCA: 224.

Table 166. Fields of SQLIMSCA (continued)

COBOL name	Data type	Purpose
SQLIMSCODE	INTEGER	Contains the SQL return code. (See note 2 on page 671)
		<b>Code</b> <b>Means</b> <b>0</b> Successful execution (though there might have been warning messages). <b>positive</b> Successful execution, but with a warning condition or other information. <b>negative</b> Error condition.
SQLIMSERRML (See note 1 on page 671)	SMALLINT	Length indicator for SQLIMSERRMC, in the range 0 through 255. 0 means that the value of SQLIMSERRMC is not pertinent.
SQLIMSERRMC (See note 1 on page 671)	VARCHAR(158)	Contains the error message.
SQLIMSERRP	CHAR(8)	Provides a product signature and, in the case of an error, diagnostic information such as the name of the module that detected the error. In all cases, the first three characters are 'DQF' or 'DFS' for IMS.
SQLIMSERRD(1)	INTEGER	Reserved.
SQLIMSERRD(2)	INTEGER	Reserved.
SQLIMSERRD(3)	INTEGER	Contains the number of rows that qualified to be deleted, inserted, or updated after a DELETE, INSERT, or UPDATE statement.
SQLIMSERRD(4)	INTEGER	Reserved.
SQLIMSERRD(5)	INTEGER	Reserved.
SQLIMSERRD(6)	INTEGER	Reserved.
SQLIMSWARN0	CHAR(1)	Contains a blank if no other indicator is set to a warning condition (that is, no other indicator contains a W or Z). Contains a W if at least one other indicator contains a W or Z.
SQLIMSWARN1	CHAR(1)	Contains a W if the value of a string column was truncated when assigned to a host variable.
SQLIMSWARN2	CHAR(1)	Reserved.
SQLIMSWARN3	CHAR(1)	Contains a W if the number of result columns is larger than the number of host variables.
SQLIMSWARN4	CHAR(1)	Contains a W if a prepared UPDATE or DELETE statement does not include a WHERE clause.
SQLIMSWARN5	CHAR(1)	Contains a W if the SQL statement was not executed because it is not a valid SQL statement in IMS.
SQLIMSWARN6	CHAR(1)	Reserved.
SQLIMSWARN7	CHAR(1)	Reserved.
SQLIMSWARN8	CHAR(1)	Reserved.
SQLIMSWARN9	CHAR(1)	Reserved.
SQLIMSWARNA	CHAR(1)	Reserved.
SQLIMSSTATE	CHAR(5)	Contains a return code for the outcome of the most recent execution of an SQL statement (See note 2 on page 671).

**Notes:**

1. In COBOL, SQLIMSERRM includes SQLIMSERRML and SQLIMSERRMC. See the examples for the various host languages in “The included SQLIMSCA.”
2. For a description of the SQLIMSSTATE values, see SQL codes (Messages and Codes).

## The included SQLIMSCA

The description of the SQLIMSCA that is given by INCLUDE SQLIMSCA is shown for COBOL.

### COBOL:

```

01 SQLIMSCA GLOBAL.
   05 SQLIMSCAID PIC X(8).
   05 SQLIMSCABC PIC S9(9) COMP-5.
   05 SQLIMSCODE PIC S9(9) COMP-5.
   05 SQLIMSERRM.
       49 SQLIMSERRML PIC S9(4) COMP-5.
       49 SQLIMSERRMC PIC X(158).
   05 SQLIMSERRP PIC X(8).
   05 SQLIMSERRD PIC S9(9) COMP-5.
   05 SQLIMSWARN.
       10 SQLIMSWARN0 PIC X.
       10 SQLIMSWARN1 PIC X.
       10 SQLIMSWARN2 PIC X.
       10 SQLIMSWARN3 PIC X.
       10 SQLIMSWARN4 PIC X.
       10 SQLIMSWARN5 PIC X.
       10 SQLIMSWARN6 PIC X.
       10 SQLIMSWARN7 PIC X.
   05 SQLIMSEXT.
       10 SQLIMSWARN8 PIC X.
       10 SQLIMSWARN9 PIC X.
       10 SQLIMSWARNA PIC X.
       10 SQLIMSSTATE PIC X(5).

```

---

## SQL descriptor area (SQLIMSDA)

An SQLIMSDA is a collection of variables that is required for execution of the SQLIMS DESCRIBE statement, and can be optionally used by the FETCH statements. An SQLIMSDA can be used in a DESCRIBE statement, modified with the addresses of host variables, and then reused in a FETCH statement.

The meaning of the information in an SQLIMSDA depends on the context in which it is used. For DESCRIBE, IMS sets the fields in the SQLIMSDA to provide information to the application program. For FETCH, the application program sets the fields in the SQLIMSDA to provide IMS with information:

### DESCRIBE *statement-name*

With the exception of SQLIMSN, IMS sets fields of the SQLIMSDA to provide information to an application program about a prepared statement. Each SQLIMSVAR occurrence describes a column of the result table.

### FETCH

The application program sets fields of the SQLIMSDA to provide information about host variables or output buffers in the application program to IMS. Each SQLIMSVAR occurrence describes a host variable or output buffer.

- For FETCH, each SQLIMSVAR occurrence describes a host variable or buffer in the application program that is to be used to contain an output value from a row of the result.

## Description of SQLIMSDA fields

An SQLIMSDA consists of four variables, a header, and an arbitrary number of occurrences of a sequence of variables collectively named SQLIMSVAR.

In DESCRIBE, each occurrence of the SQLIMSVAR describes the column of a table. In FETCH, each occurrence describes a host variable.

### The SQLIMSDA header

The fields in the SQLIMSDA header have different usage depending on whether the SQLIMSDA is being used in a DESCRIBE or FETCH statement.

The following table describes the fields in the SQLIMSDA header.

Table 167. Fields of the SQLIMSDA header

COBOL name	Data type	Usage in DESCRIBE	Usage in FETCH
sqlimsdaid SQLIMSDAID	CHAR(8)	An "eye catcher" for storage dumps, containing the text 'SQLIMSDA '.	SQLIMSDAID is not used.
sqlimsdabc SQLIMSDABC	INTEGER	Length of the SQLIMSDA, equal to $SQLIMSN \times 44 + 16$ .	Length of the SQLIMSDA, greater than or equal to $SQLIMSN \times 44 + 16$ .
sqlimsn SQLIMSN	SMALLINT	The field must be set to a value greater than or equal to zero before the statement is executed. The field indicates the total number of occurrences of SQLIMSVAR. In COBOL, the included SQLIMSDA contains up to 750 occurrences of SQLIMSVAR.	Total number of occurrences of SQLIMSVAR provided in the SQLIMSDA. SQLIMSN must be set to a value greater than or equal to zero. In COBOL, the included SQLIMSDA contains up to 750 occurrences of SQLIMSVAR.
sqlimsd SQLIMSD	SMALLINT	The number of columns described by occurrences of SQLIMSVAR.	The number of host variables described by occurrences of SQLIMSVAR.

### SQLIMSVAR entries

For each column or host variable described by the SQLIMSDA, it is described using the SQLIMSVAR entry.

The fields of this entry contain the base information about the column or host variable such as data type code, length attribute (except for LOBs), column name, host variable address, and indicator variable address.

### Determining how many SQLIMSVAR occurrences are needed:

The number of SQLIMSVAR occurrences needed depends on the statement that the SQLIMSDA was provided for and the data types of the columns or parameters being described.

The included SQLIMSDA provides up to 750 occurrences of SQLIMSVAR. The SQLIMSD is set to the number of columns in the result and represents the number of SQLIMSVAR occurrences needed. If an insufficient number of SQLIMSVAR occurrences were provided, IMS returns a +239 warning in SQLIMSCODE.

SQLIMSD is set to the number of columns in the result.

**Field descriptions of an occurrence of a base SQLIMSVAR:**

The fields of a base SQLIMSVAR have different uses depending on the SQL statement.

The following table describes the contents of the fields of a base SQLIMSVAR.

*Table 168. Fields in an occurrence of a base SQLIMSVAR*

COBOL name	Data type	Usage in DESCRIBE	Usage in FETCH
sqlimstype SQLIMSTYPE	SMALLINT	Indicates the data type of the column and whether it can contain null values. For a description of the type codes, see Table 169.	Indicates the data type of the host variable and whether an indicator variable is provided. For a description of the type codes, see "SQLIMSTYPE and SQLIMSLEN."
sqlimslen SQLIMSLEN	SMALLINT	The length attribute of the column. See "SQLIMSTYPE and SQLIMSLEN" for a description of allowable values.	The length attribute of the host variable. See "SQLIMSTYPE and SQLIMSLEN" for a description of allowable values.
sqlimsdata SQLIMSDATA	pointer	Reserved.	Contains the address of the host variable.
sqlimsind SQLIMSIND	pointer	Reserved	Contains the address of an associated indicator variable, if SQLIMSTYPE is odd. Otherwise, the field is not used.
sqlimsname SQLIMSNAME	VARCHAR(30)	Contains the unqualified name or label of the column, or a string of length zero if the name or label does not exist. If the name is longer than 30 bytes, it is truncated at a byte boundary.	Contains the unqualified name or label of the column, or a string of length zero if the name or label does not exist. If the name is longer than 30 bytes, it is truncated at a byte boundary.

**SQLIMSTYPE and SQLIMSLEN:**

The contents of the SQLIMSTYPE and SQLIMSLEN fields of the SQLIMSDA depends on the SQL statement that is returning the value.

The following table shows the values that can appear in the SQLIMSTYPE and SQLIMSLEN fields of the SQLIMSDA. In DESCRIBE, an even value of SQLIMSTYPE means the column does not allow nulls, and an odd value means the column does allow nulls. In FETCH, an even value of SQLIMSTYPE means no indicator variable is provided, and an odd value means that SQLIMSIND contains the address of an indicator variable.

*Table 169. SQLIMSTYPE and SQLIMSLEN values for DESCRIBE, FETCH, OPEN, and EXECUTE*

SQLIMSTYPE	For DESCRIBE		For FETCH	
	Column or parameter data type	SQLIMSLEN	Host variable data type	SQLIMSLEN
384/385	date	10 <sup>1</sup>	fixed-length character string representation of a date	length attribute of the host variable
388/389	time	8 <sup>2</sup>	fixed-length character string representation of a time	length attribute of the host variable

Table 169. SQLIMSTYPE and SQLIMSLEN values for DESCRIBE, FETCH, OPEN, and EXECUTE (continued)

SQLIMSTYPE	For DESCRIBE		For FETCH	
	Column or parameter data type	SQLIMSLEN	Host variable data type	SQLIMSLEN
392/393	timestamp	26	fixed-length character string representation of a timestamp	length attribute of the host variable
452/453	fixed-length character string	length attribute of the column	fixed-length character string	length attribute of the host variable
480/481	floating point	4 for single precision, 8 for double precision	floating point	4 for single precision, 8 for double precision
484/485	packed decimal	precision in byte 1; scale in byte 2	packed decimal	precision in byte 1; scale in byte 2
492/493	big integer <sup>4</sup>	8	big integer	8
496/497	large integer	4	large integer	4
500/501	small integer	2	small integer	2
504/505	N/A	N/A	DISPLAY SIGN LEADING SEPARATE, NATIONAL SIGN LEADING SEPARATE	precision in byte 1; scale in byte 2
912/913	fixed-length binary string	length attribute of the column	fixed-length binary string	length attribute of the host variable

**Note:**

1. SQLIMSLEN might be different if a date installation exit is specified.
2. SQLIMSLEN might be different if a time installation exit is specified.
3. Field SQLIMSLONGLEN in the extended SQLIMSVAR contains the length attribute of the column.
4. BIGINT is supported by other IMS platforms.

## The included SQLIMSDA

Only COBOL is supported for the SQLIMSDA that is given by INCLUDE SQLIMSDA.

### COBOL (IBM COBOL only):

```

01 SQLIMSDA GLOBAL.
  02 SQLIMSDAID      PIC X(8).
  02 SQLIMSDABC      PIC S9(9) COMP-5.
  02 SQLIMSN         PIC S9(4) COMP-5.
  02 SQLIMSD         PIC S9(4) COMP-5.
  02 SQLIMSVAR OCCURS 0 TO 750 TIMES DEPENDING ON SQLIMSN.
    03 SQLIMSVAR1.
      04 SQLIMSTYPE  PIC S9(4) COMP-5.
      04 SQLIMSLEN   PIC S9(4) COMP-5.
      04 FILLER REDEFINES SQLIMSLEN.
      05 SQLIMSPRECISION PIC X.
      05 SQLIMSSCALE   PIC X.
      04 SQLIMSDATA  POINTER.
      04 SQLIMSIND   POINTER.
      04 SQLIMSNAME.
        49 SQLIMSNAMEL PIC S9(4) COMP-5.
        49 SQLIMSNAMEC PIC X(30).
    03 SQLIMSVAR2 REDEFINES SQLIMSVAR1.

```







---

## Chapter 9. XML support reference

These topics contain reference information for the IMS support for storing and retrieving XML data.

---

### SQL extensions for XML storage and retrieval

The Java class libraries for IMS have two SQL99 extensions for user-defined functions (UDFs): `retrieveXML` and `storeXML`.

**Recommendation:** Because the IMS Universal JDBC driver provides improved support for XML, use the IMS Universal JDBC driver to store XML data in IMS databases or retrieve and compose XML documents from existing IMS databases.

These UDFs are used during JDBC calls to store and retrieve XML from IMS databases. This interface is independent of the physical storage of the data.

Retrieval of XML from IMS databases can also be achieved when UDFs are combined with XQuery functions and operators.

**Related reference:**

“XQuery support in the IMS classic JDBC driver” on page 680

#### **retrieveXML UDF**

The `retrieveXML` UDF creates an XML document from an IMS database and returns an object that implements the `java.sql.Clob` interface.

**Recommendation:** Because the IMS Universal JDBC driver provides improved support for XML, use the IMS Universal JDBC driver to store XML data in IMS databases or retrieve and compose XML documents from existing IMS databases.

It does not matter to the application whether the data is decomposed into standard IMS segments or the data is in intact XML documents in the IMS database.

The `Clob` JDBC type stores a Character Large Object as a column value in a row of the result set. The `getClob` method retrieves the XML document from the result set. The following figure shows the relationship between the `retrieveXML` UDF and the `getClob` method.

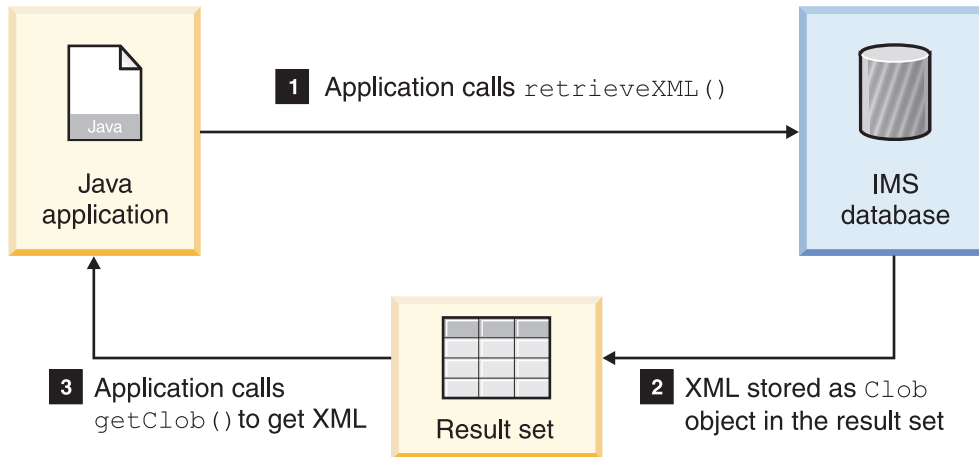


Figure 37. Creating XML using the `retrieveXML` UDF and the `getClob` method

To create an XML document, use a `retrieveXML` UDF in the `SELECT` statement of your JDBC call. Pass in the name of the segment that will be the root element of the XML document (for example, `retrieveXML(Model)`). The dependent segments of the segment that you pass in will be in the generated XML document if they match the criteria listed in the `WHERE` clause.

The segment that you specify to be the root element of the XML document does not have to be the root segment of the IMS record. The dependent segments are mapped to the XML document based on the generated XML schema.

Within a single application program, you can issue `SELECT` calls that contain `retrieveXML` UDFs against multiple PCBs in an application's PSB. You can also issue multiple `retrieveXML` UDFs that pass in various segments along the requested hierarchical path from a single `SELECT` call. From a single `SELECT` call, you can also retrieve other types of data in addition to the XML document (for example, `SELECT retrieveXML(Model), Dealer.DealerNo`).

The following example creates an XML document that has the root element of `Model`:

```
SELECT retrieveXML(Model)
FROM DealershipDB.Model
WHERE Model.CarYear = '1969'
```

The XML document that is created has the root element of the `Model` segment that has the `CarYear` field of 1969.

The XML document that is retrieved is stored in the result set. For each row in the result set, the UDF creates an implementation of the JDBC `java.sql.Clob` interface, and places it in the corresponding result set column. This `Clob` object encapsulates the XML document created from the database.

The storage requirements for the XML document `Clob` objects in a result set depend on whether the result set is forward-only or scroll-insensitive.

If the `Clob` object is returned to a forward-only `ResultSet` object, data is retrieved from the database and composed into XML only when the application requests the data. For example, if the application invokes the `getAsciiStream` or `getCharacterStream` method, the application receives a `Stream` object. As the application reads the XML stream, the segments are retrieved from the database

and composed into XML. At the end of the stream, the entire XML document has been returned to the application having never been fully materialized in the Clob object.

If the Clob object is returned to a scroll-insensitive ResultSet object, the whole document is materialized in the Clob. This option requires more memory than forward-only result sets, especially for large XML documents and result sets with a lot of rows.

To retrieve the XML document from the result set, use the getClob method.

The following example retrieves an XML document, encapsulated by the Clob object, from the result set:

```
Clob xmlDoc = resultSet.getClob(1);
```

Using the getClob interface, you can, for example, retrieve all or part of document content as a String object, or request a Stream or Reader object for the document. With the Stream or Reader object, you can send the document to an output queue or as a response to an HTTP or SOAP request, or save it in a local HFS file. You can also selectively retrieve elements using a selected subset of XPath expressions, or transform the document using XSLT.

## storeXML UDF

The storeXML UDF inserts an XML document into an IMS database at the position in the database that the WHERE clause indicates.

**Recommendation:** Because the IMS Universal JDBC driver provides improved support for XML, use the IMS Universal JDBC driver to store XML data in IMS databases or retrieve and compose XML documents from existing IMS databases.

IMS, not the application, uses the XML schema and the Java metadata class to determine the physical storage of the data into the database. It does not matter to the application whether the XML is stored intact or decomposed into standard IMS segments.

An XML document must be valid before it can be stored into a database. The storeXML UDF validates the XML document against the XML schema before storing it. If you know that the XML document is valid and you do not want IMS to revalidate it, use the storeXML(false) UDF.

To store an XML document, use the storeXML UDF in the INSERT INTO clause of a JDBC prepared statement. Within a single application program, you can issue INSERT calls that contain storeXML UDFs against multiple PCBs in an application's PSB.

The SQL query must have the following syntax:

```
INSERT INTO PCB.Segment (storeXML())  
VALUES ( ? )  
WHERE Segment.Field = value
```

Because an XML document is not a valid argument in the VALUES clause of the INSERT statement, you must use a prepared statement.

The following example stores the XML document named myDoc.xml from the file system into an IMS database using the Dealership PCB. A new Model segment,

which is the root of the XML document, is inserted under the Dealer segment that has the number A123. The rest of the XML document is stored as dependent segments of Model as specified by the XML Schema.

```
InputStreamReader myXMLDoc =
    new InputStreamReader(new FileInputStream("myDoc.xml"));

String query = "INSERT INTO Dealership.Model (storeXML())" +
    " VALUES ( ? )" +
    " WHERE Dealer.DealerNumber = 'A123' ";

PreparedStatement pstmt = conn.prepareStatement(query);

pstmt.setCharacterStream(1, myXMLDoc, -1);
```

---

## XQuery support in the IMS classic JDBC driver

XQuery is a flexible language that is designed for querying and processing XML documents. IMS delivers an XQuery engine that is capable of evaluating XQuery expressions against an IMS database with established IMS to XML mappings.

**Recommendation:** Because the IMS Universal JDBC driver provides improved support for XML, use the IMS Universal JDBC driver to store XML data in IMS databases or retrieve and compose XML documents from existing IMS databases.

Implement XQuery with the `retrieveXML()` and `storeXML()` SQL99 extensions in the IMS classic JDBC driver. These extensions use an XML schema and the Java metadata class to store and retrieve data from IMS. The XML schema and the Java metadata class are generated using the IMS Enterprise Suite DLIModel utility plug-in. XQuery support for IMS shares with the IMS classic JDBC driver the functionality to be implemented in any development environment that is supported by IMS, including WebSphere Application Server for distributed platforms, WebSphere Application Server for z/OS, Db2 for z/OS, and CICS.

**Related reference:**

“SQL extensions for XML storage and retrieval” on page 677

“GN command” on page 169

“GN/GHN call” on page 11

## XQuery function and operation extensions for IMS

You can optimize your search results with XQuery support with the JDBC driver for IMS by further refining your results with IMS-specific function and operation extensions.

**Recommendation:** Because the IMS Universal JDBC driver provides improved support for XML, use the IMS Universal JDBC driver to store XML data in IMS databases or retrieve and compose XML documents from existing IMS databases.

### ims:particle

`ims:particle()` is a helper function, used to identify specific nodes in the XML representation of an IMS database.

#### Syntax

```
ims:particle($path)
```

#### Options

##### **\$path**

A notation that navigates along paths in an XML document.

## Example

The following example demonstrates how to create a particle reference for the book node in an XML representation of an IMS database.

```
ims:particle('/bib/book')
```

## ims:gn

`ims:gn()` is used to represent a sequence of Get Next DL/I calls over an IMS database. At each iteration over the `gn` sequence the position in the database is defined by the `SSAList` and `ims:particle` function

### Syntax

```
ims:gn($particle, $predicate)
```

### Options

#### **\$particle**

Specifies the target position of the GN call within the constructed IMS XML document. This position is specified using the `ims:particle` function. For more details, see “`ims:particle`” on page 680.

#### **\$predicate**

Specifies an expression that contains a comparator such as `equals`, `greater-than`, `greater-than-or-equals`, `less-than`, `less-than-or-equals`, and `not-equals`. Alternately, the expression may contain a binary operator. One expression can contain a combination of both comparators and binary operators. In all cases, the expression takes a comparison value.

### Usage

The `ims:gn()` call requires two sets of arguments: `$particle` and `$predicate`. These arguments build the same Segment Search Argument (SSA) that is used in the evaluation of DL/I GN call.

The first argument, `$particle`, must be an `ims:particle()` call that specifies an XPath of type string, denoted with single quotation marks. This XPath logically crawls down the hierarchy of the database to a PCB.

The second argument, `$predicate`, is an optional expression that filters the results returned from `$particle`. The expression consists of using comparators or boolean operators.

For more information about SSAs and their restrictions, see *IMS Version 13 Database Administration*.

## Example

The following example returns all books stored in the PCB mapped to the node 'book'.

```
ims:gn(ims:particle('/bib/book'))
```

## ims:eq

`ims:eq` defines an SSA using the EQ (equals) qualification.

### Syntax

```
ims:eq($particle,$value)
```

### Options

#### **\$particle**

Specifies the virtual position within a constructed XML Schema. This position is specified using the `ims:particle` function. For more details, see “`ims:particle`” on page 680.

**\$value**

Specifies the comparison value. The comparison value can be of any type shared with constructor functions. For a list of these types, see “XQuery functions and operators supported by IMS” on page 686.

**Example**

The following example returns result sets where the publisher element is equal to 'Addison-Wesley '.

```
ims:eq(ims:particle('/bib/book/publisher'), 'Addison-Wesley ')
```

**ims:gt**

ims:gt defines an SSA using the GT (greater than) qualification.

**Syntax**

```
ims:gt(particle,$value)
```

**Options****\$particle**

Specifies the virtual position within a constructed XML Schema. This position is specified using the `ims:particle` function. For more details, see “ims:particle” on page 680.

**\$value**

Specifies the comparison value. The comparison value can be of any type shared with constructor functions. For a list of these types, see “XQuery functions and operators supported by IMS” on page 686.

**Example**

The following example returns a result set for books whose attribute of year is greater than 1991.

```
ims:gt(ims:particle('/bib/book/@year'), 1991)
```

**ims:ge**

ims:ge defines an SSA using the GE (greater than or equals) qualification.

**Syntax**

```
ims:ge($particle,$value)
```

**Options****\$particle**

Specifies the virtual position within a constructed XML Schema. This position is specified using the `ims:particle` function. For more details, see “ims:particle” on page 680.

**\$value**

Specifies the comparison value. The comparison value can be of any type shared with constructor functions. For a list of these types, see “XQuery functions and operators supported by IMS” on page 686.

**Example**

The following example returns results where the attribute of year for the element book is greater than or equal to 1991.

```
ims:ge(ims:particle('/bib/book/@year'), 1991)
```

**ims:lt**

ims:lt defines an SSA using the LT (less than) qualification.

**Syntax**



```
ims:lt($particle,$value)
```

### Options

#### **\$particle**

Specifies the virtual position within a constructed XML Schema. This position is specified using the `ims:particle` function. For more details, see “`ims:particle`” on page 680.

#### **\$value**

Specifies the comparison value. The comparison value can be of any type shared with constructor functions. For a list of these types, see “XQuery functions and operators supported by IMS” on page 686.

### Example

The following example returns results where the attribute of year of publication is less than 1991.

```
ims:lt(ims:particle('/bib/book/@year'), 1991)
```

### **ims:le**

`ims:le` defines an SSA using the LE (less than or equals) qualification.

### Syntax

```
ims:le($particle,$value)
```

### Options

#### **\$particle**

Specifies the virtual position within a constructed XML Schema. This position is specified using the `ims:particle` function. For more details, see “`ims:particle`” on page 680.

#### **\$value**

Specifies the comparison value. The comparison value can be of any type shared with constructor functions. For a list of these types, see “XQuery functions and operators supported by IMS” on page 686.

### Example

The following example returns results where the attribute of year of publication is less than or equal to 1991.

```
ims:le(ims:particle('/bib/book/@year'), 1991)
```

### **ims:ne**

`ims:ne` defines an SSA using the NE (not equals) qualification.

### Syntax

```
ims:ne($particle,$value)
```

### Options

#### **\$particle**

Specifies the virtual position within a constructed XML Schema. This position is specified using the `ims:particle` function. For more details, see “`ims:particle`” on page 680.

#### **\$value**

Specifies the comparison value. The comparison value can be of any type shared with constructor functions. For a list of these types, see “XQuery functions and operators supported by IMS” on page 686.

### Example

The following example returns results where the element of publisher is not Addison-Wesley.

```
ims:ne(ims:particle('/bib/book/publisher'), 'Addison-Wesley ')
```

## **ims:and**

`ims:and` joins SSA qualifications with an AND statement.

### **Format**

```
ims:and($predicate1,$predicate2)
```

### **Options**

#### **\$predicate1**

Specifies the first of at least two expressions that contain a comparator such as equals, greater-than, greater-than-or-equals, less-than, less-than-or-equals, and not-equals. Alternately, this expression can contain a binary operator. The first expression can contain a combination of both comparators and binary operators. In all cases, this expression takes a comparison value, which can be another expression.

#### **\$predicate2**

Specifies the second of at least two expressions that contain a comparator such as equals, greater-than, greater-than-or-equals, less-than, less-than-or-equals, and not-equals. Alternately, this expression can contain a binary operator. The second expression can contain a combination of both comparators and binary operators. In all cases, this expression takes a comparison value.

### **Usage**

An `ims:and()` call allows you to combine predicates to create an arbitrarily complex IMS predicate only if all predicates are true. You can use two or more predicates. Each predicate follows the rules for predicate statements as stated in “`ims:gn`” on page 681

### **Example**

The following example returns results containing only books published by Addison-Wesley after 1991.

```
ims:and(
    ims:eq(ims:particle('/bib/book/publisher'), 'Addison-Wesley '),
    ims:gt(ims:particle('/bib/book/@year'), 1991)
)
```

## **ims:or**

`ims:or` joins SSA qualifications with an OR statement.

### **Format**

```
ims:or($predicate1,$predicate2)
```

### **Options**

#### **\$predicate1**

Specifies the first of at least two expressions that contain a comparator such as equals, greater-than, greater-than-or-equals, less-than, less-than-or-equals, and not-equals. Alternately, this expression can contain a binary operator. The first expression can contain a combination of both comparators and binary operators. In all cases, this expression takes a comparison value, which can be another expression.

### **\$predicate2**

Specifies the first of at least two expressions that contain a comparator such as equals, greater-than, greater-than-or-equals, less-than, less-than-or-equals, and not-equals. Alternately, this expression can contain a binary operator. The first expression can contain a combination of both comparators and binary operators. In all cases, this expression takes a comparison value, which can be another expression.

### **Usage**

An `ims:or()` call allows you to combine predicates to create an arbitrarily complex IMS predicate only if one of the predicates is true. You can use two or more predicates. Each predicate follows the rules for predicate statements as stated in “`ims:gn`” on page 681

### **Example**

The following example returns results containing only books whose attributes contain a field year which is greater than 1991 or less than 1951.

```
ims:or(  
    ims:gt(ims:particle('/bib/book/@year'), 1991),  
    ims:le(ims:particle('/bib/book/@year'), 1951)  
)
```

## **Standard XQuery features in IMS**

IMS supports the following standard XQuery features.

**Recommendation:** Because the IMS Universal JDBC driver provides improved support for XML, use the IMS Universal JDBC driver to store XML data in IMS databases or retrieve and compose XML documents from existing IMS databases.

### **IMS support for XQuery primary expressions**

- Literal
- Variable reference
- Parenthesized expressions
- Context item expression
- Function calls
- XPath or XQuery comments in IMS

### **IMS support for XQuery path expressions**

- Axes (Except the optional namespace: axis)
- Node tests
- Predicates

### **IMS support for XQuery sequence expressions**

- ,
- to
- Filter expressions
- union

### **IMS support for XQuery arithmetic expressions**

- +
- -

- \*
- div
- mod

### **IMS support for XQuery comparison expressions**

- Value comparisons
  - lt
  - gt
  - eq
  - le
  - ge
  - ne
- General comparisons
  - <
  - >
  - =
  - <=
  - >=
  - !=
- Node comparisons
  - ==
  - <<
  - >>

### **IMS support for XQuery logical expressions**

- or
- and

### **IMS support for XQuery conditional expressions**

- if\_then\_else

## **XQuery functions and operators supported by IMS**

IMS supports several XQuery 1.0 and XPath 2.0 functions and operators.

**Recommendation:** Because the IMS Universal JDBC driver provides improved support for XML, use the IMS Universal JDBC driver to store XML data in IMS databases or retrieve and compose XML documents from existing IMS databases.

### **XQuery Constructor functions**

- xs:string
- xs:boolean
- xs:decimal
- xs:float
- xs:double
- xs:integer
- xs:int
- op:numeric-add

- op:numeric-subtract
- op:numeric-multiply
- op:numeric-divide
- op:numeric-integer-divide
- op:numeric-mod
- op:numeric-unary-plus
- op:numeric-unary-minus

### **XQuery numeric functions and operators**

- Numeric comparisons:
  - op:numeric-equal
  - op:numeric-less-than
  - op:numeric-greater-than
- Numeric functions:
  - fn:abs
  - fn:ceiling
  - fn:floor
  - fn:round

### **XQuery string functions**

- Functions to assemble and disassemble strings:
  - fn:codepoints-to-string
  - fn:string-to-codepoints
- Equality and comparison of strings:
  - fn:compare
- Functions on string values:
  - fn:concat
  - fn:string-join
  - fn:substring
  - fn:string-length
  - fn:normalize-space
  - fn:translate
- Functions based on substring matching:
  - fn:contains
  - fn:starts-with
  - fn:ends-with
  - fn:substring-before
  - fn:substring-after

### **Boolean functions and operators**

- Boolean constants:
  - fn:false
  - fn:true
- Boolean operators:
  - op:boolean-equal
  - op:boolean-less-than

- op:boolean-greater-than
- Boolean functions:
  - fn:not

## Sequence functions and operators

- General:
  - fn:boolean
  - fn:empty
  - fn:distinct-values
  - fn:subsequence
- Union-intersection-except:
  - op:union
- Aggregate:
  - fn:count
  - fn:max
  - fn:min
  - fn:sum
- Sequence generation:
  - op:to
  - fn:sum
  - fn:doc

## Accessor functions

- fn:node-name
- fn:nilled
- fn:string
- fn:data
- fn:base-uri
- fn:document-uri

## Context functions

- fn:position
- fn:last
- fn:static-base-uri

## Special formal semantics functions

- fs:convert-operand
- fs:convert-simple-operand
- fs:distinct-doc-order
- fs:distinct-doc-order-or-atomic-sequence
- fs:item-sequence-to-node-sequence
- fs:item-sequence-to-untypedAtomic
- fs:item-sequence-to-untypedAtomic-PI
- fs:item-sequence-to-untypedAtomic-text
- fs:item-sequence-to-untypedAtomic-comment
- fs:apply-ordering-mode

- fs:to

### **Error functions**

- fn:error

### **URIs**

- fn:resolve-uri





---

## Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. 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. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

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

*IBM Director of Licensing  
IBM Corporation  
North Castle Drive, MD-NC119  
Armonk, NY 10504-1785  
US*

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

*Intellectual Property Licensing  
Legal and Intellectual Property Law  
IBM Japan, Ltd.  
19-21, Nihonbashi-Hakozakicho, Chuo-ku  
Tokyo 103-8510, Japan*

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those

websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

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

*IBM Director of Licensing  
IBM Corporation  
North Castle Drive, MD-NC119  
Armonk, NY 10504-1785  
US*

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

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

#### COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample

programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work must include a copyright notice as follows:

© (your company name) (year).

Portions of this code are derived from IBM Corp. Sample Programs.

© Copyright IBM Corp. \_enter the year or years\_.

---

## Programming interface information

This information 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 this software product. 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 in order to run with new product releases or versions, or as a result of service. Product-sensitive Programming Interface and Associated Guidance Information is identified where it occurs, either by an introductory statement to a section or topic, or by a Product-sensitive programming interface label. IBM requires that the preceding statement, and any statement in this information that refers to the preceding statement, be included in any whole or partial copy made of the information described by such a statement.

---

## Trademarks

IBM, the IBM logo, and [ibm.com](http://www.ibm.com)<sup>®</sup> are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at <http://www.ibm.com/legal/copytrade.shtml>.

Adobe, the Adobe logo, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

UNIX is a registered trademark of The Open Group in the United States and other countries.

---

## Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

### Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

### Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

### Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

### Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

---

## IBM Online Privacy Statement

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

This Software Offering does not use cookies or other technologies to collect personally identifiable information.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, See IBM's Privacy Policy at <http://www.ibm.com/privacy> and IBM's Online Privacy Statement at <http://www.ibm.com/privacy/details> the section entitled "Cookies, Web Beacons and Other Technologies" and the "IBM Software Products and Software-as-a-Service Privacy Statement" at <http://www.ibm.com/software/info/product-privacy>.



---

## Bibliography

This bibliography lists all of the publications in the IMS Version 13 library, supplemental publications, publication collections, and accessibility titles cited in the IMS Version 13 library.

<b>Title</b>	<b>Acronym</b>	<b>Order number</b>
<i>IMS Version 13 Application Programming</i>	APG	SC19-3646
<i>IMS Version 13 Application Programming APIs</i>	APR	SC19-3647
<i>IMS Version 13 Commands, Volume 1: IMS Commands A-M</i>	CR1	SC19-3648
<i>IMS Version 13 Commands, Volume 2: IMS Commands N-V</i>	CR2	SC19-3649
<i>IMS Version 13 Commands, Volume 3: IMS Component and z/OS Commands</i>	CR3	SC19-3650
<i>IMS Version 13 Communications and Connections</i>	CCG	SC19-3651
<i>IMS Version 13 Database Administration</i>	DAG	SC19-3652
<i>IMS Version 13 Database Utilities</i>	DUR	SC19-3653
<i>IMS Version 13 Diagnosis</i>	DGR	GC19-3654
<i>IMS Version 13 Exit Routines</i>	ERR	SC19-3655
<i>IMS Version 13 Installation</i>	INS	GC19-3656
<i>IMS Version 13 Licensed Program Specifications</i>	LPS	GC19-3663
<i>IMS Version 13 Messages and Codes, Volume 1: DFS Messages</i>	MC1	GC19-4240
<i>IMS Version 13 Messages and Codes, Volume 2: Non-DFS Messages</i>	MC2	GC19-4241
<i>IMS Version 13 Messages and Codes, Volume 3: IMS Abend Codes</i>	MC3	GC19-4242
<i>IMS Version 13 Messages and Codes, Volume 4: IMS Component Codes</i>	MC4	GC19-4243
<i>IMS Version 13 Operations and Automation</i>	OAG	SC19-3657
<i>IMS Version 13 Release Planning</i>	RPG	GC19-3658
<i>IMS Version 13 System Administration</i>	SAG	SC19-3659
<i>IMS Version 13 System Definition</i>	SDG	GC19-3660
<i>IMS Version 13 System Programming APIs</i>	SPR	SC19-3661
<i>IMS Version 13 System Utilities</i>	SUR	SC19-3662

### Supplementary publications

<b>Title</b>	<b>Order number</b>
<i>Program Directory for Information Management System Transaction and Database Servers V13.0</i>	GI10-8914
<i>Program Directory for Information Management System Transaction and Database Servers V13.0 Database Value Unit Edition V13R1</i>	GI10-8966
<i>Program Directory for Information Management System Transaction and Database Servers V13.0 Transaction Manager Value Unit Edition V13R1</i>	GI10-9001
<i>IRLM Messages and Codes</i>	GC19-2666

## Publication collections

Title	Format	Order number
IMS Version 13 Product Kit	CD	SK5T-8864

## Accessibility titles cited in the IMS Version 13 library

Title	Order number
<i>z/OS TSO/E Primer</i>	SA22-7787
<i>z/OS TSO/E User's Guide</i>	SA22-7794
<i>z/OS ISPF User's Guide Volume 1</i>	SC34-4822



# Index

## Special characters

- : (colon)
  - preceding a host variable 629
- ! (exclamation mark) as not sign 631
- !token
  - IMSQUERY function 379
  - STORAGE command 377
- ? (question mark) 644
- /DISPLAY command 546
- /DISPLAY POOL command 439
- /FORMAT command 505, 546
- /MODIFY COMMIT command 505
- /MODIFY PREPARE command 505
- /RDISPLAY command 547
- /RESET command 465
- /SET command 482
- . (period) usage
  - null or void placeholder 369
  - parsing, transparent additions 369
  - REXX 366
- \$\$IMSDIR
  - effect on performance 439
- \*mapname 373, 374
- &DPN= operand (DIV statement), specifying 467

## Numerics

- 3180
  - in partitioned format mode
    - clearing the display 545
    - paging 545
    - restrictions 545
    - scrolling 545
  - screen formatting 433
- 3270 Information Display System
  - compatibility with 5550 433
  - copy function
    - bit 4 of SCA, byte 1 452
    - description 537
  - default literal input message fields 493
  - defining IMS TM password 495
  - defining system message field 522
  - entering and exiting formatted mode 482
  - increasing performance 439
  - master terminal format
    - display area 547
    - literals defined for PF keys 547
  - multiple physical page input 501
  - PA (program access) key, control functions 537
  - printed page format control 522
  - screen formatting 429
  - selector pen
    - for control functions 537
- 3270 operator identification card reader
  - application program device-dependent information 446

- 3270 operator identification card reader
  - (continued)
  - defining IMS TM password 495
  - effect on input fields 446
  - system message field 522
- 3270P Printer
  - printed page format control 523
- 3270P Printers
  - MFS controlling format 525
- 3275/3277 Display Station
  - physical paging 508
  - using default formats with 426
- 3276 Control Unit/Display Station
  - physical paging 508
  - using default formats with 426
- 3278 Display Station
  - compatibility with 5550 433
  - physical paging 508
  - using default formats with 426
- 3279 Display Station, default formats 426
- 3290 Display Panel
  - in partitioned format mode 520
  - in standard format mode 431
  - screen formatting 431
- 3770 Data Communication System
  - entering and exiting formatted mode 482
  - printed page format control 522
- 3790 Communication System
  - operating with MFS
    - FTABs 496
    - input modes 495
- 5550 Family (as 3270) 514
  - compatibility with other devices 433
  - using DBCS fields 514
  - using DBCS/EBCDIC fields 514
- 8-blanks (null) 42

## A

- abend statement 236
- abend, avoiding an 165
- ABNUOWRM reply message 326
  - format 326
- ACCEPT command
  - ACCEPT command
    - description 165
    - example 165
    - format 165
    - options 165
    - system service command 162
    - usage 165
  - accessibility
    - features x
    - keyboard shortcuts x
- ACCRDB command 283
  - format 283
- ACCRDBRM reply message 327
  - format 327
- ACCSEC command 285
  - ACCSEC command (*continued*)
    - format 285
- ACCSECRD reply object 329
- ACTVPID= operand (DPAGE statement)
  - cursor positioning (3290 only) 520
  - specifying 476
  - use 543
- adding
  - a segment sequentially 190
  - segments to a database 185
- addressing environments 361, 363
- AGNPRMRM reply message 330
  - format 330
- AIB (application interface block)
  - AIB identifier
    - in RCMD call 66
  - AIB identifier (AIBID) in GSCD call 43
  - AIB identifier (AIBID)
    - in APSB call 37
    - in CHKP (basic) call 38
    - in CHKP (symbolic) call 39
    - in DPSB call 41
    - in GMSG call 42
    - in ICMD call 45
    - in INIT call 46
    - in INQY call 54
    - in LOG call 63
    - in ROLB call 67
    - in ROLS call 68
    - in SETS/SETU call 70
    - in SNAP call 71
    - in STAT call 74
    - in SYNC call 76
  - AIB identifier (AIBID) in XRST call 77
  - AIBLEN (DFS AIB allocated length) in GSCD call 43
  - AIBLEN (DFS AIB allocated length)
    - in APSB call 37
    - in CHKP (basic) call 38
    - in CHKP (symbolic) call 39
    - in DPSB call 41
    - in GMSG call 42
    - in ICMD call 45
    - in INIT call 46
    - in INQY 54
    - in LOG call 63
    - in RCMD call 66
    - in ROLB call 67
    - in ROLS call 68
    - in SETS/SETU call 70
    - in SNAP call 71
    - in STAT call 74
    - in SYNC call 76
  - AIBLEN (DFS AIB allocated length) in XRST call 77
  - AIBOALEN ((maximum output area length) in ROLS call 68

- AIB (application interface block)
    - (continued)
    - AIBOALEN (maximum output area length)
      - in LOG call 63
    - AIBOALEN (maximum output area length) in GSCD call 43
    - AIBOALEN (maximum output area length)
      - in CHKP (symbolic) call 39
      - in GMSG call 42
      - in ICMD call 45
      - in INIT call 46
      - in INQY call 54
      - in RCMD call 66
      - in SETS/SETU call 70
      - in SNAP call 71
      - in STAT call 74
    - AIBOALEN (maximum output area length) in ROLB call 67
    - AIBOALEN (maximum output area length) in XRST call 77
    - AIBOAUSE (used output area length)
      - in GMSG call 42
      - in ICMD call 45
      - in RCMD call 66
    - AIBRSNM1 (resource name)
      - in APSB call 37
      - in CHKP (symbolic) call 39
      - in DPSB call 41
      - in GMSG call 42
      - in INIT call 46
      - in INQY call 54
      - in LOG call 63
      - in ROLB call 67
      - in ROLS call 68
      - in SETS/SETU call 70
      - in SNAP call 71
      - in STAT call 74
      - in SYNC call 76
    - AIBRSNM1 (resource name) in CHKP (basic) call 38
    - AIBRSNM1 (resource name) in XRST call 77
    - AIBRSNM2
      - in APSB call 37
      - in CHKP (basic) call 38
    - AIBSFUNC (subfunction code)
      - in DPSB call 41
      - in GMSG call 42
      - in INQY call 54
    - DL/I calls, system service
      - ROLB 67
    - interface, REXX 364
    - ROLB (Roll Back) call
      - format 67
      - parameters 67
      - subfunction, setting 375
    - AIBOALEN parameter 351
    - AIBRSNM1 (resource name)
      - in GSCD call 43
    - AIBRSNM1 parameter 351
    - AIBRSNM2 parameter 351
    - AIBSFUNC parameter 352
    - aibStream data structure
      - format 352
      - overview 352
  - Allocate PSB (APSB) call 37
    - format 37
    - parameters 37
    - usage 37
  - allocate PSB call 125
  - allowed commands, EXEC DLI 162
  - AND
    - truth table 634
  - AO (automated operator) application
    - after status codes
      - GCMD call 97
    - GCMD call
      - status codes 97
    - GMSG call 42, 129
    - ICMD call 45, 132
    - RCMD call 66, 151
  - AOI token, usage 42
  - API
    - otma\_alloc 559
    - otma\_close 568
    - otma\_create 555
    - otma\_free 568
    - otma\_open 556
    - otma\_openx 558
    - otma\_receive\_async 566
    - otma\_send\_async 564
    - otma\_send\_receive 561
    - otma\_send\_receivex 563
  - APPC environment 361
  - application program
    - deadlock occurrence, in 46
      - SQLIMSCA 669
      - SQLIMSDA 671
  - APSB (Allocate PSB) call 37
    - format 37
    - parameters 37
    - usage 37
  - APSB call 125
    - description 125
    - format 125
    - parameters 125
    - restrictions 125
    - summary 81
    - usage 125
  - area
    - in CHKP (symbolic) call 39
  - area length
    - in CHKP (symbolic) call 39
    - in XRST call 77
  - assignment
    - compatibility rules 624
    - retrieval rules 626
    - storage rules 626
    - strings, basic rules for 625
  - ATTACH FM header 467, 529
  - ATTACH manager
    - blocking algorithms 529
    - deblocking algorithms 503
  - ATTR= operand (MFLD statement)
    - example 456
    - use 511
  - attribute data
    - input message fields
      - description 494
    - output device fields
      - description 511
    - for cursor positioning 520
  - attribute simulation
    - description 511
    - restrictions 454
  - AUTH call 83
    - description 83
    - format 83
    - I/O area format 83
    - parameters 83
    - restrictions 83
    - summary 81
    - usage 83
  - authorization call 83
  - avoiding an abend 165
- ## B
- backing out
    - changes dynamically 200
  - backout point
    - setting 205
    - unconditionally setting 206
  - basic checkpoint
    - description 165
  - Basic Checkpoint (CHKP Basic)
    - description 38
    - format 38
    - parameters 38
    - usage 38
  - basic checkpoint call 126
  - Basic CHKP call 126
    - description 126
    - format 126
    - parameters 126
    - restrictions 126
    - summary 81
    - usage 126
  - Basic edit
    - IMS TM 505
  - basic operations in SQL 624
  - basic predicate 631
  - batch programs
    - deadlock occurrence, in 46
  - BETWEEN predicate 632
  - BIGINT
    - data type 622
  - binding
    - SQL statements 617
  - BKO execution parameter 200
  - block error message format 546
  - BSAM (basic sequential access method)
    - using with Spool API 112
  - built-in data type 620
- ## C
- call functions, DL/I 248
  - CALL statement 237
    - CALL DATA 242
    - CALL DATA statement internal field 242
    - CALL FUNCTION 238
    - SETO, DFSDDLTO
      - description 238
  - CALL statements
    - CALL FUNCTION statement 241
    - FEEDBACK DATA statement 244

- CALL statements (*continued*)
  - OPTION DATA statement 245
- call summary, transaction management 81
- Callable Interface (C/I) 553
  - otma\_alloc API 559
  - otma\_close API 568
  - otma\_create API 555
  - otma\_free API 568
  - otma\_open API 556
  - otma\_openx API 558
  - otma\_receive\_async API 566
  - otma\_send\_async API 564
  - otma\_send\_receive API 561
  - otma\_send\_receivex API 563
  - sample programs 569
- calls, DB
  - CIMS 3
  - CLSE 5
  - DEQ 6
  - DLET 8
  - FLD 9
  - GHNP 16
  - GHU 19
  - GN 11
  - GNP 16
  - GU 19
  - GUR 21
  - ISRT 25
  - OPEN 29
  - POS 29
  - REPL 32
  - RLSE 34
- calls, system service
  - APSB (allocate PSB) 37
  - CHKP (basic) 38
  - CHKP (symbolic) 39
  - GMSG (get message) 42
  - ICMD (issue command) 45
  - INIT (initialize) 46
  - INQY (inquiry) 54
  - LOG (log) 63
  - PCB (schedule a PSB) 65
  - RCMD (retrieve command) 66
  - ROLB (roll back) 67
  - SETS/SETU (set a backout point) 70
  - SNAP 71
  - STAT (statistics) 74
  - SYNC (synchronization point) 76
  - TERM (terminate) 77
  - XRST (extended restart) 77
- change call 88
- changing the values of a segment's fields 195
- CHAR
  - data type 623
- character 619
- CHARACTER data type
  - description 623
- character string
  - assignment 626
  - comparison 626
  - constants 627
  - description 623
  - empty 623
- checkpoint (CHKP)
  - command
    - description 165
    - example 165
    - format 165
    - issuing 165
    - options 165
    - restrictions 165
    - usage 165
  - checkpoint call, basic 126
  - checkpoint call, symbolic 127
  - CHKP (basic checkpoint) call
    - CHKP (basic checkpoint) call
      - description 38
      - format 38
      - parameters 38
      - usage 38
  - CHKP (Checkpoint)
    - command
      - description 165
      - example 165
      - format 165
      - issuing 165
      - options 165
      - restrictions 165
      - usage 165
  - CHKP (symbolic checkpoint) call
    - CHKP (symbolic checkpoint) call
      - description 39
      - format 39
      - parameters 39
      - usage 39
  - CHKP call function 245
  - CHKPT=EOV parameter 166
  - CHNG call 88
    - and OTMA environment 88
      - description 88
      - format 88
      - parameters 88
      - restrictions 88
      - summary 82
      - usage 88
  - CHNG call function 245
  - CICS online programs 65
    - PCB call 65
    - TERM call 77
  - CIMS call
    - description 3
    - format 3
    - parameters 3
    - usage 3
  - class, record segment 222
  - CLEAR key 429
  - CLEAR PARTITION key 429
  - Clob interface
    - result set 677
    - retrieveXML 677
  - CLOSE
    - statement
      - description 639
      - example 639
  - closing a GSAM database explicitly 5
  - CLSE (Close) call
    - CLSE (Close) call
      - description 5
      - format 5
      - parameters 5
- CLSE (Close) call (*continued*)
  - usage 5
- CLSQR command 286
  - format 286
- CMD call
  - description 95
  - examples 95
  - format 95
  - parameters 95
  - restrictions 95
  - summary 81
  - usage 95
- CMD call function 245
- CMDVLRM reply message 331
  - format 331
- CNTQRY command 288
  - format 288
- COBOL application program
  - host structure 630
  - host variable
    - description 629
  - INCLUDE SQLIMSCA 671
- colon
  - host variable in SQL 629
- column
  - derived
    - INSERT statement 649
    - UPDATE statement 666
  - naming convention 620
- Command (CMD) call
  - See CMD call 95
- Command (ICAL) call
  - See ICAL call 101
- command codes 214, 218, 220
  - A
    - description 214
  - C
    - description 214
  - D
    - examples 214
    - Get calls 215
    - ISRT call 215
    - P processing option 215
  - DL/I calls 212
  - F
    - Get calls 217
    - HERE insert rule 25
    - ISRT call 217
  - G
    - description 218
  - L
    - FIRST insert rule 25, 218
    - Get calls 218
  - M 229
  - N 219
  - Null 226
  - O
    - description 220
  - P 221
  - Q 222
  - R 230
  - reference 212
  - S 231
  - U 224
  - V 225
  - Z 233

- command-level programs
    - command-level programs
      - syntax of EXEC DLI
        - commands 162
  - command, path 195
  - commands
    - EXEC DLI
      - ACCEPT 165
      - CHKP (Checkpoint) 165
      - DEQ (Dequeue) 166
      - DLET (Delete) 167
      - GN (Get Next) 169
      - GNP (Get Next in Parent) 174
      - GU (Get Unique) 180
      - ISRT (Insert) 185
      - LOAD 190
      - LOG 191
      - POS (Position) 192
      - QUERY 193
      - REFRESH 194
      - REPL (Replace) 195
      - RETRIEVE 199
      - ROLB (Rollback) 200
      - ROLL 201
      - ROLS (Rollback to SETS or SETU) 202
      - SCHD (Schedule) 204
      - SETS (Set a Backout Point) 205
      - SETU (Set a Backout Point Unconditionally) 206
      - STAT (Statistics) 207
      - summary 163
      - SYMCHKP (Symbolic Checkpoint) 208
      - TERM (Terminate) 210
      - XRST (Extended Restart) 210
    - symbolic checkpoint 208
    - system service 162
  - commands allowed, EXEC DLI 162
  - comment
    - SQL 619
  - COMMENT statement
    - conditional (T) 258
    - unconditional (U) 258
  - COMPARE statement
    - COMPARE AIB 260
    - COMPARE DATA 260
    - COMPARE PCB 261
    - introduction 259
  - comparison
    - compatibility rules 624
    - strings 626
  - compatibility
    - 3270 printer and SLU 1 438
    - converting device definitions to SLU P 438
    - data types 624
    - rules 624
    - SLU P 438
  - COMPR= operand (DIV statement), specifying 467
  - COND= operand (DPAGE statement), specifying 476
  - connector, field search argument (FSA) 9
  - constant
    - character string 627
    - decimal 627
  - constant (*continued*)
    - floating-point 627
    - hexadecimal 627
    - integer 627
  - CONTINUE
    - clause of WHENEVER statement 668
  - control blocks, MFS
    - chained control blocks 419
  - control character 619
  - conversion
    - 3270 device format, example 436
    - device formats 435
  - copy function
    - bit 4 of SCA, byte 1 452
    - cursor positioning
      - for output messages 452
    - description 537
    - dynamic attribute modification, output message formats
      - specifying attributes 452
  - CTL (PUNCH) statement 267
  - current position
    - qualification 224
  - current position in the database, determining the 199
  - cursor
    - closing
      - CLOSE statement 639
      - error in UPDATE 667
      - naming convention 620
    - opening
      - errors 652
      - OPEN statement 651
    - using
      - DECLARE CURSOR statement 639
      - FETCH statement 645
  - cursor position input 492
  - cursor positioning
    - 3270 Information Display System selector pen 446
    - for input messages 446, 520
    - for output messages
      - CURSOR operand 476
      - dynamic 520
      - selector pen, 3270
        - application program device-dependent information 446
        - effect on input fields 446
  - cursor-name clause
    - DECLARE CURSOR statement 640
    - FETCH statement 646
  - CURSOR= operand (DPAGE statement), specifying 476
- ## D
- data mapping, define with MAXDEF command 371
  - data structures
    - hierarchy 618
    - types 618
  - data type
    - built-in 620
    - character string 623
    - compatibility matrix 624
  - data type (*continued*)
    - datetime 623
    - list of built-in types 620
    - numeric 622
  - database
    - calls
      - summary 1
    - deallocating resources 41
    - determining the current position in the 199
    - establishing a starting position in a 180
    - position
      - establishing using GU 19
    - database versioning INIT VERSION call 46
  - databases
    - hierarchical
      - comparison to relational 618
    - relational
      - comparison to hierarchical 618
  - date
    - data type 624
  - DATE
    - data type
      - description 624
  - datetime
    - data types
      - description 623
  - datetime host variables
    - data type
      - description 624
  - DB PCB
    - status codes
      - NU 46
  - DB PCB (database program communication block) 43
    - AIB (application interface block )I/O area
      - in GSCD call 43
    - I/O PCB
      - in GSCD 43
    - in GSCD 43
    - status codes
      - NA 46
  - DBCS (double byte character set)
    - definition 514
    - types of fields 514
  - DBCS/EBCDIC mixed fields
    - description 514
    - horizontal tab (SCS1 device) 514
    - input control 514
    - SO/SI control characters in 514
  - DBCS/EBCDIC mixed literals
    - continuation rules for 514
    - description 514
    - specifying as DFLD/MFLD literals 514
  - dbpcbStream data structure 353
    - format 353
  - DBQUERY
    - using with INIT call 46
  - DDM (distributed data management architecture)
    - command objects 283
    - commands 281, 283
    - commit processing 282

DDM (distributed data management architecture) (*continued*)

- data structures, product-unique 351
- DSSHDR syntax 282
- global transaction processing 282
- local transaction processing 282
- parameters, product-unique 351
- replies 281
- syntax 281
- terms 281

DDM (distributed data management)

- abnormal ending of unit of work 326
- ABNUOWRM reply message 326
- access security reply object 329
- access to database completed reply message 327
- accessing database 283
- ACCRDB command 283
- ACCRDBRM reply message 327
- ACCSEC command 285
- ACCSECRD reply object 329
- AGNPRMRM reply message 330
- aibStream data structure 352
- closing a query 286
- CLSQR command 286
- CMDVLTRM reply message 331
- CNTQRY command 288
- command violation reply message 331
- continuing a query 288
- data field 312
- database access failed reply message 342
- database deallocation completed reply message 331
- database not accessed reply message 344
- database not found reply message 345
- database update reply message 346
- dbpcbStream data structure 353
- deallocating database 289
- DEALLOCDB command 289
- DEALLOCDBRM reply message 331
- DL/I function 290
- DLIFUNC command object 290
- end of query reply message 332
- end unit of work reply message 334
- ENDQRYRM reply message 332
- ENDUOWRM reply message 334
- exchange server attributes 292, 335
- EXCSAT command 292
- EXCSATRD reply object 335
- EXCSQLMM command 294
- executing immediate SQL 294
- field entry 300
- FLDENTRY command object 300
- IMS call reply message 336
- IMSCALL command 301
- IMSCALLRM reply message 336
- INAIB command object 302
- input AIB data 302
- iopcbStream data structure 354
- issuing an IMS call 301
- not authorized to database reply message 343

DDM (distributed data management) (*continued*)

- open query failure reply message 337
- open query reply message 338
- opening a query 304
- OPNQFLRM reply message 337
- OPNQRY command 304
- OPNQRYRM reply message 338
- OUTAIBDBPCB parameter 355
- OUTAIBIOPCB parameter 356
- Output AIBDBPCB 355
- output AIBIOPCB 356
- permanent agent error reply message 330
- QRYDSC reply object 339
- QRYDTA reply object 340
- QRYPOPRM reply message 341
- query answer set data reply object 340
- query answer set description reply object 339
- query previously opened reply message 341
- RDBAFLRM reply message 342
- RDBATHRM reply message 343
- RDBNACRM reply message 344
- RDBNFNRM reply message 345
- RDBUPDRM reply message 346
- release locks reply message 347
- releasing database locks 311
- reply messages 326
- resource limits reached reply message 348
- RLSE command 311
- RLSERM reply message 347
- RSCLMTRM reply message 348
- RTRVFLD command object 312
- SECCHK command 313
- SECCHKRM reply message 349
- security access 285
- security check 313
- security check reply message 349
- SQL error condition reply message 350
- SQLERRRM reply message 350
- SSA object list 325
- SSALIST command object 325

DDM (distributed data management) architecture

- AIBRSNM1 parameter 351
- AIBRSNM2 parameter 351
- AIBSFUNC parameter 352
- RDBNAM parameter 357
- SECCHK command 313
- security check 313
- SSA parameter 357
- SSACOUNT parameter 358
- UPDCNT parameter 358

DDM (distributed data management) Architecture

- AIBOALEN parameter 351

DDM command objects

- DLIFUNCFLG command object (X'CC09') 291
- FLDENTRYREL command object (X'CC0C') 301

DDM command objects (*continued*)

- RTRVFLDREL Command object (X'CC0B') 312
- SEGMLIST Command object (X'CC0A') 314

deadlock occurrence

- application programs 46
- batch programs, in 46

deallocate PSB call 129

DEALLOCDB command 289

- format 289

DEALLOCDBRM reply message 331

- format 331

debugging, IMSRXTRC 370

decimal

- constants 627
- numbers 623

DECIMAL

- data type 623

DECLARE CURSOR statement

- description 639
- example 640

DECLARE STATEMENT statement

- description 640
- example 641

DEDB (data entry database)

- command codes 226
- PCBs and DL/I calls 234

default system control area 510

define a data mapping with MAXDEF command 371

DELETE

- statement

  - description 641
  - example 642

Delete (DLET) call

- description 8
- format 8
- parameters 5, 8, 29
- SSA 8
- usage 8

Delete (DLET) command

- description 167
- example 167
- format 167
- options 167
- restrictions 167

deleting

- rows from a table 641

dependent segments

- retrieving

  - sequentially 174
  - the location of a 192

sequential

- retrieving the location of the last one inserted 192

dependents of a segment, removing 167

DEQ (Dequeue) call

- DEQ (Dequeue) call

  - description 6

format

- Fast Path 6
- full function 6

function 245

parameters

- Fast Path 6
- full function 6

- DEQ (Dequeue) call *(continued)*
  - Q command code 6, 222
  - restrictions 6
  - summary 1
  - usage 6
- DEQ (Dequeue) command
  - DEQ (Dequeue) command
    - description 166
  - example 166
  - format 166
  - options 166
  - restrictions 166
  - usage 166
- Dequeue (DEQ) call
  - description 6
  - format
    - Fast Path 6
    - full function 6
  - function 245
  - parameters
    - Fast Path 6
    - full function 6
  - Q command code 6, 222
  - summary 1
  - usage 6
- Dequeue (DEQ) command
  - description 166
  - example 166
  - format 166
  - options 166
  - restrictions 166
  - usage 166
- DESCRIBE OUTPUT statement 642
- descriptor
  - naming convention 620
- design objectives, application 419
- designator character 446
- determining the current position in the database 199
- DEV statement 496
  - FEAT= operand 426
  - FORS= operand 525
  - FTAB= operand 496
  - HTAB= operand 523
  - PAGE= operand 522
  - SLDx= operand 523
  - SUB= operand 502
  - TYPE= operand 426
  - VT= operand 523
  - VTAB= operand 523
  - WIDTH= operand 523
- device control characters 506
- device feature selection 426
- device format conversion 435
- device formats, default 426
- device input format 550
- device output format 550
- device page 492
- DFLD/MFLD literal
  - containing DBCS/EBCDIC mixed data 514
- DFS.EDT 465
- DFS.EDTN 465
- DFS057I block error message 546
- DFS1150 492
- DFSDDLTO
  - call statements
    - CALL FUNCTION statement 241
    - FEEDBACK DATA statement 244
    - OPTION DATA statement 245
- DFSDDLTO (DL/I Test Program) 235
- DFSDF1 546
- DFSDF2 546
- DFSDF4 546
- DFS DSP01 546
- DFSIGNI 546
- DFSIGNJ 546
- DFSIGNN 546
- DFSIGNP 546
- DFSM0 547
- DFSM01 546
- DFSM02 546
- DFSM03 546
- DFSM04 546
- DFSM05 546
- DFSME000 493
- DFSMI1 546
- DFSMI2 546
- DFSMI4 546
- DFSPWSIO
  - DFSPWSH include files 595
  - DFSQGETS API 604
  - DFSQSETS API 606
  - DFSXGETS API 609
  - DFSXSETS API 611
  - overview 595
  - return codes 614
- DFSREXXU, example user exit routine 361
- DFSUDT0x (device characteristics table)
  - description 548
  - MFS Device Characteristics Table utility 548
- DIB (DL/I interface block)
  - information, obtaining the most recent 194
- DIF (device input format)
  - definition 550
  - input formatting functions 484
  - language statements used to create 467
  - DIV 467
  - DPAGE 476
  - relationship to other control blocks 419
  - selection 426
- digit, description in IMS 619
- distributed data management (DDM)
  - abnormal ending of unit of work 326
  - ABNUOWRM reply message 326
  - access security reply object 329
  - access to database completed reply message 327
  - accessing database 283
  - ACCRDB command 283
  - ACCRDBRM reply message 327
  - ACCSEC command 285
  - ACCSECRD reply object 329
  - AGNPRMRM reply message 330
  - aibStream data structure 352
  - closing a query 286
  - CLSQR command 286
- distributed data management (DDM) *(continued)*
  - CMDVLRM reply message 331
  - CNTQRY command 288
  - command violation reply message 331
  - continuing a query 288
  - data field 312
  - database access failed reply message 342
  - database deallocation completed reply message 331
  - database not accessed reply message 344
  - database not found reply message 345
  - database update reply message 346
  - dbpcbStream data structure 353
  - deallocating database 289
  - DEALLOCDB command 289
  - DEALLOCDBRM reply message 331
  - DL/I function 290
  - DLIFUNC command object 290
  - end of query reply message 332
  - end unit of work reply message 334
  - ENDQRYRM reply message 332
  - ENDUOWRM reply message 334
  - exchange server attributes 292, 335
  - EXCSAT command 292
  - EXCSATRD reply object 335
  - EXCSQLIMM command 294
  - executing immediate SQL 294
  - field entry 300
  - FLDENTRY command object 300
  - IMS call reply message 336
  - IMSCALL command 301
  - IMSCALLRM reply message 336
  - INAIB command object 302
  - input AIB data 302
  - iopcbStream data structure 354
  - issuing an IMS call 301
  - not authorized to database reply message 343
  - open query failure reply message 337
  - open query reply message 338
  - opening a query 304
  - OPNQFLRM reply message 337
  - OPNQRY command 304
  - OPNQRYRM reply message 338
  - OUTAIBDBPCB parameter 355
  - OUTAIBIOPCB parameter 356
  - Output AIBDBPCB 356
  - output AIBIOPCB 356
  - permanent agent error reply message 330
  - QRYDSC reply object 339
  - QRYDTA reply object 340
  - QRYPOPRM reply message 341
  - query answer set data reply object 340
  - query answer set description reply object 339
  - query previously opened reply message 341
  - RDBAFLRM reply message 342
  - RDBATHRM reply message 343

distributed data management (DDM)  
(continued)

- RDBNACRM reply message 344
- RDBNFNRM reply message 345
- RDBUPDRM reply message 346
- release locks reply message 347
- releasing database locks 311
- reply messages 326
- resource limits reached reply message 348
- RLSE command 311
- RLSERM reply message 347
- RSCLMTRM reply message 348
- RTRVFLD command object 312
- SECCHK command 313
- SECCHKRM reply message 349
- security access 285
- security check 313
- security check reply message 349
- SQL error condition reply message 350
- SQLERRRM reply message 350
- SSA object list 325
- SSALIST command object 325

distributed data management (DDM)  
architecture

- AIBOALEN parameter 351
- AIBRSNM1 parameter 351
- AIBRSNM2 parameter 351
- AIBSFUNC parameter 352
- RDBNAM parameter 357
- SECCHK command 313
- security check 313
- SSA parameter 357
- SSACOUNT parameter 358
- UPDCNT parameter 358

distributed data management architecture  
(DDM)

- command objects 283
- commands 281, 283
- commit processing 282
- data structures, product-unique 351
- DSSHDR syntax 282
- global transaction processing 282
- local transaction processing 282
- parameters, product-unique 351
- replies 281
- syntax 281
- terms 281

Distributed relational database  
architecture (DRDA)

- DLIFUNCFLG command object (X'CC09') 291
- FLDENTRYREL command object (X'CC0C') 301
- RTRVFLDREL Command object (X'CC0B') 312
- SEGMLIST Command object (X'CC0A') 314

Distributed Relational Database  
Architecture (DRDA)

- abnormal ending of unit of work 326
- ABNUOWRM reply message 326
- access security reply object 329
- access to database completed reply message 327
- accessing database 283

Distributed Relational Database

Architecture (DRDA) (continued)

- ACCRDB command 283
- ACCRDBRM reply message 327
- ACCSEC command 285
- ACCSECRD reply object 329
- AGNPRMRM reply message 330
- aibStream data structure 352
- closing a query 286
- CLSQRYS command 286
- CMDVLTRM reply message 331
- CNTQRY command 288
- command violation 331
- continuing a query 288
- data field 312
- data structures
  - aibStream data structure 352
  - dbpcbStream data structure 353
  - iopcbStream data type 354
- database access failed reply message 342
- database deallocation completed 331
- database not accessed reply message 344
- database not found reply message 345
- database update reply message 346
- dbpcbStream data structure 353

DDM command objects

- data field 312
- DL/I function 290
- DLIFUNC 290
- field entry 300
- FLDENTRY 300
- INAIB 302
- input AIB data 302
- RTRVFLD 312
- SSA object list 325
- SSALIST 325

DDM commands

- accessing database 283
  - ACCRDB 283
  - ACCSEC 285
  - closing a query 286
  - CLSQRYS 286
  - CNTQRY 288
  - continuing a query 288
  - deallocating database 289
  - DEALLOCDB 289
  - exchange server attributes 292
  - EXCSAT 292
  - EXCSQLIMM 294
  - executing immediate SQL 294
  - IMSCALL 301
  - issuing an IMS call 301
  - opening a query 304
  - OPNQRY 304
  - releasing database locks 311
  - RLSE 311
  - SECCHK 313
  - security access 285
  - security check 313
- DDM parameters
- OUTAIBDBPCB 355
  - OUTAIBIOPCB 356
  - Output AIBDBPCB 355
  - output AIBIOPCB 356

Distributed Relational Database

Architecture (DRDA) (continued)

DDM reply messages

- abnormal ending of unit of work 326
  - ABNUOWRM 326
  - access to database completed 327
  - ACCRDBRM 327
  - AGNPRMRM 330
  - CMDVLTRM 331
  - database access failed 342
  - database deallocation completed 331
  - database not accessed 344
  - database not found 345
  - database update 346
  - DEALLOCDBRM 331
  - end of query 332
  - end unit of work 334
  - ENDQRYRM 332
  - ENDUOWRM 334
  - IMS calls 336
  - IMSCALLRM 336
  - not authorized to database 343
  - open query 338
  - open query failure 337
  - OPNQFLRM 337
  - OPNQRYRM 338
  - permanent agent errors 330
  - QRYPOPRM 341
  - query previously opened 341
  - RDBAFLRM 342
  - RDBATHRM 343
  - RDBNACRM 344
  - RDBNFNRM 345
  - RDBUPDRM 346
  - release locks 347
  - resource limits reached 348
  - RLSERM 347
  - RSCLMTRM 348
  - SECCHKRM 349
  - security check 349
  - SQL error condition 350
  - SQLERRRM 350
- DDM reply objects
- access security 329
  - ACCSECRD 329
  - exchange server attributes 335
  - EXCSATRD 335
  - QRYDSC 339
  - QRYDTA 340
  - query answer set data 340
  - query answer set description reply object 339
- deallocating database 289
- DEALLOCDB command 289
  - DEALLOCDBRM reply message 331
  - DL/I function 290
  - DLIFUNC command object 290
  - end of query reply message 332
  - end unit of work reply message 334
  - ENDQRYRM reply message 332
  - ENDUOWRM reply message 334
  - exchange server attributes 292, 335
  - EXCSAT command 292
  - EXCSATRD reply object 335
  - EXCSQLIMM command 294

Distributed Relational Database  
 Architecture (DRDA) (*continued*)  
 executing immediate SQL 294  
 field entry 300  
 FLDENTRY command object 300  
 IMS call reply message 336  
 IMSCALL command 301  
 IMSCALLRM reply message 336  
 INAIB command object 302  
 input AIB data 302  
 iopcbStream data structure 354  
 issuing an IMS call 301  
 not authorized to database reply  
 message 343  
 open query failure reply  
 message 337  
 open query reply message 338  
 opening a query 304  
 OPNQFLRM reply message 337  
 OPNQRY command 304  
 OPNQRYRM reply message 338  
 OUTAIBDBPCB parameter 355  
 OUTAIBIOPCB parameter 356  
 Output AIBDBPCB 355  
 output AIBIOPCB 356  
 permanent agent errors 330  
 QRYDSC reply object 339  
 QRYDTA reply object 340  
 QRYPOPRM reply message 341  
 query answer set data reply  
 object 340  
 query answer set description reply  
 object 339  
 query previously opened reply  
 message 341  
 RDBAFLRM reply message 342  
 RDBATHRM reply message 343  
 RDBNACRM reply message 344  
 RDBNFNRM reply message 345  
 RDBUPDRM reply message 346  
 release locks reply message 347  
 releasing database locks 311  
 reply messages 326  
 resource limits reached reply  
 message 348  
 RLSE command 311  
 RLSERM reply message 347  
 RSCLMTRM reply message 348  
 RTRVFLD command object 312  
 SECCHK command 313  
 SECCHKRM reply message 349  
 security access 285  
 security check 313  
 security check reply message 349  
 SQL error condition reply  
 message 350  
 SQLERRRM reply message 350  
 SSA object list 325  
 SSALIST command object 325

Distributed Relational Database  
 Architecture (DRDA) Specification  
 AIBOALEN parameter 351  
 AIBRSNM1 parameter 351  
 AIBRSNM2 parameter 351  
 AIBSFUNC parameter 352  
 RDBNAM parameter 357  
 SECCHK command 313

Distributed Relational Database  
 Architecture (DRDA) Specification  
 (*continued*)  
 security check 313  
 SSA parameter 357  
 SSACOUNT parameter 358  
 UPDCNT parameter 358

DIV statement 498  
 &DPN= operand 467  
 COMPR= operand 467  
 HDRCTL= operand 525  
 NOSPAN= operand 467  
 NULL= operand 467, 498  
 OFTAB= operand  
 output mode 529  
 specifying 467  
 OPTIONS= operand 467, 525  
 PRN= operand 467  
 RCDCTL= operand 467, 525  
 RDPN= operand 467  
 RPRN= operand 467  
 SPAN= operand 467  
 TYPE= operand 467

DL/I  
 setting a backout point 205

DL/I call functions  
 special DFSDDLT0  
 END 257  
 SKIP 257  
 STAK 257  
 START 257

supported  
 CHKP 245  
 CHNG 245  
 CMD 245  
 DEQ 245  
 DLET 245  
 FLD 245  
 GCMD 245  
 GHN 245  
 GHNP 245  
 GHU 245  
 GMSG 245  
 GN 245  
 GNP 245  
 ICAL 245  
 ICMD 245  
 INIT 245  
 INQY 245  
 ISRT 245  
 LOG 245  
 POS 245  
 PURG 245  
 RCMD 245  
 REPL 245  
 ROLB 245  
 ROLL 245  
 ROLS 245  
 ROLX 245  
 SETO 245  
 SETS 245  
 SNAP 245  
 STAT 245  
 SYNC 245  
 XRST 245

DL/I call functions, examples 248

DL/I call functionsDL/  
 supported  
 GU 245  
 GUR 245

DL/I calls (general information)  
 qualifying calls  
 concatenated key 214  
 relationships to PCBs, FF PCBs 234

DL/I calls for transaction management  
 AUTH call 83  
 call summary 81  
 CHNG call 88  
 CMD call 95  
 GCMD call 97  
 GN call 98  
 GU call 99  
 ISRT call 112  
 PURG call 115  
 SETO call 117

DL/I calls, database management  
 CIMS 3  
 DEDB (data entry database)  
 root segments, order 11  
 DEQ 6  
 DL/I calls, database management  
 CLSE 5  
 FLD 9  
 GNHP call 11  
 DLET 8  
 FLD 9  
 FLD (Field) call  
 description 9  
 get hold next (GHN), usage 11  
 Get Next (GN) call  
 hold form (GHN) 11  
 parameters 11  
 SSA 11  
 usage 11  
 GHN (get hold next), usage 11  
 GHNP call 16  
 GN 11  
 GN (Get Next) call  
 hold form (GHN) 11  
 parameters 11  
 SSA 11  
 usage 11  
 GNP 16  
 GU 19  
 GUR 21  
 HDAM  
 order of root segments 11  
 ISRT 25  
 OPEN 29  
 PHDAM database 11  
 POS 29  
 randomizing routine  
 exit routine 11  
 REPL 32  
 RLSE 34  
 summary 1

DL/I calls, system service  
 APSB 37  
 CHKP 38, 39  
 CHKP (basic) 38  
 description 35  
 DPSB 41  
 GMSG 42



- DL/I calls, system service (*continued*)
  - GSCD 43
  - INIT 46
  - INQY 54
  - LOG 63
  - PCB 65
  - ROLB 67
  - ROLL 68
  - ROLS 68
  - SETS/SETU 70
  - SNAP 71
  - STAT 74, 77
  - summary 35
  - SYNC 76
  - XRST 77
- DL/I processing
  - batch processing options 291, 301, 312, 314
- DL/I return codes (REXX) 364
- DL/I system service calls 123
  - APSB call 125
  - Basic CHKP call 126
  - call summary 123
  - DPSB call 129
  - GSCD Call 131
  - INIT call 135
  - INQY call 138
  - LOG call 149
  - ROLB call 153
  - ROLL Call 154
  - ROLS call 155
  - SETS call 157
  - SETU call 157
  - Symbolic CHKP call 127
  - SYNC call 159
  - XRST call 159
- DL/I test program (DFSDDLTO)
  - JCL requirements
    - PRINTDD DD statement 275
    - SYSIN DD statement 274
    - SYSIN2 DD statement 274
- DL/I Test Program (DFSDDLTO)
  - control statements 273
    - guidelines 235
  - execution in IMS regions 277
  - explanation of return codes 277
  - hints on usage 277
  - JCL requirements 273, 275
  - overview 235
  - restarting input stream 275
- DLET (Delete) call
  - DLET (Delete) call
    - description 8
  - format 8
  - parameters 5, 8, 29
  - SSA 8
  - usage 8
- DLET (Delete) command
  - Delete (DLET) command
    - usage 167
  - DLET (Delete) command
    - description 167
    - usage 167
  - example 167
  - format 167
  - options 167
  - restrictions 167
- DLET (Delete) command (*continued*)
  - usage
    - DLET (Delete) command 167
- DLET call function 245
- DLIFUNC command object 290
  - format 290
- DLIFUNCFLG command object (X'CC09') 291
- DLIINFO
  - . (period) usage 369
  - REXX extended command 369
- DOCMD exec 387
- DOF (device output format)
  - associated MFS functions 506
  - definition 550
  - language statements used to create 467
  - DIV 467
  - DPAGE 476
  - relationship to other control blocks 419
  - selection 426
- double byte character set 514
- DOUBLE data type
  - description 622
- DOUBLE PRECISION data type
  - description 622
- double precision floating-point number 622
- DPAGE 492
  - ACTVPID= operand 476, 543
  - COND= operand 476
  - CURSOR= operand 476
  - input 492
  - MULT= operand 476
  - OFTAB= operand
    - output mode 529
    - specifying 476
  - ORIGIN= operand 476
  - overview 492
  - PD= operand 476
  - SELECT= operand 476
  - selection
    - using conditional data 503
    - using conditional test on the data 503
    - using DSN transmission chains 503
    - specifying conditional 503
    - specifying unconditional 503
- DPM (distributed presentation management)
  - control character translation 451, 506
  - deleting nulls on input 498
  - GRAPHIC= operand (SEG statement)
    - use 451
  - increasing performance 441
  - naming conventions 525
  - output message header examples 525
  - version identification 550
- DPN field
  - control block linkages 426
  - DIV statement 467
  - MFS formatting 482
- DPSB call 129
  - description 41, 129
  - format 41, 129
- DPSB call (*continued*)
  - parameters 41, 129
  - restrictions 129
  - summary 81
  - usage 41, 129
- DRDA
  - DLIFUNCFLG command object (X'CC09') 291
  - FLDENTRYREL command object (X'CC0C') 301
  - RTRVFLDREL Command object (X'CC0B') 312
  - SEGMLIST Command object (X'CC0A') 314
- DRDA (Distributed Relational Database Architecture)
  - abnormal ending of unit of work 326
  - ABNUOWRM reply message 326
  - access security reply object 329
  - access to database completed reply message 327
  - accessing database 283
  - ACCRDB command 283
  - ACCRDBRM reply message 327
  - ACCSEC command 285
  - ACCSECRD reply object 329
  - AGNPRMRM reply message 330
  - aibStream data structure 352
  - closing a query 286
  - CLSQRY command 286
  - CMDVLTRM reply message 331
  - CNTQRY command 288
  - command violation 331
  - continuing a query 288
  - data field 312
  - data structures
    - aibStream data structure 352
    - dbpcbStream data structure 353
    - iopcbStream data type 354
  - database access failed reply message 342
  - database deallocation completed 331
  - database not accessed reply message 344
  - database not found reply message 345
  - database update reply message 346
  - dbpcbStream data structure 353
- DDM command objects
  - data field 312
  - DL/I function 290
  - DLIFUNC 290
  - field entry 300
  - FLDENTRY 300
  - INAIB 302
  - input AIB data 302
  - RTRVFLD 312
  - SSA object list 325
  - SSALIST 325
- DDM commands
  - accessing database 283
  - ACCRDB 283
  - ACCSEC 285
  - closing a query 286
  - CLSQRY 286
  - CNTQRY 288
  - continuing a query 288

DRDA (Distributed Relational Database Architecture) (*continued*)

- DDM commands (*continued*)
  - deallocating database 289
  - DEALLOCDB 289
  - exchange server attributes 292
  - EXCSAT 292
  - EXCSQLIMM 294
  - executing immediate SQL 294
  - IMSCALL 301
  - issuing an IMS call 301
  - opening a query 304
  - OPNQRY 304
  - releasing database locks 311
  - RLSE 311
  - SECCHK 313
  - security access 285
  - security check 313
- DDM parameters
  - OUTAIBDBPCB 355
  - OUTAIBIOPCB 356
  - Output AIBDBPCB 355
  - output AIBIOPCB 356
- DDM reply messages
  - abnormal ending of unit of work 326
  - ABNUOWRM 326
  - access to database completed 327
  - ACCRDBRM 327
  - AGNPRMRM 330
  - CMDVLTRM 331
  - database access failed 342
  - database deallocation completed 331
  - database not accessed 344
  - database not found 345
  - database update 346
  - DEALLOCDBRM 331
  - end of query 332
  - end unit of work 334
  - ENDQRYRM 332
  - ENDUOWRM 334
  - IMS calls 336
  - IMSCALLRM 336
  - not authorized to database 343
  - open query 338
  - open query failure 337
  - OPNQFLRM 337
  - OPNQRYRM 338
  - permanent agent errors 330
  - QRYPOPRM 341
  - query previously opened 341
  - RDBAFLRM 342
  - RDBATHRM 343
  - RDBNACRM 344
  - RDBNFNRM 345
  - RDBUPDRM 346
  - release locks 347
  - resource limits reached 348
  - RLSERM 347
  - RSCLMTRM 348
  - SECCHKRM 349
  - security check 349
  - SQL error condition 350
  - SQLERRRM 350
- DDM reply objects
  - access security 329

DRDA (Distributed Relational Database Architecture) (*continued*)

- DDM reply objects (*continued*)
  - ACCSECRD 329
  - exchange server attributes 335
  - EXCSATRD 335
  - QRYDSC 339
  - QRYDTA 340
  - query answer set data 340
  - query answer set description reply object 339
- deallocating database 289
- DEALLOCDB command 289
- DEALLOCDBRM reply message 331
- DL/I function 290
- DLIFUNC command object 290
- end of query reply message 332
- end unit of work reply message 334
- ENDQRYRM reply message 332
- ENDUOWRM reply message 334
- exchange server attributes 292, 335
- EXCSAT command 292
- EXCSATRD reply object 335
- EXCSQLIMM command 294
- executing immediate SQL 294
- field entry 300
- FLDENTRY command object 300
- IMS call reply message 336
- IMSCALL command 301
- IMSCALLRM reply message 336
- INAIB command object 302
- input AIB data 302
- iopcbStream data structure 354
- issuing an IMS call 301
- not authorized to database reply message 343
- open query failure reply message 337
- open query reply message 338
- opening a query 304
- OPNQFLRM reply message 337
- OPNQRY command 304
- OPNQRYRM reply message 338
- OUTAIBDBPCB parameter 355
- OUTAIBIOPCB parameter 356
- Output AIBDBPCB 355
- output AIBIOPCB 356
- permanent agent errors 330
- QRYDSC reply object 339
- QRYDTA reply object 340
- QRYPOPRM reply message 341
- query answer set data reply object 340
- query answer set description reply object 339
- query previously opened reply message 341
- RDBAFLRM reply message 342
- RDBATHRM reply message 343
- RDBNACRM reply message 344
- RDBNFNRM reply message 345
- RDBUPDRM reply message 346
- release locks reply message 347
- releasing database locks 311
- reply messages 326
- resource limits reached reply message 348

DRDA (Distributed Relational Database Architecture) (*continued*)

- RLSE command 311
- RLSERM reply message 347
- RSCLMTRM reply message 348
- RTRVFLD command object 312
- SECCHK command 313
- SECCHKRM reply message 349
- security access 285
- security check 313
- security check reply message 349
- SQL error condition reply message 350
- SQLERRRM reply message 350
- SSA object list 325
- SSALIST command object 325

DRDA (Distributed Relational Database Architecture) Specification

- AIBOALEN parameter 351
- AIBRSNM1 parameter 351
- AIBRSNM2 parameter 351
- AIBSFUNC parameter 352
- RDBNAM parameter 357
- SECCHK command 313
- security check 313
- SSA parameter 357
- SSACOUNT parameter 358
- UPDCNT parameter 358

DSCA (default system control area) 510

- autopaged output 529
- description 510
- destroying screen format 522
- ERASE/DO NOT ERASE option 453
- use 542

DSN (data structure name) 535

DSSHDR syntax 282

dynamic attribute modification, output message formats

- default attributes 511
- specifying extended field attributes 454

dynamic modification of EGCS data 456

dynamic SQL 617

- description 636
- EXECUTE statement 643
- execution 637
- INTO clause
  - DESCRIBE statement 642
- invocation of SELECT statement 637
- preparation 637
- SQLIMSDA 671

dynamically backing out changes 200, 201

## E

E (COMPARE) statement 259

EATTR= operand (DFLD statement) example 456

- use 511

EBCDIC format 492

edit routines, IMS-supplied

- field edit routine 492, 493

EGCS (extended graphic character set) 511

- /EBCDIC data, dynamic modification 456

EGCS (extended graphic character set) (continued)  
 description 511  
 modifying data 461  
 SO/SI framing characters 513  
 use with selector pen 446

END call function 257

end multiple page input request 536

ending a logical unit of work 165, 208

ENDMPPI request 536

ENDQRYRM reply message 332  
 format 332

ENDUOWRM reply message 334  
 format 334

environment (REXX)  
 address 361, 363  
 determining 364  
 DL/I calls (general information)  
 REXXTDLI 363  
 extended 363

erase all unprotected option (SCA/DSCA) 429

error  
 during update 667

ERROR key 502

establishing a starting position in a database 180

examples  
 ACCEPT command 165  
 CHKP (Checkpoint) command 165  
 D command code 215  
 DEQ (Dequeue) command 166  
 DFSDDLTO statements  
 COMMENT 258  
 DATA/PCB COMPARE 263  
 DD 275  
 DL/I call functions 248  
 IGNORE 265  
 OPTION 266  
 PUNCH 267  
 STATUS 269  
 SYSIN, SYSIN2, and PREINIT 275  
 WTO 272  
 WTOR 273  
 DFSREXXU user exit routine 361  
 DLET (Delete) command 167  
 GN (Get Next) command 169  
 GNP (Get Next in Parent) command 174  
 GU (Get Unique) command 180  
 ISRT (Insert) command 185  
 L command code 218  
 LOAD command 190  
 LOG command 191  
 N command code 219  
 Null command code 226  
 P command code 221  
 QUERY command 193  
 REFRESH command 194  
 REPL (Replace) 195  
 REPL (Replace) command 195  
 RETRIEVE command 199  
 ROLB (Rollback) command 200  
 ROLL command 201  
 ROLS (Rollback to SETS or SETU) command 202  
 SCHD (Schedule) command 204

examples (continued)  
 SETS (Set a Backout Point) command 205  
 STAT (Statistics) command 207  
 SYMCHKP (Symbolic Checkpoint) command 208  
 TERM (Terminate) command 210  
 U Command Code 224  
 V command code 225  
 XRST (Extended Restart) command 210

EXCSAT command 292  
 format 292

EXCSATRD reply object 335  
 format 335

EXCSQLIMM command 294

EXCSQLSET command  
 format 298

EXEC DLI  
 allowable commands 162  
 commands  
 ACCEPT 165  
 CHKP (Checkpoint) 165  
 DEQ (Dequeue) 166  
 DLET (Delete) 167  
 GN (Get Next) 169  
 GNP (Get Next in Parent) 174  
 GU (Get Unique) 180  
 ISRT (Insert) 185  
 LOAD 190  
 LOG 191  
 POS (Position) 192  
 QUERY 193  
 REFRESH 194  
 REPL (Replace) 195  
 RETRIEVE 199  
 ROLB (Rollback) 200  
 ROLL 201  
 ROLS (Rollback to SETS or SETU) 202  
 SCHD (Schedule) 204  
 SETS (Set a Backout Point) 205  
 SETU (Set a Backout Point Unconditionally) 206  
 STAT (Statistics) 207  
 SYMCHKP (Symbolic Checkpoint) 208  
 TERM (Terminate) 210  
 XRST (Extended Restart) 210

EXEC DLI  
 syntax of commands 162  
 program summary 163

EXEC statement, operands  
 DEVCHAR= 548

EXECIO  
 managing resources 361

executable statement 636

EXECUTE statement  
 description 643  
 example 644

expression  
 row-value 630

extended attribute data 494  
 input message fields 494  
 output devices, dynamic modification 511

extended commands 369

extended environment 363

extended functions 379

extended graphic character set 511

Extended Recovery Facility 522

Extended Restart (XRST)  
 with Symbolic Checkpoint (CHKP Symbolic) 39

Extended Restart (XRST) command  
 description 210  
 example 210  
 format 210  
 options 210  
 restrictions 210  
 usage 210

## F

Fast Path  
 FSA 9

FEAT= operand (DEV statement), specifying 426

FETCH statement  
 description 645  
 example 646

field edit exit routine  
 use 493

field edit routine  
 about 493  
 designing 494  
 DFSME000 493  
 using 494  
 using edit routines, IMS-supplied segment edit routine 493

field format  
 input message 446  
 output message 449

field name  
 FSA 9

field names  
 qualified 628

field search argument (FSA)  
 connector 9  
 field name 9  
 Op code 9  
 operand 9  
 reference 9  
 status code 9

field tab  
 example 496

fields  
 changing the values of a segment's 195

fill characters  
 DPAGE  
 FILL= operand 476  
 input message fields  
 MFS treatment 495  
 output device fields  
 MFS treatment 509  
 specifying 476

FILL= operand  
 DPAGE statement, specifying 476  
 multiple physical pages, input messages  
 specifying 476

Fill=NULL 492

- FIN (Finance Communication System) workstation
    - entering and exiting formatted mode 483
    - FTABs 496
    - input modes 495
    - physical page positioning 476
  - Finance Communication System 483
  - FIRST insert rule 185
  - FIRST insert rule, L command code 218
  - FLD (Field) call
    - format 9
    - FSAAs 9
    - parameters 9
    - summary 1
    - usage 9
  - FLD call function 245
  - FLDENTRY command object 300
  - FLDENTRYREL command object (X'CC0C') 301
  - FLOAT
    - data type
    - description 622
  - floating-point
    - constants 627
    - double precision number 622
  - force format write option (SCA/DSCA) 429
  - format library member selection 426
  - format set
    - IMS-provided format sets 545
  - format, message 482
    - input 482
      - device-dependent considerations 446, 452
    - output 439
    - output device-dependent considerations 448, 452
  - formats
    - ACCEPT command 165
    - CHKP (Checkpoint) command 165
    - DEQ (Dequeue) command 166
    - DLET (Delete) command 167
    - GN (Get Next) command 169
    - GNP (Get Next in Parent) command 174
    - GU (Get Unique) command 180
    - ISRT (Insert) command 185
    - LOAD command 190
    - LOG command 191
    - POS (Position) command 192
    - QUERY command 193
    - REFRESH command 194
    - REPL (Replace) command 195
    - RETRIEVE command 199
    - ROLB (Rollback) command 200
    - ROLL command 201
    - ROLS (Rollback to SETS or SETU) command 202
    - SCHD (Schedule) command 204
    - SETS (Set a Backout Point) command 205
    - SETU (Set a Backout Point Unconditionally) command 206
    - STAT (Statistics) command 207
    - SYMCHKP (Symbolic Checkpoint) command 208
  - formats (*continued*)
    - TERM (Terminate) command 210
    - XRST (Extended Restart) command 210
  - FORS= operand (DEV statement), use for DPM 525
  - framing characters (SO/SI) 513
  - FROM clause
    - DELETE statement 642
    - PREPARE statement 653
  - FSA (field search argument)
    - connector 9
    - field name 9
    - Op code 9
    - operand 9
    - reference 9
    - status code 9
  - FTAB= operand (DEV statement)
    - ALL 496
    - ALL parameter 497
    - defining 496
    - description 496
    - FORCE 496
    - forced FTABs, FORCE parameter 496
    - MIX 496
    - mixed FTABs, MIX parameter 496
    - with NULL=DELETE specified 498
  - full format write 429
  - full-function database
    - PCBs and DL/I calls 234
    - segment release 222
  - function
    - aggregate
    - field name 628
- G**
- GB (end of database), return status code 215
  - GCMD call 97
    - description 97
    - format 97
    - parameters 97
    - restrictions 98
    - status codes 97
    - summary 82
    - usage 97
  - GCMD call function 245
  - GE (segment not found), return status code 215
  - Get calls
    - D command code 215
    - F command code 217
    - function 245
    - L command code 218
    - Null command code 226
    - P command code 221
    - Q command code 222
    - U Command Code 224
    - V command code 225
  - Get Command (GCMD) call
    - See GCMD call 97
  - Get Hold Unique (GHU) description 19
  - Get Message (GMSG) call
    - description 42
    - format 42
    - parameters 42
  - Get Message (GMSG) call (*continued*)
    - restrictions 42
    - See GMSG call 129
  - Get Next (GN) call
    - description 11
    - format 11
  - Get Next (GN) command
    - description 169
    - examples 169
    - format 169
    - options 170
    - restrictions 169
    - usage 169
  - get next call 98
  - Get Next in Parent (GNP) call
    - description 16
    - effect in parentage 16
    - format 16
    - hold form (GHNP) 16
    - parameters 16
    - SSA 16
    - usage 16
      - linking with previous DL/I calls 16
      - processing with parentage 16
  - Get Next in Parent (GNP) command
    - description 174
    - examples 174
    - format 174
    - options 174
    - restrictions 174
    - usage 174
  - Get System Contents Directory (GSCD) call
    - description 43
    - format 43
    - parameters 43
    - usage 43
  - get system contents directory call 131
  - Get Unique (GU) call
    - description 19
    - DL/I calls, database management GHU call 19
    - format 19
    - hold form (GHU) 19
    - parameters 19
    - usage 19
  - Get Unique (GU) command
    - description 180
    - examples 180
    - format 180
    - Get Unique (GU) command
      - options 180
    - GU (Get Unique) command
      - options 180
      - GU (Get Unique) command 180
      - restrictions 180
      - usage 180
  - get unique call 99
  - Get Unique Record (GUR) call
    - description 21
    - format 21
    - parameters 21
    - usage 21
  - getting IMS database statistics 207

- GHNP
  - call 16
  - hold form 16
- GHU (Get Hold Unique), description 19
- GMSG call 129
  - description 42, 129
  - format 42, 129
  - parameters 42, 129
  - restrictions 42, 129
  - usage 42
  - use 129
- GN (Get Next) call
  - format 11
  - GN (Get Next) call
    - description 11
- GN (Get Next) command
  - description 169
  - examples 169
  - format 169
  - options 170
  - restrictions 169
  - usage 169
- GN call 98
  - description 98
  - format 98
  - parameters 98
  - restrictions 98
  - summary 82
  - usage 98
- GNP (Get Next in Parent) call
  - effect in parentage 16
  - format 16
  - GNP (Get Next in Parent) call
    - description 16
    - hold form (GHNP) 16
    - parameters 16
    - SSA 16
    - usage 16
      - linking with previous DL/I calls 16
      - processing with parentage 16
- GNP (Get Next in Parent) command
  - description 174
  - examples 174
  - format 174
  - options 174
  - restrictions 174
  - usage 174
- GO TO clause of WHENEVER
  - statement 668
- GRAPHIC= operand (SEG statement)
  - use 506
- GSAM (generalized sequential access method)
  - PCBs and DL/I calls 234
- GSCD (Get System Contents Directory) call
  - format 43
  - GSCD (Get System Contents Directory) call
    - description 43
    - parameters 43
    - usage 43
- GSCD call
  - description 131
  - format 131
  - parameters 131

- GSCD call (*continued*)
  - restrictions 131
  - summary 81
  - usage 131
- GU (Get Unique) call
  - description 19
  - format 19
  - Get Unique (GU) call
    - restrictions 19
  - hold form (GHU) 19
  - parameters 19
  - restrictions 19
  - usage 19
- GU (Get Unique) command
  - description 180
  - examples 180
  - format 180
  - restrictions 180
  - usage 180
- GU call 99
  - description 99
  - format 99
  - parameters 99
  - restrictions 99
  - summary 82
  - usage 99
- GUR (Get Unique Record) call
  - description 21
  - format 21
  - Get Unique Record (GUR) call
    - restrictions 21
  - parameters 21
  - restrictions 21
  - usage 21

## H

- HDRCTL= operand (DIV statement), use 525
- HERE insert rule 185
  - F command code 217
  - L command code 218
- hexadecimal constant 627
- hierarchic sequence 11
- host identifier 620
- host structure
  - description 630
- host variable
  - colon 629
  - description 629
- FETCH statement 645
- input 629
- naming convention 620
- output 629
- PREPARE statement 653
- HTAB= operand (DEV statement)
  - use 523

## I

- I/O area
  - for XRST 159
  - in CHKP (symbolic) call 39
  - in GMSG call 42
  - in INIT call 46
  - in INQY call 54

- I/O area (*continued*)
  - Initialize (INIT) call
    - usage 46
  - length in CHKP (symbolic) call 39
  - returned
    - keywords 29
    - map of 29
- I/O area format, AUTH call 83
- I/O PCB
  - PCBs and DL/I calls 234
- ICAL call
  - description 101
  - format 101
  - parameters 101
  - restrictions 101
  - return and reason codes 101
  - summary 81
  - usage 101
- ICMD call 134
  - commands that can be issued 45, 132
  - description 45, 46, 132
  - format 45, 132
  - parameters 45, 132
  - restrictions 45, 132
  - use 45, 132
- identifier in SQL
  - ordinary 620
- IGNORE (N or period (.)) statement 265
- IMS database statistics, obtaining 207
- IMS JDBC driversDatabaseMetaData interface
  - methods supported 397
- IMS JDBC driversDriverManager interface
  - methods supported 401
- IMS JDBC driversPreparedStatement interface
  - methods supported 402
- IMS JDBC driversResultSetMetaData interface
  - methods supported 410
- IMS JDBC driversStatement interface
  - methods supported 403
- IMS TM
  - password 495
- IMS Universal Database resource adapter
  - Common Client Interface (CCI) API support 411
  - Connection
    - methods supported 411
  - ConnectionFactory
    - methods supported 411
  - ConnectionMetaData
    - methods supported 412
  - Interaction
    - methods supported 412
  - javax.resource.cci.ResultSetInfo
    - methods supported 413
  - LocalTransaction
    - methods supported 412
  - RecordFactory
    - methods supported 414
  - ResourceAdapterMetaData
    - methods supported 413
- IMS Universal JCA/JDBC driver
  - driver support for JDBC 395

- IMS Universal JDBC driver
  - driver support for JDBC 395
  - ResultSet object
    - supported field constants 405
- IMS Universal JDBC driverClob interface
  - methods supported 395
- IMS Universal JDBC driverDataSource interface
  - methods supported 401
- IMS Universal JDBC driverParameterMetaData interface
  - methods supported 402
- IMS-provided formats
  - /DISPLAY command format 546
  - DFS057I block error message
    - format 546
  - multisegment format 546
  - multisegment system message
    - format 546
  - output message default format 546
  - system message format 546
- IMS.FORMAT
  - member selection 426
- IMS.RESLIB 548
- IMSCALL command 301
  - format 301
- IMSCALLRM reply message 336
  - format 336
- IMSQUERY extended function
  - arguments 379
  - usage 379
- IMSRXTRC command 369, 370
- IN
  - predicate 633
- INAIB command object 302
  - format 302
- INCLUDE statement
  - description 646
  - example 647
  - SQLIMSCA
    - COBOL 671
  - SQLIMSDA
    - COBOL 674
- indicator variable
  - description 629
- infinite loop, stopping 391
- INIT (Initialize) call
  - automatic INIT DBQUERY 46
  - call function 245
  - database availability, determining 46
  - enabling data availability, status
    - codes 46
  - enabling deadlock occurrence, status
    - codes 46
  - format 46
  - I/O PCB
    - in INIT call 46
  - INIT (Initialize) call
    - description 46
  - INIT STATUS GROUPA 46
  - INIT STATUS GROUPB 46
  - INIT STATUS RSA12 46
    - parameters 46
    - performance 46
    - restrictions 46
    - status codes 46
  - DB PCB, for 46
- INIT (Initialize) call (*continued*)
  - usage 46
  - using with DBQUERY 46
  - VERSION function 46
- INIT call
  - description 135
  - determining data availability 135
  - format 135
  - parameters 135
  - performance considerations 135
  - summary 81
  - usage 135
- Initialize (INIT) call
  - automatic INIT DBQUERY 46
  - database availability, determining 46
  - description 46
  - enabling data availability, status
    - codes 46
  - enabling deadlock occurrence, status
    - codes 46
  - format 46
  - INIT STATUS GROUPA 46
  - INIT STATUS GROUPB 46
  - INIT STATUS RSA12 46
    - parameters 46
    - performance 46
    - restrictions 46
    - status codes 46
  - using with DBQUERY 46
  - VERSION function 46
- initialize call 135
- input field tab (FTAB)
  - See FTAB= operand (DEV statement) 496
- input host variable 629
- input message
  - field attribute data 494
  - fill characters 495
  - formatting options 484
  - IMS TM password 495
  - input modes 495
  - input substitution character 502
  - literal fields 493
  - MFS formatting of 484
  - nonliteral fields 494
  - with multiple physical pages 501, 536
- input message field
  - defining 495
  - record mode 495
  - stream mode 495
- input message format
  - device-dependent information 446, 452
  - field and segment format 446
  - formatting options, examples 484
- input modes
  - record mode
    - description 495
    - process of record in 503
    - treatment of nulls 498
    - with ISC 503
  - stream mode
    - description 495
    - process of record in 503
    - treatment of nulls 498
    - with ISC 503
- inquiry call 138
- INQY (Inquiry) call
  - format 54
  - INQY (Inquiry) call
    - description 54
  - map of INQY subfunction to PCB
    - type 54
  - parameters 54
  - querying
    - data availability 54
    - environment 54
    - PCB 54
    - program name 54
    - restriction 54
    - return and reason codes 54
    - usage 54
- INQY call
  - description 138
  - format 138
  - querying
    - LERUNOPT, using LERUNOPT subfunction 54
    - summary 81
- INQY call function 245
- INQY DBQUERY 54
- INQY ENVIRON, data output 54
- INQY FIND 54
- INQY PROGRAM 54
- Insert (ISRT) command
  - description 185
  - examples 185
  - format 185
  - insert rules 185
  - options 185
  - restrictions 185
  - usage 185
- insert call 112
- insert rule 649
- INSERT statement
  - description 647
- inserting
  - declaration in a program 646
  - first occurrence of a segment 217
  - last occurrence 218
  - rows in a table 647
  - segments 27
- inserting a segment
  - as first occurrence 217
  - as last occurrence 218
  - in sequence 215
  - path of segments 215
  - root 25
  - rules to obey 25
  - specifying rules 25
- INTEGER
  - data type
    - large 622
    - small 622
- integer constants 627
- interactive SQL 618
- intersystem communication 482
- INTO clause
  - DESCRIBE statement 643
  - FETCH statement 645
  - INSERT statement 649
- INTO DESCRIPTOR clause
  - FETCH statement 645

- iopcbStream data structure 354
  - format 354
- ISC (intersystem communication)
  - ATTACH FM header 467, 529
  - blocking algorithms 529
  - entering and exiting formatted modes 482
  - increasing performance 441
  - input format control
    - input modes 502
  - MFS definitions 444
  - output format control
    - data structure name 535
    - for paging messages 529
    - trailing blank compression 531
  - output modes 529
  - subsystem definition 482
  - use of DPN field 426, 482
  - use of RDPN field 426, 482
- ISRT (Insert) call
  - D command code 215
  - F command code 217
  - format 25
  - ISRT (Insert) call
    - description 25
  - L command code 218
  - loading a database 218
  - parameters 25
  - RULES parameter 217
  - SSA 25
- ISRT (Insert) command
  - description 185
  - examples 185
  - format 185
  - insert rules 185
  - options 185
  - restrictions 185
  - usage 185
- ISRT call 112
  - description 112
  - format 112
  - parameters 112
  - restrictions 112
  - Spool API functions 112
  - summary 82
  - usage 112
- ISRT call function 245
- Issue Command (ICMD) call
  - See ICMD call 45, 132
- issuing
  - a basic checkpoint 165
  - an extended restart 210
- IVPREXX exec 391
- IVPREXX sample application 391

**J**

- Java API specification
  - for IMS Universal drivers 414
- Java reference
  - for IMS solutions for Java development 395
- java.sql.Clob 677
- JCL (job control language),
  - requirements 273, 275

- JDBC
  - methods supported
    - Connection 396
    - XML, extension for 677
  - justification
    - of input messages 484

**K**

- keyboard shortcuts x
- keys
  - concatenated 214
- keyword, SYSSERVE 162

**L**

- L (CALL) statement 237
- LAST insert rule 185
- last inserted sequential dependent
  - segment, retrieving the location of the 192
- legal notices
  - notices 691
  - trademarks 691, 693
- length attribute of column 623
- length field 486
- letter, description in IMS 619
- limiting
  - number of full-function database calls 222
- literal 627
- literal fields
  - input message, default literals 493
  - output message
    - system literals 511
- LOAD command
  - description 190
  - example 190
  - format 190
  - options 190
  - usage 190
- location of a dependent segment,
  - retrieving the 192
- lock
  - during update 667
- lock class and Q command code 222
- LOCKCLASS option 166
- LOG (Log) call
  - format 63
  - LOG (Log) call
    - description 63
  - parameters 63
  - restrictions 63
  - usage 63
- LOG call 149
  - description 149
  - examples 149
  - format 149
  - on LOG I/O area 149
  - parameters 149
  - restrictions 149
  - restrictions on I/O area 149
  - summary 81
  - usage 149
- LOG call function 245

- LOG command
  - example 191
  - format 191
  - LOG command
    - description 191
  - options 191
  - restrictions 191
  - usage 191
- logical operator 634
- logical page advance request 536
- logical page request 535
- logical page. See LPAGELPAGE
  - input 492
- logical unit of work, ending 165, 208
- lowercase character folded to
  - uppercase 619
- LPAGE
  - input, conditional LPAGE
    - selection 476
  - output
    - format 449
  - overview 492

**M**

- M command code
  - examples 229
  - subset pointers, moving forward 229
- MAP definition (MAPDEF) 369, 371
- map name 373
- MAP reading (MAPGET) 369, 373
- MAP writing (MAPPUT) 369, 374
- MAPGET 373
- mapping
  - MAPDEF 371
  - MAPGET 373
  - MAPPUT 374
- MAXQ and Q command code 222
- MDT (modified data tag) 522
- message advance protect 536
- message advance request 536
- message calls
  - call summary 81
- Message Format Buffer Pool 429
- Message Format Service (MFS)
  - 3270 or SLU 2 display devices 425, 440
  - control blocks
    - Finance or SLU P
      - workstations 426
  - output messages
    - format control for 3270P
      - printers 525
    - MFS bypass for SLU 2 (3290) 466
    - specifying descriptor name 463
    - paging action at device 538
    - programmed symbol buffers
      - determining if loaded 442
- message formatting options
  - input
    - description 484
    - examples 484
    - performance factors 439
  - output
    - description 506
    - effects on segments 449
    - performance factors 439

MFLD (message field statement) 484  
 FILL=NULL 492  
 function 484  
 MFS (Message Format Service)  
 control blocks  
 Finance or SLU P  
 workstations 426  
 how input messages are formatted by  
 MFS 484  
 input message  
 formats 484  
 output message  
 field format options 451  
 format control for ISC 531  
 formatting 506  
 modifying EGCS data 461  
 processing output message 505  
 MFS bypass  
 printer byte restriction 464  
 protected and unprotected  
 messages 542  
 specifying for 3270 or SLU 2 464  
 specifying for 3290 with  
 partitioning 465  
 MFS Device Characteristics table  
 (DFSUDT0x), description 548  
 MFS language utility  
 construction of member names 426  
 treatment of EGCS input/output 513  
 MID (message input descriptor)  
 input formatting functions 484  
 relationship to other control  
 blocks 419  
 MOD (message output descriptor)  
 associated MFS functions 506  
 name specification 462  
 relationship to other control  
 blocks 419  
 modified data tag (MDT) 522  
 MONITORRD command  
 format 303  
 MSDB (main storage database)  
 PCBs and DL/I calls 234  
 MULT= operand (DPAGE statement),  
 specifying 476  
 multiple physical pages, input messages  
 description 501  
 terminating (ENDMPPI request) 536  
 multisegment format 546

## N

N command code 219  
 NA 46  
 names, prepared SQL statements 640  
 naming convention  
 SQL 620  
 NEXTLP request  
 description 536  
 operator control table function 535  
 NEXTMSG request  
 description 536  
 NEXTMSGP request  
 description 536  
 NEXTTPP request 536  
 use 536  
 nonexecutable statement 636

nonliteral input fields  
 defining 494  
 NOT FOUND clause of WHENEVER  
 statement 668  
 NU 46  
 null  
 coding in COBOL 449  
 compression  
 example 486  
 specifying 467  
 deleting on input (DPM) 498  
 fill character  
 input message fields 484  
 output device fields 509  
 segment, output 449  
 transmitting to IMS TM 498  
 truncating fields with 506  
 Null command code 226  
 null value  
 assignment 624  
 description 622  
 specified by indicator variable 629  
 NULL= operand (DIV statement)  
 example 498  
 options 498  
 specifying 467  
 numbers in SQL 622  
 numeric  
 data type 622  
 NUMERIC data type  
 description 623

## O

O (OPTION) Statement 266  
 obtaining  
 IMS database statistics 207  
 recent information from the DIB 194  
 status code 193  
 OFTAB= operand (DIV statement),  
 specifying 467  
 OFTAB= operand (DPAGE statement),  
 specifying 476  
 OID 522  
 online performance 439  
 op code 9  
 OPEN  
 statement  
 description 651  
 example 652  
 OPEN (Open) call  
 format 29  
 OPEN (Open) call  
 description 29  
 usage 29  
 Open Transaction Manager Access  
 Callable Interface (C/I) 553  
 otma\_alloc API 559  
 otma\_close API 568  
 otma\_create API 555  
 otma\_free API 568  
 otma\_open API 556  
 otma\_openx API 558  
 otma\_receive\_async API 566  
 otma\_send\_async API 564  
 otma\_send\_receive API 561  
 otma\_send\_receivex API 563

Open Transaction Manager Access  
 (continued)  
 Callable Interface (C/I) (continued)  
 sample programs 569  
 CHNG call 88  
 PURG call 115  
 SETO call 117  
 operand  
 FSA 9  
 operation parameter, SNAP external  
 call 72  
 operator control of MFS 535  
 operator control tables  
 functions  
 ENDMPPI request 536  
 NEXTLP request 536  
 NEXTMSG request 536  
 NEXTMSGP request 536  
 NEXTTPP request 536  
 operator logical paging  
 description 507, 535  
 format design considerations 535  
 in partitioned format mode, 3180 545  
 in partitioned format mode, 3290 543  
 transaction codes and page  
 requests 535  
 OPNQFLRM reply message 337  
 format 337  
 OPNQRY command 304  
 format 304  
 OPNQRYRM reply message 338  
 format 338  
 OPTION statement 266  
 options  
 ACCEPT command 165  
 CHKP (Checkpoint) command 165  
 DEQ (Dequeue) command 166  
 DLET (Delete) command 167  
 GN (Get Next) command 170  
 GNP (Get Next in Parent)  
 command 174  
 ISRT (Insert) command 185  
 LOAD command 190  
 LOCKCLASS 166  
 LOG command 191  
 POS (Position) command 192  
 QUERY command 193  
 REFRESH command 194  
 REPL (Replace) command 195  
 RETRIEVE command 199  
 ROLB (Rollback) command 200  
 ROLL command 201  
 ROLS (Rollback to SETS or SETU)  
 command 202  
 SCHD (Schedule) command 204  
 SETS (Set a Backout Point)  
 command 205  
 SETU (Set a Backout Point  
 Unconditionally) command 206  
 STAT (Statistics) command 207  
 SYMCHKP (Symbolic Checkpoint)  
 command 208  
 TERM (Terminate) command 210  
 XRST (Extended Restart)  
 command 210  
 options list parameter 88  
 CHNG call 88



- options list parameter (*continued*)
  - advanced print function 88
  - APPC 92
  - SETO call 117
  - advanced print function 117
  - APPC 117
- OPTIONS= operand (DIV statement)
  - effects on performance 441
  - specifying 467
  - use 525
  - use with ISC 529
- OR truth table 634
- ordinary identifier in SQL 620
- ORIGIN= operand (DPAGE statement), specifying 476
- OTMA C/I
  - hints and tips 553
- OTMA C/Iwarranty
  - sample programs 569
- otma\_alloc API 559
- otma\_close API 568
- otma\_create API 555
- otma\_free API 568
- otma\_open API 556
- otma\_openx API 558
- otma\_receive\_async API 566
- otma\_send\_async API 564
- otma\_send\_receive API 561
- otma\_send\_receivex API 563
- OUTAIBDBPCB parameter 355
- OUTAIBIOPCB parameter 356
  - format 356
- output device fields
  - dynamic modification 452
  - for cursor positioning 452
- output host variable 629
- output message 482
  - cursor positioning 520
  - default system control area 510
  - device field attributes 511
  - extended field attributes for devices 511
  - extended graphic character set (EGCS) 513
  - fill characters for device fields 509
  - formatting options 506
    - description 506
  - header 482
  - how MFS formats messages 506
  - literal fields 511
  - mixed DBCS/EBCDIC fields 514
  - operator logical paging 507
  - physical paging 508
  - processing 505
  - prompt facility 521
  - system control area (SCA) 510
  - truncation 506
- output message format
  - default 546
  - device-dependent information 448, 452
- overriding
  - FIRST insert rule 218
  - HERE insert rule 217, 218
  - insert rules 25

## P

- P command code 221
- P processing option 215
- page advance request 536
- PAGE= operand (DEV statement)
  - use 522
- PAGEREQ function 535
- paging, operator logical
  - description 535
  - format design considerations 535
  - in partitioned format mode, 3180 545
  - in partitioned format mode, 3290 543
  - transaction codes and page requests 535
- PAGINGOP= operand (PDB statement), use 543
- parameter marker
  - description 654
  - EXECUTE statement 644
  - host variables in dynamic SQL 630
  - PREPARE statement 654
  - rules 654
- parameters
  - BKO execution 200
  - CHKPT=EOV 166
  - RULES 185
- parentage, P command code 221
- PART exec 384
- partition
  - activating 520
  - considerations for defining 431
  - defining 425
  - descriptor (PD) 425
  - descriptor block (PDB) 425
  - initialization options
    - for the 3180 545
    - for the 3290 543
  - uses 431
  - partition set, description 425
- PARTNAME exec 386
- PARTNUM exec 385
- password, IMS
  - description 495
- path call 215
  - D command code 215
- path command 195
- PCB (program communication block)
  - DL/I calls, relationship 234
  - DLIINFO call 369
- PCB (schedule a PSB) call
  - format 65
  - parameters 65
  - PCB (schedule a PSB) call
    - description 65
    - usage 65
- PCBINFORM exec 383
- PCHSEGTS 31
- PCLBSGTS 31
- PCSEGRTS 31
- PD statement (partition definition)
  - use 425
- PD= operand (DPAGE statement), specifying 476
- PDB (partition descriptor block)
  - function 425
  - language statements used to create PD 425

- PDB (partition descriptor block) (*continued*)
  - PAGINGOP= operand 543
- performance factors
  - 3270 or SLU 2 439
  - all devices 439
  - large screen 3270 or SLU 2 devices 440
- period usage 366
- physical page positioning (FIN) 476
- physical paging
  - description 508
  - specifying multiple input pages 476
- PL/I segmentation APIs
  - DFSPWSH include files 595
  - DFSQGETS 595
  - DFSQGETS API 604
  - DFSQSETS 595
  - DFSQSETS API 606
  - DFSXGETS 595
  - DFSXGETS API 609
  - DFSXSETS 595
  - DFSXSETS API 611
  - overview 595
  - return codes 614
- POS (Position) call
  - examples 29
  - format 29
  - I/O area 29
  - parameters 29
  - POS (Position) call
    - description 29
  - unqualified
    - keywords 29
  - usage 29
- POS (Position) command
  - EXEC DLI command format 192
  - format 192
  - options 192
  - POS (Position) command
    - description 192
  - restriction 192
  - usage 192
- POS call function 245
- position
  - establishing in database 19
- Position (POS) command
  - description 192
  - EXEC DLI command format 192
  - format 192
  - options 192
  - restriction 192
  - usage 192
- position in the database, determining the current 199
- precision of numbers
  - description 622
  - determined by SQLIMSLEN
    - variable 673
  - values for data types 622
- predicate
  - basic 631
  - BETWEEN 632
  - description 630
  - IN 633
- PREINIT parameter, input restart 273

PREPARE statement  
 description 653  
 example 654  
 prepared SQL statement  
 dynamically prepared by  
 PREPARE 653  
 executing 643  
 identifying by DECLARE 640  
 obtaining information  
 with DESCRIBE 642  
 SQLIMSDA provides  
 information 671  
 preset destination mode 482  
 print mode 522  
 printed page format control  
 bottom margin 523  
 horizontal tabbing 523  
 left margin position 523  
 line density 523  
 line width 523  
 page depth 523  
 top margin 523  
 vertical tabbing 523  
 PRN= operand (DIV statement),  
 specifying 467  
 processing  
 options  
 P (path) 215  
 program deadlock 46  
 program function keys (3270)  
 literals for master terminal  
 format 547  
 program tab function  
 3270 or SLU 2 509  
 fill character 429  
 programmed symbol  
 buffers 442  
 feature 511  
 solving problems 443  
 programmed symbols (PS)  
 buffers  
 determining if loaded 442  
 loading 442  
 prompt facility for output messages 521  
 protecting the screen  
 PROTECT option 542  
 PRPSQLSTT command  
 format 308  
 PSB (program specification block)  
 in a CICS online program  
 scheduling a 204  
 terminating a 210  
 PSEGHWM 31  
 PT (program tab) function  
 3270 or SLU 2 509  
 fill character 429  
 PUNCH statement 267  
 PURG call 115  
 and OTMA environment 115  
 description 115  
 format 115  
 parameters 115  
 restrictions 115  
 Spool API 115  
 summary 82  
 usage 115  
 PURG call function 245

purge call 115

## Q

Q command code  
 and the DEQ call 222  
 example 222  
 full function and segment  
 release 222  
 lock class 222  
 MAXQ 222  
 QRYDSC reply object 339  
 format 339  
 QRYDTA reply object 340  
 format 340  
 QRYPOPRM reply message 341  
 format 341  
 qualified field names 628  
 QUERY command  
 example 193  
 format 193  
 options 193  
 QUERY command  
 description 193  
 restrictions 193  
 usage 193  
 question mark (?) 644

## R

R command code 230  
 RCDCTL= operand (DIV statement)  
 specifying 467  
 use 525  
 RCMD call 151  
 description 66, 151  
 format 66, 151  
 parameters 66, 151  
 restrictions 66, 151  
 use 66, 151  
 RDBAFLRM reply message 342  
 format 342  
 RDBATHRM reply message 343  
 format 343  
 RDBNACRM reply message 344  
 format 344  
 RDBNAM parameter 357  
 format 357  
 RDBNFNRM reply message 345  
 format 345  
 RDBUPDRM reply message 346  
 format 346  
 RDPN (return destination process name)  
 specifying in MFLD statement 467  
 use on Finance or SLU P  
 workstations 426  
 use with ISC subsystem  
 communication 482  
 RDPN= operand (DIV statement),  
 specifying 467  
 record mode  
 description 495  
 input example 500  
 process of record in 503  
 treatment of nulls 498  
 with ISC 503  
 REFRESH command  
 example 194  
 format 194  
 options 194  
 REFRESH command  
 description 194  
 restrictions 194  
 usage 194  
 releasing  
 a segment 166  
 removing a segment and its  
 dependents 167  
 REPL (Replace) call  
 format 32  
 N command code 219  
 parameters 32  
 REPL (Replace) call  
 description 32  
 SSAs 32  
 usage 32  
 REPL (Replace) command  
 examples 195  
 format 195  
 REPL (Replace) command  
 description 195  
 restrictions 195  
 usage 195  
 REPL call function 245  
 Replace (REPL) command  
 description 195  
 examples 195  
 format 195  
 options 195  
 REPL (Replace) command  
 options 195  
 restrictions 195  
 usage 195  
 replacing a segment 195  
 requesting a catalog record  
 using GUR 21  
 requesting a segment  
 using GU 19  
 resetting a subpointer 231  
 restart call 159  
 Restart, Extended  
 position in database 77  
 Restart, Extended (XRST)  
 description 77  
 with Symbolic Checkpoint (CHKP  
 Symbolic) 39  
 restarting your program  
 XRST call 159  
 restrictions  
 CHKP (Checkpoint) command 165  
 DEQ (Dequeue) command 166  
 DLET (Delete) command 167  
 F command code 217  
 GN (Get Next) command 169  
 GNP (Get Next in Parent)  
 command 174  
 GU (Get Unique) command 180  
 ISRT (Insert) command 185  
 LOG command 191  
 number of database calls and Fast  
 Path 222  
 POS (Position) command 192  
 QUERY command 193

restrictions (*continued*)  
 REFRESH command 194  
 REPL (Replace) command 195  
 RETRIEVE command 199  
 ROLB (Rollback) command 200  
 ROLL command 201  
 ROLS (Rollback to SETS or SETU) command 202  
 SETS (Set a Backout Point) command 205  
 SETU (Set a Backout Point Unconditionally) command 206  
 SYMCHKP (Symbolic Checkpoint) command 208  
 XRST (Extended Restart) command 210

retrieval calls  
 D command code 215  
 F command code 217  
 L command code 218

RETRIEVE command  
 example 199  
 format 199  
 options 199  
 restrictions 199  
 RETRIEVE command description 199  
 usage 199

Retrieve Command (RCMD) call  
 See RCMD call 66, 151

retrieving  
 dependent segments sequentially 174  
 dependents sequentially 16  
 first occurrence of a segment 217  
 last occurrence 218  
 segments  
 Q command code, Fast Path 222  
 Q command code, full function 222  
 sequentially 215  
 segments sequentially 169  
 segments with D 215  
 specific segments 180  
 the location of a dependent segment 192  
 the location of the last inserted sequential dependent segment 192

returning a status code 165

REXX  
 . (period) usage 366  
 calls  
 return codes 364  
 summary 364  
 syntax 364  
 commands  
 DL/I calls 363  
 summary 363  
 DL/I calls, example 364  
 execs  
 DOCMD 387  
 IVPREXX 391  
 PART 384  
 PARTNAME 386  
 PARTNUM 385  
 PCBINFO 383  
 SAY 382  
 IMSRXTRC, trace output 370

REXX (*continued*)  
 issuing synchronous callout requests  
 default output area length 364  
 ICAL 364  
 input area 364  
 output area 364

REXX, IMS adapter  
 . (period) usage 369  
 address environment 361  
 AIB, specifying 364  
 description 359  
 DFSREXX0 program 359, 391  
 DFSREXX1 359  
 DFSREXXU user exit 359  
 DFSRRC00 391  
 diagram 360  
 DL/I parameters 364  
 environment 364  
 example execs 381  
 feedback processing 364  
 I/O area 364  
 installation 359  
 IVPREXX exec 391  
 IVPREXX PSB 361  
 IVPREXX setup 361  
 LLZZ processing 364  
 LNKED requirements 359  
 non-TSO/E 359  
 PCB, specifying 364  
 programs 359  
 PSB requirements 359  
 sample generation 361  
 sample JCL 361  
 SPA processing 364  
 SRRBACK 359  
 SRRCMIT 359  
 SSA, specifying 364  
 SYSEXEC DD 359, 361  
 system environment 359, 361  
 SYSTSIN DD 361  
 SYSTSPRT DD 359, 361  
 TSO environment 359  
 TSO/E restrictions 359  
 ZZ processing 364

REXXIMS commands 371, 373, 379  
 DLIINFO 369  
 IMSRXTRC 369, 370  
 MAPDEF 369  
 MAPGET 369  
 MAPPUT 369, 374  
 SET 369, 375  
 SRRBACK 369, 376  
 SRRCMIT 369, 376  
 STORAGE 369, 377  
 WTL 369, 378  
 WTO 369, 378  
 WTOR 369, 379  
 WTP 369, 378

REXXTDLI commands 363  
 RLSE (Release locks) call summary 1

RLSE (Release Locks) call  
 format 34  
 parameters 34  
 RLSE (Release Locks) call description 34  
 SSAs 34

RLSE (Release Locks) call (*continued*)  
 usage 34  
 RLSE command 311  
 RLSERM reply message 347  
 format 347  
 RMODE 24, AMODE 31, running user modules in 493  
 ROLB (Roll Back) call  
 ROLB (Roll Back) call description 67  
 ROLB (Rollback) command  
 example 200  
 format 200  
 options 200  
 restrictions 200  
 ROLB (Rollback) command description 200  
 usage 200  
 ROLB call 153  
 description 153  
 format 153  
 parameters 153  
 restrictions 153  
 summary 81  
 usage 153  
 ROLB call function 245  
 ROLL (Roll) call  
 DL/I calls, system service ROLL 68  
 ROLL (Roll) call description 68  
 format 68  
 roll back to SETS/SETU call 155  
 ROLL call 154  
 description 154  
 format 154  
 parameters 154  
 restrictions 154  
 summary 81  
 usage 154  
 ROLL call function 245  
 ROLL command  
 example 201  
 format 201  
 options 201  
 restrictions 201  
 ROLL command description 201  
 usage 201  
 Rollback (ROLB) command  
 description 200  
 example 200  
 format 200  
 options 200  
 restrictions 200  
 usage 200  
 rollback call 153  
 Rollback to SETS or SETU (ROLS) command  
 description 202  
 examples 202  
 format 202  
 options 202  
 restrictions 202  
 usage 202  
 ROLS (Roll Back to SETS) call  
 format 68

ROLS (Roll Back to SETS) call *(continued)*  
     parameters 68  
     ROLS (Roll Back to SETS) call  
         description 68  
 ROLS (Rollback to SETS or SETU)  
     command  
         DB PCB  
             specifying 202  
     examples 202  
     format 202  
     options 202  
     restrictions 202  
     ROLS (Rollback to SETS or SETU)  
         command  
             description 202  
             specifying the DB PCB 202  
             usage 202  
 ROLS call 155  
     description 155  
     format 155  
     parameters 155  
     restrictions 155  
     Spool API functions 157  
     summary 81  
     usage 155  
 ROLS call function 245  
 ROLX call function 245  
 row  
     deleting 641  
     inserting 647  
 row-value expression 630  
 RPRN (return primary resource  
 name) 467  
 RPRN= operand (DIV statement),  
 specifying 467  
 RSCLMTRM reply message 348  
     format 348  
 RTRVFLD command object 312  
     format 312  
 RTRVFLDREL Command object  
 (X'CC0B') 312  
 RULES parameter 185  
     FIRST, L command code 218  
     HERE  
         F command code 217  
         L command code 218  
 RULES= 185

## S

S (STATUS) statement 269  
 S command code  
     examples 231  
     subpointer, resetting 231  
 sample  
     code  
         asynchronous processing 581  
         synchronous processing 569  
 sample JCL 273  
 sampleswarranty  
     OTMA C/I 569  
 SAY exec 382  
 SCA (system control area) 510  
     description 510  
     device-dependent information 452  
     specifying 452  
     use 542

scale of numbers  
     description 623  
 SCHD (Schedule) command  
     example 204  
     format 204  
     options 204  
 SCHD (Schedule) command  
     description 204  
     usage 204  
 Schedule (SCHD) command  
     description 204  
     example 204  
     format 204  
     options 204  
     usage 204  
 scheduling a PSB in a CICS online  
 program 204  
 screen formatting  
     3180 433  
     3270 or SLU 2  
         erase all unprotected option 429  
         force format write option 429  
     3290  
         logical units 431  
         partitions 431  
 SCS1 devices  
     DEV statement 467  
 search condition  
     description 634  
     order of evaluation 634  
 SECCHK command 313  
     format 313  
 SECCHKRM reply message 349  
     format 349  
 segment  
     adding one sequentially 190  
     adding to a database 185  
     and its dependents, removing 167  
     releasing a 166  
     replacing 195  
     requesting using GU 19  
     retrieving sequentially 169  
     retrieving specific 180  
     sequential dependent  
         retrieving the location of the last  
         one inserted 192  
 segment edit routine  
     use 493  
 segment format, output message 449  
     restriction 449  
 SEGMLIST Command object  
 (X'CC0A') 314  
 SELECT statement  
     description 655  
     dynamic invocation 637  
 SELECT= operand (DPAGE statement),  
 specifying 476  
 sequence  
     hierarchy 11  
 sequence, indication for statements 273  
 sequential dependent segments  
     retrieving the location of the last one  
     inserted 192  
 sequentially retrieving  
 dependent segments 174  
 segments 169

Set a Backout Point (SETS) command  
     description 205  
     example 205  
     format 205  
     options 205  
     restrictions 205  
     usage 205  
 Set a Backout Point Unconditionally  
 (SETU) command  
     description 206  
     formats 206  
     options 206  
     restrictions 206  
     usage 206  
 set backout point call 157  
 set backout point unconditional call 157  
 SET clause of UPDATE statement 666  
 SET command (REXX) 369, 375  
 set options call 117  
 SET SUBFUNC command (REXX) 375  
 SET ZZ 375  
 SETO call 117  
     and OTMA environment 117  
     description 117  
     format 117  
     parameters 117  
     restrictions 117  
     summary 81  
     usage 117  
 SETO call function 245  
 SETS (Set a Backout Point) call  
     format 70  
     parameters 70  
 SETS (Set a Backout Point) call  
     description 70  
 SETS (Set a Backout Point) command  
     example 205  
     format 205  
     options 205  
     restrictions 205  
 SETS (Set a Backout Point) command  
     description 205  
     usage 205  
 SETS call 157  
     description 157  
     format 157  
     parameters 157  
     restrictions 157  
     Spool API functions 157  
     summary 81  
     usage 157  
 SETS call function 245  
 setting  
     parentage with the P command  
         code 221  
     subset pointer to zero 233  
 setting a backout point  
     DL/I 205  
     unconditionally 206  
 SETU (Set a Backout Point Unconditional)  
 call  
     description 70  
     format 70  
     parameters 70  
 SETU (Set a Backout Point  
 Unconditionally) command  
     example 206

SETU (Set a Backout Point Unconditionally) command *(continued)*  
 examples  
 SETU (Set a Backout Point Unconditionally) command 206  
 formats 206  
 options 206  
 restrictions 206  
 Set a Backout Point Unconditionally (SETU) command  
 example 206  
 SETU (Set a Backout Point Unconditionally) command  
 description 206  
 usage 206  
 SETU call 157  
 description 157  
 restrictions 157  
 Spool API functions 157  
 summary 81  
 shift in (SI) control character 514  
 shift in (SI) framing character 513  
 shift out (SO) control character 514  
 shift out (SO) framing character 513  
 short string column 623  
 SKIP call function 257  
 SLDX= operand (DEV statement),  
 use 523  
 SLU type 2  
 default literal input message  
 fields 493  
 defining IMS TM password 495  
 SNAP call  
 format 71  
 parameters 71  
 SNAP call  
 description 71  
 status codes 71  
 SNAP call function 245  
 SO/SI control characters  
 blank suppress option 514  
 hex representation 514  
 pair verification 514  
 processing by MFS 514  
 use in mixed data field 514  
 SO/SI framing characters 513  
 space character 619  
 special character 619  
 specific segments, retrieving 180  
 Spool API  
 functions 112  
 ISRT call 112  
 STORAGE command example 377  
 SQL (Structured Query Language) *(continued)*  
 assignment operation 624  
 character 619  
 comparison operation 624  
 constants 627  
 data types  
 character strings 623  
 datetime 623  
 description 620  
 numbers 622  
 dynamic 617  
 identifier 619  
 interactive 618  
 naming conventions 620  
 null value 622  
 ordinary identifier 619  
 static 617  
 token 619  
 value 620  
 variable names 620  
 SQL statements  
 CLOSE 639  
 CONTINUE 668  
 DECLARE CURSOR  
 description 639  
 example 640  
 DECLARE STATEMENT 640  
 DELETE  
 description 641  
 example 642  
 DESCRIBE OUTPUT 642  
 EXECUTE 643  
 FETCH  
 description 645  
 example 646  
 INCLUDE  
 description 646  
 example 647  
 SQLIMSCA 671  
 SQLIMSDA 674  
 INSERT  
 description 647  
 invocation 636  
 OPEN  
 description 651  
 example 652  
 operational form 617  
 PREPARE 653  
 SELECT 655  
 UPDATE  
 description 665  
 example 667  
 WHENEVER 668  
 SQLATTR command  
 format 315  
 SQLCARD command  
 format 316  
 SQLCODE  
 +100 651  
 SQLDARD command  
 format 318  
 SQLDTA command  
 format 322  
 SQLERROR  
 clause of WHENEVER statement 668  
 SQLERRRM reply message 350  
 format 350  
 SQLIMSCA (SQL communication area)  
 contents 669  
 entry changed by UPDATE 667  
 INCLUDE statement 647  
 SQLIMSCABC field of SQLIMSCA 669  
 SQLIMSCAID field of SQLIMSCA 669  
 SQLIMSCODE  
 +100 638, 668  
 description 638  
 field of SQLIMSCA 669  
 SQLIMSD field of SQLIMSDA 672  
 SQLIMSDA  
 header 672  
 SQLIMSDA (SQL descriptor area)  
 clause of INCLUDE statement 647  
 contents 671, 672  
 SQLIMSDABC field of SQLIMSDA 672  
 SQLIMSDAID field of SQLIMSDA 672  
 SQLIMSDATA field of SQLIMSDA 673  
 SQLIMSERRD(n) field of  
 SQLIMSCA 669  
 SQLIMSERRM  
 description 638  
 SQLIMSERRMC field of SQLIMSCA 669  
 SQLIMSERRML field of SQLIMSCA 669  
 SQLIMSERRP field of SQLIMSCA 669  
 SQLIMSIND field of SQLIMSDA 673  
 SQLIMSLen field of SQLIMSDA 673  
 SQLIMSN field of SQLIMSDA  
 description 672  
 SQLIMSNNAME field of SQLIMSDA 673  
 SQLIMSTATE  
 '02000' 668  
 description 638  
 field of SQLIMSCA 669  
 SQLIMSTYPE field of SQLIMSDA  
 description 673  
 values 673  
 SQLIMSWARNn field of  
 SQLIMSCA 669  
 SQLSTATE  
 '02000' 651  
 SQLSTT command  
 format 324  
 SQLWARNING clause  
 WHENEVER statement 668  
 SRRBACK command (REXX)  
 description 369  
 format, usage 376  
 SRRCMIT command (REXX)  
 description 369  
 format, usage 376  
 SSA parameter 357  
 SSACOUNT parameter 358  
 SSALIST command object 325  
 format 325  
 SSAs (segment search arguments)  
 GN 11  
 usage 8  
 DLET 8  
 GNP 16  
 ISRT 25  
 REPL 32  
 RLSE 34  
 STACK statement (language utility) 435  
 STAK call function 257  
 standard, SQL (ANSI/ISO)  
 SQL-style comments 619  
 START call function 257  
 starting position in a database,  
 establishing a 180  
 STAT (Statistics) call  
 format 74  
 parameters 74  
 STAT (Statistics) call  
 description 74  
 usage 74

- STAT (Statistics) command
  - example 207
  - format 207
  - options 207
  - STAT (Statistics) command
    - description 207
    - usage 207
- STAT call function 245
- statement
  - naming convention 620
- STATEMENT clause of DECLARE
- STATEMENT statement 641
- static SQL 617
  - description 636
- Statistics (STAT) command
  - description 207
  - example 207
  - format 207
  - options 207
  - usage 207
- statistics, obtaining IMS database 207
- status code
  - GE (segment not found) 215
- status codes
  - GB, end of database 215
  - obtaining 193
  - returning a 165
- STATUS statement 269
- storage
  - !token 377
  - STORAGE command 377
- STORAGE command (REXX)
  - description 369
  - format, usage 377
- stream mode
  - description 495
  - input example 501
  - process of record in 503
  - treatment of nulls 498
  - with ISC 503
- string
  - character 623
  - comparison 626
  - constant 627
  - fixed-length
    - description 623
  - short 623
- structured query language (SQL)
  - result tables 617
- SUB= operand (DEV statement)
  - use 502
- subset pointers
  - M command 229
  - R command code 230
  - resetting 231
  - S command code 231
  - sample application 226
  - Z command code 233
- substitution character 502
- Summary
  - database management call 1
  - system service calls 35
- summary, EXEC DLI commands 163
- Symbolic Checkpoint (CHKP Symbolic)
  - Symbolic) 39
  - format 39
  - parameters 39
- Symbolic Checkpoint (CHKP Symbolic)
  - (continued)
  - restrictions 39
  - usage 39
- Symbolic Checkpoint (SYMCHKP)
  - command
    - description 208
    - example 208
    - format 208
    - options 208
    - restrictions 208
    - usage 208
  - symbolic checkpoint call 127
- Symbolic CHKP call
  - description 127
  - format 127
  - parameters 127
  - restrictions 127
  - summary 81
  - usage 127
- SYMCHKP (Symbolic Checkpoint)
  - command
    - current position 208
    - example 208
    - format 208
    - options 208
    - restrictions 208
  - SYMCHKP (Symbolic Checkpoint)
    - command
      - description 208
      - usage 208
- SYNC (Synchronization Point) call
  - format 76
  - parameters 76
- SYNC (Synchronization Point) call
  - description 76
  - usage 76
- SYNC call
  - description 159
  - format 159
  - parameters 159
  - restrictions 159
  - summary 81
  - usage 159
- SYNC call function 245
- synchronization call 159
- syntax diagram
  - how to read viii
- syntax of EXEC DLI commands 162
- SYSIN input 273
- SYSIN2 input processing 273
- SYSSERVE keyword 162
- system contents directory 131
- system control area 510
- system definition
  - 3270 master terminal format
    - support 547
  - considerations, with MFS 525
- system literals
  - description 511
- system log, writing information to the 191
- system message format,
  - IMS-provided 546
- system service
  - ACCEPT 165
  - CHKP (Checkpoint) 165
- system service (continued)
  - command summary 162
  - DEQ (Dequeue) 166
  - LOAD 190
  - LOG 191
  - QUERY 193
  - REFRESH 194
  - ROLB (Rollback) 200
  - ROLL 201
  - ROLS (Rollback to SETS or SETU) 202
  - SETS (Set a Backout Point) 205
  - SETU (Set a Backout Point Unconditionally) 206
  - STAT (Statistics) 207
  - SYMCHKP (Symbolic Checkpoint) 208
  - XRST (Extended Restart) 210
- system service calls
  - APSB (Allocate PSB) 37
  - CHKP (Basic) 38
  - CHKP (Symbolic) 39
  - DPSB (deallocate) 41
  - GMSG (Get Message) 42
  - ICMD (Issue Command) 45
  - INIT (Initialize) 46
  - INQY (Inquiry) 54
  - LOG (Log) 63
  - PCB (schedule a PSB) 65
  - RCMD (Retrieve Command) 66
  - ROLB (Roll Back) 67
  - SETS/SETU (Set a Backout Point) 70
  - SNAP 71
  - STAT (Statistics) 74
  - SYNC (Synchronization Point) 76
  - TERM (Terminate) 77
  - XRST (Extended Restart) 77

## T

- T (Comment) statement 258
- tabbing
  - control characters 523
  - field tabs 496
  - horizontal 523
  - vertical 523
- table
  - naming convention 620
  - result table 652
- TERM (Terminate) call
  - format 77
  - TERM (Terminate) call
    - description 77
    - usage 77
- TERM (Terminate) command
  - example 210
  - format 210
  - options 210
  - TERM (Terminate) command
    - description 210
    - usage 210
- Terminate (TERM) command
  - description 210
  - example 210
  - format 210
  - options 210
  - usage 210

- terminating a PSB in a CICS online program 210
- test program 235
- time
  - data type 624
- TIME
  - data type
    - description 624
- timestamp
  - data type 624
- TIMESTAMP
  - data type
    - description 624
- trademarks 691, 693
- trailing blank compression 531
- transaction code 535
- translation, character
  - for input messages
    - using XX'3F' 502
  - for output messages
    - device control characters 506
    - SUB= operand (DEV statement) 502
- transmission chains 529
- truncation
  - of input messages 484
  - of output fields 506
- TSO/E REXX 359
- TYPE= operand (DEV statement), specifying 426
- TYPE= operand (DIV statement) specifying 467

## U

- U (Comment) statement 258
- U Command Code 224
- unconditionally setting a backout point 206
- unit of work (UOW)
  - ending a logical 165
- unprotecting the screen
  - UNPROTECT option 542
- unqualified POS call
  - I/O returned area
    - key words 29
    - map of 29
  - keywords 29
- UNSTACK statement (language utility) 435
- UOW (unit of work)
  - ending a logical 165
- UPDATE
  - statement
    - description 665
    - example 667
- update rule 667
- updating
  - rows in a table 665
- UPDCNT parameter 358
- usage
  - ACCEPT command 165
  - CHKP (Checkpoint) command 165
  - DEQ (Dequeue) command 166
  - GN (Get Next) command 169
  - GNP (Get Next in Parent) command 174
  - GU (Get Unique) command 180

- usage (*continued*)
  - ISRT (Insert) command 185
  - LOAD command 190
  - LOG command 191
  - POS (Position) command 192
  - QUERY command 193
  - REFRESH command 194
  - REPL (Replace) command 195
  - RETRIEVE command 199
  - ROLB (Rollback) command 200
  - ROLL command 201
  - ROLS (Rollback to SETS or SETU) command 202
  - SCHD (Schedule) command 204
  - SETS (Set a Backout Point) command 205
  - SETU (Set a Backout Point)
    - Unconditionally command 206
  - STAT (Statistics) command 207
  - SYMCHKP (Symbolic Checkpoint) command 208
  - TERM (Terminate) command 210
  - XRST (Extended Restart) command 210
- USING clause
  - EXECUTE statement 644
  - OPEN statement 651

## V

- V command code 225
- V5SEGRBA 31
- value
  - SQL 620
- VALUES clause
  - INSERT statement 649
- variable
  - description 629
  - host
    - referencing 629
    - SQL syntax 629
  - referencing 629
  - SQL syntax 629
  - substitution for parameter markers 644
- VERSION
  - function of INIT call 46
- version identification
  - description 535
  - for DPM formats 550
  - for SLU P 483
- VSAM, STAT call 75
- VT= operand (DEV statement)
  - use 523
- VTAB= operand (DEV statement)
  - use 523

## W

- W command code
  - examples 232
- WAITAOI 42
- WHENEVER statement
  - description 668
  - example 669

- WIDTH= operand (DEV statement)
  - use 523
- writing information to the system
  - log 191
- WTL command (REXX)
  - description 369
  - format, usage 378
- WTO command (REXX)
  - description 369
  - format, usage 378
- WTO statement 272
- WTOR command (REXX)
  - description 369
  - format, usage 379
- WTOR statement 273
- WTP command (REXX)
  - description 369
  - format, usage 378

## X

- XML (Extensible Markup Language)
  - JDBC extensions for 677
  - retrieveXML 677
    - Clob interface 677
    - example 677
  - storeXML 679
    - example 679
    - SQL syntax 679
  - UDFs 677
- XQuery
  - function and operator extensions for IMS
    - ims:and 684
    - ims:eq 681
    - ims:ge 682
    - ims:gn 681
    - ims:gt 682
    - ims:le 683
    - ims:lt 682
    - ims:ne 683
    - ims:or 684
    - ims:particle 680
  - standard features in IMS 685
- XQuery support
  - functions and operators supported 686
- XRF (Extended Recovery Facility)
  - message format after takeover 522
- XRST (Extended Restart) 39
- XRST (Extended Restart) call
  - XRST (Extended Restart) call description 77
- XRST (Extended Restart) command
  - example 210
  - format 210
  - options 210
  - restrictions 210
  - usage 210
  - XRST (Extended Restart) command description 210
- XRST call 159
  - description 159
  - format 159
  - parameters 159
  - restrictions 159
  - summary 81

XRST call (*continued*)  
usage 159  
XRST call function 245

## **Z**

Z command code  
examples 233  
setting a subpointer to zero 233  
z/OS environment 361







Product Number: 5635-A04  
5655-DSM  
5655-TM2

Printed in USA

SC19-3647-04



Spine information:

IMS Version 13

Application Programming APIs

