

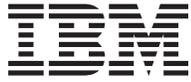
iSeries

CICS for iSeries Application Programming Guide

Version 5

SC41-5454-02





@server

iSeries

CICS for iSeries Application
Programming Guide

Version 5

SC41-5454-02

Note

Before using this information and the product it supports, be sure to read the information in Appendix F, "Notices," on page 585.

Third Edition (April 2004)

- | This edition applies to version 5, release 3, modification 0 of IBM CICS Transaction Server for iSeries (product number 5722-DFH) and to all subsequent releases and modifications until otherwise indicated in new editions. This version does not run on all reduced instruction set computer (RISC) models nor does it run on CISC models.
- | This edition replaces SC41-5454-01.

© Copyright International Business Machines Corporation 1998, 2004. All rights reserved.

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

Contents

Figures ix

Tables xi

About CICS® for iSeries Application Programming Guide (SC41-5454) xiii

Who should read this book xiii

What you need to know to understand this book xiii

Conventions and terminology used in this book xiii

Prerequisite and related information xiv

 CICS/400 library xiv

 Books from related libraries. xiv

How to send your comments xv

Part 1. Introduction 1

Chapter 1. Introducing CICS for iSeries application programming 3

If CICS is new to you 3

What's different about CICS programs? 3

Benefits of CICS for iSeries to CICS programmers 3

Chapter 2. Portability and migration considerations 5

Migrating from another CICS platform 5

 CICS/400 5

 BMS 6

 Terminal Control 6

 COBOL 6

 ILE C. 9

 SQL 9

 iSeries 9

 Source code. 9

Migrating from another CICS/400 release 10

 BMS. 10

 Application programs 10

 Resource definitions 10

Chapter 3. Preparing and writing CICS applications in COBOL 11

Preparing a COBOL application 11

 Coding CICS statements in COBOL applications 12

 Preprocessing. 13

 Translating a COBOL program 13

 Compiling an application program 16

Writing CICS programs in COBOL 17

 Modular programming 18

 Pointer-based addressing 18

 Getting map set storage 19

 Source code considerations 20

Calling programs from COBOL. 22

 Using CICS commands 22

 Using COBOL CALL statements 23

 Rules governing calling CICS COBOL programs 24

 Program activation 25

Sample application programs 27

 Data declarations used by the ACCT sample 28

 Defining resources for the ACCT sample 29

 Running the ACCT sample 29

 Displaying an account record 29

 Adding an account record 30

 Searching by account holder's name 31

 Modifying an account record 31

 Deleting an account record 32

 Printing an account record 33

Chapter 4. Preparing and writing CICS applications in ILE C. 35

Preparing an ILE C application. 35

 Coding CICS statements in iSeries applications 37

 Preprocessing. 38

 Translating an ILE C program 38

 Compiling an application program 40

Writing CICS programs in ILE C 41

 Modular programming 42

 Pointer-based addressing 42

 Getting map set storage 44

 Passing arguments by value 44

 Exception handling. 46

 Data declarations needed for ILE C 47

 Naming EIB fields 47

 Source code considerations 47

Calling programs and ILE procedures from ILE C 48

 Using EXEC CICS commands 48

 Using C language calls 49

 Rules governing calling CICS ILE C programs. 50

 Program activation 51

Sample application programs 53

 Data declarations used by the FILEA sample 55

 Defining resources for the FILEA sample 55

 Running the FILEA sample 56

Part 2. Application design 59

Chapter 5. Designing efficient applications 61

Program size and structure 61

Choosing between pseudoconversational and conversational design 61

 General programming techniques 63

 Processor usage 64

 Recovery design implications 64

 Terminal interruptibility 66

 Summary of pseudoconversational and conversational design 66

Using resources effectively 66

 Processor storage 66

Processor time	67
Exclusive-use resources	67
Line transmission capacity	67
Other suggestions	67
Auxiliary trace	67
Unnecessary commands	68
Resource retention	68
Data definition and manipulation considerations	68
Storing data within a transaction	68
Transaction work area (TWA)	69
User storage	69
COMMAREA in EXEC CICS LINK and EXEC	
CICS XCTL commands	70
Program storage	70
Sharing data across transactions	70
Common work area (CWA)	71
TCTTE user area (TCTUA)	71
COMMAREA in EXEC CICS RETURN	
commands	72
Display screen	72
Temporary storage	73
Intrapartition transient data	74
Your own files	74
Data operations	75
Emulating VSAM files	75
Browsing files	76
Logging files	76
Sequential file access	76
Terminal operations	77
Data stream considerations	77
BMS considerations	77
Additional terminal control considerations	80
Performance considerations	81
CICS and multiprocessor AS/400s	81
CICS SIT parameters	81
COBOL application code	82
ILE C application code	83
*DEBUG or *NODEBUG	83
EXEC CICS LINK command or host language	
call	83
Terminal communication	84

Chapter 6. Dealing with exception conditions 87

Programs in any supported language	87
How to use the RESP and RESP2 options	87
How to use the NOHANDLE option	90
COBOL programs only	91
How to use the EXEC CICS IGNORE	
CONDITION command	91
Passing control to a specified label	92
Relying on the system default action	95
Mixing the methods	98
How CICS keeps track of what to do	99

Chapter 7. Testing your application 101

Testing applications	101
Screen usage, checks and considerations	101
Types of problems	102
Levels of testing	103

Finding a problem in application code on a production system	103
--	-----

Chapter 8. Recovery considerations 105

CICS and OS/400 commitment control recovery	105
Defining recoverable files to CICS (an overview)	105
Syncpointing	106
User journaling	106
Journal records	106
Journal output synchronization	107

Chapter 9. Abnormal termination recovery 109

Creating a program-level abend exit	110
Restrictions on retrying operations	110
Trace	111
Trace entry points	112
Dump	112

Part 3. Files and databases 113

Chapter 10. File control 115

Emulated VSAM files	115
Key-sequenced file (KSDS)	116
Entry-sequenced file (ESDS)	116
Relative record file (RRDS)	116
VSAM-like logical views	116
Reading records	117
Direct reading (using EXEC CICS READ)	118
Sequential reading (browsing)	119
Skip-sequential processing	121
Updating records	122
Specifying record length	122
Deleting records	123
Deleting groups of records (generic delete)	123
Adding records	123
Adding to a KSDS	123
Adding to an ESDS	124
Adding to an RRDS	124
Specifying record length	124
Review of file control command options	124
The RIDFLD option	124
The INTO and SET options	125
The FROM option	125
Avoiding transaction deadlocks	126
KEYLENGTH option for remote files	127
Record identification	127
Identifying records by key	127
Relative byte address (RBA) and relative record number (RRN)	128
CICS locking of emulated VSAM records in recoverable files	128

Part 4. Data communication 131

Chapter 11. Introduction to data communication 133

Chapter 12. Introduction to basic mapping support (BMS) 135

How BMS affects programming 135
BMS maps 136
 BMS map definition 136
 Creating BMS map sets 137
 Cataloging BMS map sets 137
BMS commands 138
Level of BMS 139
 Base and towers architecture 139
Summary of support for CICS/400 BMS 139

Chapter 13. CICS/400 basic mapping support (BMS) 141

Information display systems 141
 IBM 3270 Information Display System 141
 IBM 5250 Information Display System 141
 Input operations 141
 Output operations 143
 Display field concepts 143
 Attribute character 144
Screen layout design 146
 Screen sizes 147
Defining BMS maps 147
 Defining a map set 147
 Defining maps within a map set 147
 Defining fields within a BMS map 148
 Terminating a map set definition 148
Creating BMS maps 148
 Symbolic description map 148
 Physical map 149
 Map set suffixing 149
Writing programs to use BMS services 151
 Copying symbolic description maps 152
 Data structures 152
 Sending data to a display device 155
 Cursor positioning 159
 Accessing data outside the program 159
 Receiving data from a display 160
 Responding to terminal input 162
Text processing 164
 Display characters in text 164
 Control characters in text 165
 Character attribute control (3270 devices only) 165
 Graphic data fields 166
Printed output 167
 Using the hardware print key 167
 Using asynchronous page build transaction 167
 Printer formatting considerations 168

Chapter 14. Terminal control 169

Terminal-oriented task identification 170
Logical unit communication protocol 170
 Send/receive mode 170
 Send/receive protocol (INVITE option) 171
 Chaining the input data 171
 Chaining the output data 171
 Response protocol 172
 Preventing interruptions (bracket protocol) 172

Handling attention identifiers (EXEC CICS HANDLE AID) 173
OS/400 display data streams 174
Terminal control and DBCS 174

Chapter 15. Intercommunication considerations 175

Design considerations 175
Transaction routing 175
Function shipping 176
Distributed program link (DPL) 176
 Using the distributed program link function 177
 Examples of distributed program link 178
 Programming considerations for distributed program link 182
Asynchronous processing 185
Distributed transaction processing (DTP) 186
 Common Programming Interface Communications (CPI Communications) 186

Part 5. CICS management functions 187

Chapter 16. Control region. 189

Chapter 17. Application shell. 191

Chapter 18. Interval control 193

Timer-related tasks 194
Expiration times 194
Request identifiers 196

Chapter 19. Task control 197

Chapter 20. Program control 199

Defining and using CICS tables 199
Application program logical levels 200
Link to another program expecting return 200
Passing data to other programs 201
 COMMAREA 201
 INPUTMSG 203
 Using the INPUTMSG option on the EXEC CICS RETURN command 205
 Other ways of passing data 205
 Examples of passing data 205

Chapter 21. Access to system information 211

System programming commands 211
EXEC interface block (EIB) 211

Chapter 22. Storage control 213

Chapter 23. Transient data control 215

Intrapartition destinations 215
Extrapartition destinations 215
Indirect destinations 216

Automatic transaction initiation (ATI) 216

Chapter 24. Temporary storage control 219

Temporary storage queues 219
Temporary storage commands 220
Typical uses of temporary storage control 220

Chapter 25. Printer spooling 223

When are printer spooling files closed? 223

Part 6. Supplied transactions 225

Chapter 26. Introduction to CICS-supplied transactions 227

Chapter 27. Execution diagnostic facility (EDF). 229

Getting started 229
 Restrictions when using EDF 229
Where does EDF intercept the program? 230
What does EDF display? 231
 The header 231
 The body 231
 How you can intervene in program execution 234
 EDF menu functions 235
How to use EDF 238
 Using EDF in single-screen mode 238
 Using EDF in dual-screen mode 240
 Stopping EDF 240
 Overtyping to make changes 240

Chapter 28. Temporary storage browse (CEBR) 243

How to use the CEBR transaction 243
What does the CEBR transaction display? 244
 The header 244
 The command area 244
 The body 244
 The message line 245
 The CEBR options on function keys 245
The CEBR commands 246
Using the CEBR transaction with transient data 248
Security considerations 248

Chapter 29. Command-level interpreter (CECI). 249

How to use CECI 249
What does CECI display? 250
 The command line 250
 The status line 251
 The body 253
 The message line 254
 CECI options on function keys 254
Additional displays 255
 Expanded area 255
 Variables 255
 The EXEC interface block (EIB) 257

 Error messages display 257
Making changes 258
How CECI runs 258
 CECI sessions 258
 Abends 259
 Exception conditions 259
 Program control commands 259
 Terminal Sharing 259
 Saving commands 260
Security considerations 261

Part 7. Programming reference 263

Chapter 30. OS/400 control language (CL) commands 265

Interpreting the syntax diagrams 265
CRTCICSCBL 266
CRTCICSC 286
CRTCICSMAP 301

Chapter 31. Programming reference 305

Introduction to EXEC CICS commands 305
Command format 305
CICS syntax notation used 306
Argument values 307
 COBOL argument values 308
 ILE C argument values 309
CICS-value data areas (CVDAs) 309
DATASET option 311
INTO and SET options 311
LENGTH options 312
NOHANDLE option 312
RESP and RESP2 options 313
System programming commands 314
 INQUIRE and SET commands 315
 PERFORM command 318
 DISCARD commands 318
Commands by function 319
 Abend support 319
 APPC mapped conversation 319
 BMS 319
 Built-in function 319
 Diagnostic services 319
 Environment services 319
 Exception support 320
 File control 320
 Interval control 321
 Journaling 321
 Printer spooling 321
 Program control 321
 Storage control 321
 Syncpoint 321
 Task control 321
 Temporary storage control 321
 Terminal control 322
 Transient data control 322

Chapter 32. Application programming commands - reference 323

ABEND	323
ADDRESS	324
ALLOCATE	325
ASKTIME	326
ASSIGN	327
BIF DEEDIT	332
CANCEL	333
CONNECT PROCESS	335
CONVERSE (APPC)	337
CONVERSE (5250 or 3270 logical)	339
DELAY	341
DELETE	344
DELETEQ TD	348
DELETEQ TS	349
DEQ	350
DUMP TRANSACTION	352
ENDBR	353
ENQ	355
ENTER TRACENUM	357
EXTRACT ATTRIBUTES (APPC)	359
EXTRACT PROCESS	360
FORMATIME	361
FREE (APPC)	364
FREEMAIN	365
GETMAIN	367
HANDLE ABEND	369
HANDLE AID	371
HANDLE CONDITION	372
IGNORE CONDITION	373
ISSUE ABEND	374
ISSUE CONFIRMATION	375
ISSUE ERASEAUP	376
ISSUE ERROR	377
ISSUE PREPARE	378
ISSUE SIGNAL (APPC)	379
LINK	380
LOAD	385
POP HANDLE	386
POST	387
PUSH HANDLE	389
READ	390
READNEXT	395
READPREV	400
READQ TD	404
READQ TS	407
RECEIVE (APPC)	410
RECEIVE (5250 or 3270 logical)	412
RECEIVE MAP	415
RELEASE	417
RESETBR	418
RETRIEVE	421
RETURN	424
REWRITE	427
SEND (APPC)	430
SEND (SCS)	432
SEND (5250 or 3270 logical)	433
SEND CONTROL	435
SEND MAP	436
SEND TEXT	439
SPOOLCLOSE	441
SPOOLOPEN OUTPUT	442

SPOOLWRITE	444
START	445
STARTBR	452
SUSPEND	457
SYNCPOINT	457
SYNCPOINT ROLLBACK	458
UNLOCK	458
WAIT CONVID	461
WAIT EVENT	461
WAIT JOURNALNUM	462
WRITE	463
WRITE JOURNALNUM	467
WRITEQ TD	469
WRITEQ TS	471
XCTL	474

Chapter 33. System programming reference 477

DISCARD commands	477
DISCARD AUTINSTMODEL	477
DISCARD FILE	478
DISCARD PROGRAM	478
DISCARD TRANSACTION	479
INQUIRE commands	480
INQUIRE AUTINSTMODEL	480
INQUIRE AUTINSTMODEL (browse)	480
INQUIRE CONNECTION	481
INQUIRE CONNECTION (browse)	483
INQUIRE FILE	484
INQUIRE FILE (browse)	487
INQUIRE JOURNALNUM	488
INQUIRE JOURNALNUM (browse)	489
INQUIRE PROGRAM	490
INQUIRE PROGRAM (browse)	492
INQUIRE SYSTEM	493
INQUIRE TASK	494
INQUIRE TDQUEUE	496
INQUIRE TDQUEUE (browse)	499
INQUIRE TERMINAL or NETNAME	500
INQUIRE TERMINAL (browse)	504
INQUIRE TRACEDEST	505
INQUIRE TRANSACTION	506
INQUIRE TRANSACTION (browse)	508
PERFORM SHUTDOWN command	509
SET commands	509
SET CONNECTION	509
SET FILE	511
SET JOURNALNUM	514
SET PROGRAM	515
SET SYSTEM	517
SET TASK	517
SET TDQUEUE	518
SET TERMINAL	520
SET TRACEDEST	522
SET TRANSACTION	524

Part 8. Appendixes 527

Appendix A. EXEC interface block	529
EIB fields	529

Appendix B. BMS-related constants 545

Field attribute and printer control characters . . . 545
Attention identifier constants, DFHAID . . . 548

Appendix C. Terminal control 549

Commands and options for terminals and logical units 549
 Fullword lengths 549
 Read from terminal or logical unit (EXEC CICS RECEIVE) 549
 Write to terminal or logical unit (EXEC CICS SEND). 550
 Converse with terminal or logical unit (EXEC CICS CONVERSE). 550
Display device operations 550
 Erase all unprotected fields (EXEC CICS ISSUE ERASEAUP). 551
 Input operation without data (EXEC CICS RECEIVE) 551

Appendix D. BMS macro summary 553

Defining map sets, maps, and fields 553
 Map set definition macro (DFHMSD) 553
 Map definition macro (DFHMDI). 553

 Field definition macro (DFHMDF) 553
 Ending a map set definition 553
Defining field groups. 553
DFHMSD. 555
DFHMDI. 562
DFHMDF 566
Sample map with DBCS data definitions 576

Appendix E. CICS-value data areas supported by CICS/400 579

CVDAs by symbolic name 579
CVDAs by numeric value 581
CVDAs returned by the INQUIRE
 TERMINAL|NETNAME DEVICE command . . . 583

Appendix F. Notices 585

Programming Interface Information 586
Trademarks 586

Glossary 589

Index 599

Figures

1. Preparing a COBOL application program	12	31. Setting output map data structure to nulls	154
2. Screen showing an example of using source type CICSCBL	17	32. Modifying map field attributes	155
3. Example of an SEU screen showing code containing CICS commands	18	33. Illustration of distributed program link	177
4. Example of pointer-based addressing in a COBOL program	19	34. COBOL example of a distributed program link	178
5. Example of processing BMS maps in a COBOL program	20	35. Using distributed program link with the SYNCONRETURN option	181
6. Control is returned to the next higher logical level	23	36. Using distributed program link without the SYNCONRETURN option	182
7. Flow of control between COBOL programs and run units in CICS/400	27	37. Example of mixing DPL and DTP	183
8. ACCT sample screen: Working with the ACCT samples	28	38. API commands prohibited in programs invoked by distributed program link	185
9. ACCT sample screen: Menu	29	39. Application program logical levels	201
10. ACCT sample screen: Displaying a record	30	40. Use of INPUTMSG in a linked-to chain	204
11. ACCT sample screen: Adding a new record	31	41. COBOL example-EXEC CICS LINK command	206
12. ACCT sample screen: Searching by account holder's name	31	42. ILE C example-EXEC CICS LINK command	207
13. ACCT sample screen: Modifying a record	32	43. COBOL example-EXEC CICS RETURN command	208
14. ACCT sample screen: Deleting a record	33	44. ILE C example-EXEC CICS RETURN command	209
15. Preparing an ILE C application program	37	45. Typical EDF display	231
16. An example of using source type QCSRC	41	46. Typical EDF display at program initiation	232
17. Example of using pointer-based addressing in a ILE C program	43	47. Typical EDF display at start of execution of a CICS command	232
18. Example of processing BMS maps in an ILE C program	45	48. Typical EDF display at completion of a CICS command	233
19. Control is returned to the next higher logical level	48	49. Typical EDF display at program termination	233
20. Flow of control between ILE C programs and activation groups in CICS	53	50. Typical EDF display at task termination	233
21. Screen showing the members of file QCSRC containing the FILEA application programs	54	51. Typical EDF display when an abend occurs	234
22. Screen showing the members of file QMAPSRC containing FILEA sample maps	55	52. Typical EDF display at abnormal task termination	234
23. Data stream conversion	84	53. Typical EDF display for STOP CONDITIONS	237
24. An extract from COBOL program ACCT01	89	54. Typical CEBR display of temporary storage queue contents	243
25. Trapping the unexpected with the EXEC CICS HANDLE CONDITION ERROR command	94	55. Typical CEBR display of default temporary storage queue	244
26. Using EXEC CICS PUSH HANDLE and EXEC CICS POP HANDLE commands	96	56. Typical CECI display for command syntax check	250
27. How CICS selects whether to take the system default action	98	57. Typical CECI display for about to start command	252
28. ABEND exit processing	111	58. Typical CECI display for command completed	253
29. BMS map set suffixing logic	150	59. Typical CECI display of variables associated with CECI session	255
30. Some suffixes and subfields	153	60. Typical CECI display of the EIB	257
		61. Typical CECI display of the message display	257
		62. Sample map with DBCS data definitions	577

Tables

I	1. COBOL compiler limits	21	10. ADDCICSPPT and CHGCICSPPT CL command parameters supporting DPL . . .	178
	2. Rules to be used with CICS COBOL programs	24	11. Language identifiers	283
	3. Rules for passing values as arguments in EXEC CICS commands.	46	12. Command syntax conventions	306
	4. Rules to be used with ILE C programs . . .	50	13. Standard attribute and printer control character list, DFHBMSCA	545
	5. Requests that require exclusive use and when the reservation terminates.	75	14. Bit map for attributes	547
	6. Commands that hold position and when the hold is released	75	15. Standard attention identifier constants list, DFHAID	548
	7. Correspondence between 5250 and 3270 AIDs	143	16. BMS terminal types	560
	8. CICS/400 LU protocol options.	170		
	9. Options on EXEC CICS LINK command supporting DPL.	178		

About CICS® for iSeries Application Programming Guide (SC41-5454)

This book contains two types of information. Parts 1-6 of this book give *guidance* in using the CICS/400 application programming interface; they are complemented by the *reference* information in Part 7, “Programming reference,” on page 263.

Who should read this book

This book is for CICS application programmers. If you are an experienced programmer, you do not need to read all of the information in the guidance section of this manual (parts 1-6). If you are less experienced, you should find the guidance information helpful. If you are new to CICS programming, you should consider reading the *CICS Application Programming Primer*, SC33-0674, which will give you a basic introduction to CICS application programming. The primer is primarily for mainframe CICS users, but most of the content of the book also applies to CICS/400 application programming. Also see Chapter 1, “Introducing CICS for iSeries application programming,” on page 3 of this book to determine what material you should become familiar with.

What you need to know to understand this book

You must be able to program in either COBOL/400, ILE COBOL or ILE C. You need general knowledge about CICS and the general concepts of the Integrated Language Environment® (ILE).

Conventions and terminology used in this book

- CICS refers to CICS/400 unless otherwise stated.
- API refers to the CICS command-level application programming interface.
- COBOL refers to COBOL/400 or ILE COBOL unless specifically stated otherwise.
- C refers to ILE C unless specifically stated otherwise.
- As a general rule, any files, parameters, and messages whose names start with the letters “AEG” or “DFH” belong to CICS.
- Any item whose name begins with an asterisk and is spelled in capitals is a special name recognized by the iSeries, either an *object type* (see Glossary), such as *PRTF for printer file, or a *special value* that can be assumed by an option of a command. For example: *NOSOURCE is a special value of the CICSOPT options on the CRTICSCBL (create CICS COBOL) CL command.
- The term **mainframe** is used to refer to CICS products that run on computers of the S/370™, System/390®, or zSeries® family. (Not all of these products run on all of these computers, for example, CICS Transaction Server for ZOS Version 2 does not run on System/370™).
 - CICS Transaction Server for z/OS® Version 2, program number 5697-E93
 - CICS Transaction Server for OS/390® Version 1, program number 5655-147
 - CICS/ESA® Version 4, program number 5655-018
 - CICS Transaction Server for VSE/ESA™, program number 5648-054
 - CICS/VSE Version 2, program number 5686-026

Prerequisite and related information

CICS/400 library

These books form the CICS/400 library that is delivered with the product:

CICS for iSeries Administration and Operations Guide, SC41-5455-00

This guide gives introductory information about CICS/400. It then provides information about system and resource definition, setup of a system, and operator commands.

CICS for iSeries Application Programming Guide, SC41-5454-02

This manual provides programming guidance information, in narrative form with examples. This is followed by the reference section describing the syntax and use of each command.

CICS for iSeries Intercommunication, SC41-5456-00

This manual describes the CICS/400 side of communication between CICS systems running on different platforms. There is a similar manual for each CICS platform.

CICS for iSeries Problem Determination, SC41-5453-00

This manual provides guidance in problem determination for users of CICS/400.

CICS Family: Interproduct Communication, SC34-6267-00

This manual, which is also part of the libraries of the other CICS family members, gives an overview of communication between CICS systems running on different platforms.

CICS Family: API Structure, SC33-1007-02

This manual, which is also part of the libraries of the other CICS family members, gives a quick reference to the level of support that each member of the CICS family gives to the CICS application programming interface. It is designed for customers and software vendors developing applications able to run on more than one CICS platform and porting applications from one platform to another.

Books from related libraries

Other CICS books

CICS Application Programming Primer, SC33-0674-01

Compilers

ILE C:

ILE Concepts, SC41-5606-07

WebSphere Development Studio: ILE C/C++ Language Reference, SC09-7852-00

WebSphere Development Studio: ILE C/C++ Programmer's Guide, SC09-2712-04

ILE COBOL:

ILE Concepts, SC41-5606-07

WebSphere Development Studio: ILE COBOL Reference, SC09-2539-04

WebSphere Development Studio: ILE COBOL Programmer's Guide, SC09-2540-04

COBOL/400®:

COBOL/400 User's Guide, SC09-1812-00

COBOL/400 Reference, SC09-1813-00
SAA® CPI COBOL Reference, SC26-4354-02

SQL

Refer to the Database topic and File and file systems topic in the iSeries Information Center.

System Programming Support, Control language (CL)

CL Programming, SC41-5721-06

Refer to the Programming topic in the iSeries Information Center.

Common user access

* (CUA®*)

SAA CUA Basic Interface Design Guide, SC26-4583-00

Miscellaneous books

3270 Device Emulation Support, SC41-5408-00

Refer to the Database topic and File and file systems topic in the iSeries Information Center.

Backup and Recovery, SC41-5304-07

Performance Tools for iSeries, SC41-5340-01

Install, upgrade, or delete OS/400 and related software, SC41-5120-07

Refer to the Programming APIs topic in the iSeries™ Information Center.

iSeries Security Reference, SC41-5302-07

3270 Data Stream Programmer's Reference, GA23-0059-07

Use the iSeries Information Center as your starting point for looking up iSeries technical information. You can access the Information Center two ways:

- From the following Web site:

<http://www.ibm.com/eserver/iseries/infocenter>

- From the *iSeries Information Center*, SK3T-4091-04 CD-ROM. This CD-ROM ships with your new iSeries hardware or IBM Operating System/400 software upgrade order. You can also order the CD-ROM from the IBM® Publications Center:

<http://www.ibm.com/shop/publications/order>

The iSeries Information Center contains new and updated iSeries information such as software and hardware installation, Linux, WebSphere®, Java™, high availability, database, logical partitions, CL commands, and system application programming interfaces (APIs). In addition, it provides advisors and finders to assist in planning, troubleshooting, and configuring your iSeries hardware and software.

With every new hardware order, you receive the *iSeries Setup and Operations CD-ROM*, SK3T-4098-02. This CD-ROM contains IBM @server IBM e(logo)server iSeries Access for Windows and the EZ-Setup wizard. iSeries Access Family offers a powerful set of client and server capabilities for connecting PCs to iSeries servers. The EZ-Setup wizard automates many of the iSeries setup tasks.

How to send your comments

Your feedback is important in helping to provide the most accurate and high-quality information. If you have any comments about this book or any other iSeries documentation, fill out the readers' comment form at the back of this book.

- If you prefer to send comments by mail, use the readers' comment form with the address that is printed on the back. If you are mailing a readers' comment form from a country or region other than the United States, you can give the form to the local IBM branch office or IBM representative for postage-paid mailing.
- If you prefer to send comments by FAX, use either of the following numbers:
 - United States, Canada, and Puerto Rico: 1-800-937-3430
 - Other countries or regions: 1-507-253-5192
- If you prefer to send comments electronically, use one of these e-mail addresses:
 - Comments on books:
RCHCLERK@us.ibm.com
 - Comments on the iSeries Information Center:
RCHINFOC@us.ibm.com

Be sure to include the following:

- The name of the book or iSeries Information Center topic.
- The publication number of a book.
- The page number or topic of a book to which your comment applies.

Part 1. Introduction

Chapter 1. Introducing CICS for iSeries application programming	3
If CICS is new to you	3
What's different about CICS programs?	3
Benefits of CICS for iSeries to CICS programmers	3

Chapter 2. Portability and migration considerations	5
Migrating from another CICS platform	5
CICS/400	5
BMS	6
Terminal Control	6
COBOL	6
ILE C.	9
SQL	9
iSeries	9
Source code.	9
Migrating from another CICS/400 release	10
BMS.	10
Application programs	10
Resource definitions	10

Chapter 3. Preparing and writing CICS applications in COBOL	11
Preparing a COBOL application	11
Coding CICS statements in COBOL applications	12
Preprocessing.	13
Translating a COBOL program	13
Characteristics of the input source file	14
CCSID of source files	15
Output from the translator	15
Compiling an application program	16
Writing CICS programs in COBOL	17
Modular programming	18
Pointer-based addressing	18
Example of using pointer variables	19
Getting map set storage	19
Source code considerations	20
Calling programs from COBOL.	22
Using CICS commands	22
Using COBOL CALL statements	23
Static COBOL call	23
Dynamic COBOL call	24
Rules governing calling CICS COBOL programs	24
Program activation	25
Sample application programs	27
Data declarations used by the ACCT sample	28
Defining resources for the ACCT sample	29
Running the ACCT sample	29
Displaying an account record	29
Adding an account record	30
Searching by account holder's name	31
Modifying an account record	31
Deleting an account record	32
Printing an account record	33

Chapter 4. Preparing and writing CICS applications in ILE C.	35
Preparing an ILE C application.	35
Coding CICS statements in iSeries applications	37
Preprocessing.	38
Translating an ILE C program	38
Characteristics of the input source file	39
Output from the translator	39
Compiling an application program	40
Writing CICS programs in ILE C	41
Modular programming	42
Use of #include	42
Use of modules	42
Pointer-based addressing	42
Example of using pointer variables	42
EXEC CICS ADDRESS EIB	42
EXEC CICS ADDRESS COMMAREA.	43
EXEC CICS READ/REWRITE	43
Getting map set storage	44
Passing arguments by value	44
Exception handling.	46
Data declarations needed for ILE C	47
Naming EIB fields	47
Data types.	47
Source code considerations	47
Calling programs and ILE procedures from ILE C	48
Using EXEC CICS commands	48
Using C language calls	49
Dynamic program calls	49
Calling procedures	49
Procedure pointer calls	50
Rules governing calling CICS ILE C programs.	50
Program activation	51
Sample application programs	53
Data declarations used by the FILEA sample	55
Defining resources for the FILEA sample	55
Running the FILEA sample	56
Operator instruction sample program.	56
Browse sample program	56
Inquiry and update sample program	56
Low balance report sample program	57
Order entry sample program	57
Order entry queue print sample program	58

Chapter 1. Introducing CICS for iSeries application programming

This manual assumes that you know about CICS in general, and about Original Program Model (OPM) COBOL/400 or Integrated Language Environment (ILE), ILE C or ILE COBOL application programming in particular. If you are not experienced in CICS, read “If CICS is new to you” for a list of recommended reading. If you are an experienced CICS programmer, and already have some CICS applications you want to migrate to CICS/400, see Chapter 2, “Portability and migration considerations,” on page 5.

If CICS is new to you

If you are not familiar with the basic concepts of CICS programming, you need to read some introductory text before reading this manual, for example the *CICS Application Programming Primer*, SC33-0674-01. The primer was written for CICS mainframe users, but most of the content of the book also applies to CICS/400 application programming.

There are significant differences, so consider the following points when reading the primer:

- CICS/400 supports emulated VSAM and SQL databases. It does not support DL/I.
- Basic mapping support is for minimum function BMS and the EXEC CICS SEND TEXT command.
- CICS/400 supports both 3270 and 5250 terminals. However, the programmer uses the same options for either terminal.
- OS/400 jobs use Control Language (CL) instead of Job Control Language (JCL).
- Resource definition is handled differently from the way it is in CICS mainframes. For information about setting up CICS tables, see the *CICS for iSeries Administration and Operations Guide*, SC41-5455-00.
- The architecture of the iSeries is different from that of CICS mainframe implementations. For example: partitions are not used, storage requests are handled differently, and security is controlled by OS/400 rather than by CICS.

What’s different about CICS programs?

There is not very much different about CICS programs. A typical CICS transaction is like the core of a program in which a single input is processed, with CICS taking care of opening and closing the files for you.

With CICS programs, you request operating system services, such as file input/output, by issuing an EXEC CICS command instead of using the corresponding language facility (for example, READ or WRITE).

Benefits of CICS for iSeries to CICS programmers

- OS/400 offers online help and messaging for all OS/400 CL commands used in the development of CICS/400 applications
- You can port CICS COBOL and C applications from any CICS platform to individual iSeries systems with minimal work.

- The execution diagnostic facility (EDF) similar to mainframe CICS EDF is available for debugging both COBOL and C application programs.
- CICS/400 programs can access mainframe VSAM data transparently through function shipping. Access to data in the OS/400 is through a CICS VSAM interface. CICS/400 cannot function-ship requests for DL/I (IMS) databases, but these can be accessed by using distributed transaction processing or the distributed program link function.
- Communication facilities may be used on iSeries-to-mainframe links or in peer-to-peer processing in a LAN. In addition to function shipping, the full range of CICS intercommunication facilities between systems is available:
 - Transactions can be run on a remote system (transaction routing). The remote system may be a CICS/400 system or that of another CICS platform. Terminal definitions may be shipped to a mainframe on request or duplicated there permanently. You effect transaction routing either by defining a transaction as remote, or by using the CRTE routing transaction.
 - Distributed transaction processing enables transactions to initiate and communicate synchronously with transactions in remote systems. To do this, transactions issue CICS commands for APPC conversations.
 - Transactions may also start remote transactions using asynchronous processing.
 - Distributed program link allows you to use the EXEC CICS LINK command from CICS/400 to link to a program on a remote system.
- CICS/400 supports:
 - Both mainframe and iSeries temporary storage (TS). Local TS requests are always mapped to auxiliary storage whether the request is main or auxiliary.
 - Transient data (TD), including extrapartition TD destinations.
 - Recoverable and unrecoverable queues.
 - Interval control start and automatic transaction initiation (ATI) from TD trigger level.
- Data integrity, both in and between iSeries and mainframe, is achieved by dynamic transaction backout, emergency restart, coordination of iSeries and mainframe syncpoints, and coordination support for external resource managers.
- CICS/400 provides server support for CICS clients. A CICS client is a front-end CICS system running in a workstation. A CICS client gains access to a CICS network through a communications link with a CICS server. The client requests that are supported by a CICS/400 server are transaction routing and function shipping. To the CICS/400 application program, it is transparent whether the request is from a CICS client or from a peer system.
- CICS/400 supports two-phase commitment of protected resources.

Chapter 2. Portability and migration considerations

This chapter lists some specific items that have been identified as possible migration considerations. You should read it if you are migrating application programs, either from another CICS platform, or from one release of CICS/400 to another.

Migrating from another CICS platform

This section lists some points that you should consider when migrating to CICS/400 from another CICS platform.

CICS/400

- The *CICS Family: API Structure*, SC33-1007-02 manual describes the differences at the keyword level (for the supported commands, options, and conditions) between the CICS application programming interface (API) implementation on the various CICS platforms.
- CICS/400 supports the command-level interface only. The CICS/400 API is a subset of that defined by the CICS architecture.
- The virtual storage access method (VSAM) is emulated. BDAM and DL/I are not supported. Support is provided for KSDS, ESDS, and RRDS file structures. Read Part 3, "Files and databases," on page 113 for details.
- Relative byte address (RBA) record identification is not supported for key-sequenced files on the EXEC CICS DELETE, EXEC CICS READ, EXEC CICS READNEXT, EXEC CICS READPREV, EXEC CICS RESETBR, EXEC CICS STARTBR, and EXEC CICS WRITE commands.
- CICS/400 accommodates connectivity between itself and other CICS products, by using the intersystem communication facilities of OS/400 for handling the LU 6.2 communication protocol. For further information, see Chapter 15, "Intercommunication considerations," on page 175.
- CICS/400 uses the prefix "AEG" for naming internal CICS/400 files and functions. But for portability, the prefix "DFH" is recognized, to allow CICS application programs to be compatible with CICS/400. The BMS macros have the "DFH" prefix, see Appendix D, "BMS macro summary," on page 553, as do the BMS-related constants, see Appendix B, "BMS-related constants," on page 545, and copybook members.
- CICS/400 does not suspend HANDLE processing if a subprogram is invoked by a host language call. You can use the EXEC CICS LINK command as an alternative, or if programming in COBOL/400, place the EXEC CICS PUSH HANDLE command before and the EXEC CICS POP HANDLE command after the host language call.
- EXEC CICS commands that specify an INTO option and some length parameters require special care. A mismatch between the length specified on the EXEC CICS command, and the actual length of the INTO area may provide unpredictable results.
- CICS task numbers may not be unique in CICS/400. The task number is based on the job number of the user running a CICS shell, followed by a single digit. This digit is incremented each time a new task is initiated within the shell. When the digit reaches 9, it is reset to zero. In addition, the task number is reset to the job number+1 each time the CICS shell is started. If the user repeatedly

stops and starts CICS shells without signing off OS/400 completely, the first task in each shell will have the same task number. If any part of your application relies on a unique task numbers (for example, when allocating temporary storage queue names), the code must be changed.

- The following commands and options are retained for compatibility but are treated as no-operations:
 - EXEC CICS SUSPEND, EXEC CICS WAIT JOURNAL and EXEC CICS WAIT JOURNALNUM
 - MASSINSERT option on EXEC CICS WRITE
 - WAIT, STARTIO, and NOSUSPEND options on EXEC CICS WRITE JOURNALNUM
 - DEFRESP option on EXEC CICS CONVERSE
 - BELOW and ANY options on EXEC CICS GETMAIN
 - LAST, CNOTCOMPL, and DEFRESP options on EXEC CICS SEND command

Note: The LAST option is available if you are sending data in an APPC mapped conversation.

BMS

- CICS/400 basic mapping support (BMS) most closely matches minimum function BMS, with the addition of EXEC CICS SEND TEXT. See “Level of BMS” on page 139 for a discussion of the levels of BMS. See Chapter 13, “CICS/400 basic mapping support (BMS),” on page 141 for a discussion of the support provided for BMS.
- BMS macro source input is the only acceptable input for CICS/400 map generation.
- You must remove any non-BMS assembler-language statements (for example, PRINT NOGEN) from BMS source maps.
- On CICS/400, when a user begins a conversational or pseudoconversational transaction, no messages can be delivered to that user’s terminal from any source except the transaction being processed. On other CICS platforms, pseudoconversational sequences deliver these messages as soon as no transaction is running for that user’s terminal, which may be immediately after any screen in the sequence.
- To override the attribute data displayed by BMS, you can set the corresponding subfield in the data structure to X'00', if using 3270 devices, but should use X'40' if using 5250 devices.

Terminal Control

- An application that uses EXEC CICS RECEIVE rather than EXEC CICS RECEIVE MAP should allow for the addition of SBA data to the front of data returned from a 5250 terminal.

COBOL

CICS/400® supports both the COBOL/400 and ILE COBOL compilers. In the following discussion, the term COBOL is used to apply to both compilers.

- If you want to use your own source code line numbers for debugging purposes, you do not have to use a CRTCLPGM option. You can use the NUMBER option of the COBOL PROCESS statement in your source to get columns 1–6 (the statement numbers) as the reference numbers for the compile.

- USAGE IS POINTER fields cannot be used in data areas associated with the FROM option of EXEC CICS START, WRITEQ TD, and WRITEQ TS commands, because the process used to pass the data results in the value not being recognized as a pointer by OS/400.

- CICS/400 does not support BLL cells.

- You should check COBOL conditional statements of the form:

```
IF A = B OR C
```

because complicated, abbreviated, combined conditions may fail. You should consider using the form:

```
IF A = B OR A = C
```

to avoid possible compiler errors.

- Any COBOL statements containing redefined groups, for example:

```
01 A          PIC X(100)
01 B REDEFINES A.
01 C REDEFINES B.
```

cause OS/400[®] to issue warning messages. You should consider adding PIC to the redefines to avoid warning messages, for example:

```
01 A PIC X(100)
01 B REDEFINES A PIC X(2).
01 C REDEFINES B PIC X(2).
```

- Any COBOL OCCURS statements must be of the form:

```
02 A PIC X(02) OCCURS 1 TO 100 TIMES DEPENDING ON X
instead of
OCCURS 100 TIMES
```

- Any COMPUTE statements must be:

```
COMPUTE X = Y
rather than
COMPUTE X EQUAL Y
```

- You must change all COMP and USAGE IS COMPUTATIONAL fields to BINARY (COMP equates to COMP-3 in CICS/400).

- You should bear in mind the following restrictions when using pointers in COBOL:

- Pointers are 16 bytes.
- Pointer arithmetic is not allowed.
- Pointer variables should be defined as USAGE IS POINTER.
- Redefining pointers is not recommended.
- Writing a pointer to a file nullifies the pointer.
- OS/400 initializes working storage depending upon whether the *STDINZ or the *NOSTDINZ compiler option is used, as given in the following table:

	*STDINZ	*NOSTDINZ
Group	XX'40'.	The value in the VALUE clause, else XX'00'.
Single item	The value in the VALUE clause, else a default appropriate for the type of field, for example zero for numeric fields.	The value in the VALUE clause, else XX'00'.

- You must change SELECT statement ASSIGN clauses to DATABASE-XXX or PRINTER-XXX.

- You should remove any LABEL RECORD clauses in the sort description of your SORT files.
- A GROUP item that contains spaces does not compare equal to zeros. You will receive a warning message if you code the following in COBOL under OS/400:

```
IF NBR-FLD = SPACES
```

- If there are any pointer fields used by the program, it is advisable to compile migrated code with the MAP compiler option specified on the COBOL PROCESS statement. You should also examine the layout of storage in the linkage section and working storage section. This is because the amount of storage used for a pointer on OS/400 is 16 bytes, whereas on MVS™ and ESA it is 4 bytes. The compiler attempts to insert fillers to achieve the correct alignment but these do not always lead to the most efficient use of storage, nor to the correct positioning within data structures.
- NOSYNC must be used for COBOL programs using CICS maps. Aligned maps create an output copybook where the USAGE BINARY fields are SYNC. COBOL ignores this; the compiler option default is NOSYNC. If you use the process option of SYNC, there may be serious problems with CICS maps.
- COBOL does not do a propagated MOVE. For example, with COBOL, the code:

```
01 GROUPC.
   03 DATA01 PIC X.
   03 DATA02 PIC X(30).

01 GROUPD   PIC X(31) VALUE "ABCDEFGHJKLMNOPQRSTUVWXYZ12345".

MOVE GROUPD TO GROUPC.
MOVE SPACE TO DATA01.
MOVE GROUPC TO DATA02.
```

results in the following:

Statement	Resulting Value in GROUPC
MOVE GROUPD TO GROUPC	"ABCDEFGHJKLMNOPQRSTUVWXYZ12345"
MOVE SPACE TO DATA01	" BCDEFHJKLMNOPQRSTUVWXYZ12345"
MOVE GROUPC TO DATA02	" "

With COBOL, the same code results in the following:

Statement	Resulting Value in GROUPC
MOVE GROUPD TO GROUPC	"ABCDEFGHJKLMNOPQRSTUVWXYZ12345"
MOVE SPACE TO DATA01	" BCDEFHJKLMNOPQRSTUVWXYZ12345"
MOVE GROUPC TO DATA02	" BCDEFHJKLMNOPQRSTUVWXYZ1234"

- If you are migrating copybooks that contain EXEC CICS statements, they can be translated and the generated source copied into COBOL programs. However, if your copybook contains EXEC CICS HANDLE or EXEC CICS IGNORE statements, the generated source should not be copied into the CICS application. EXEC CICS HANDLE and EXEC CICS IGNORE statements must be translated as inline source statements.
- If you are migrating COBOL applications from platforms that support non-AD/Cycle COBOL structures, you must check that all continuation statements start in the correct column. A valid CICS for MVS/ESA COBOL program can contain continuation statements that start in column 8, (following the hyphen in column 7). To conform to AD/Cycle®, continuation statements must not start before column 12. If your continuation statements start before column 12, the CICS/400 translator will not recognize them.
- COBOL does not support a graphic data type. VS COBOL II programs can use "PIC G" definitions, but these are not currently supported in COBOL/400.

You can define graphic fields in DDS copybooks, and COBOL accepts fields defined in this way, but converts the fields to alphanumeric fields in the compiled program. For ILE COBOL, these fields are converted to fixed-length G-type fields if *PICGGRAPHIC is defined as a CICS option.

- COBOL programs may contain “DBCS only” literals, or DBCS/SBCS literals, but not both types of literal.

“DBCS only” fields may contain a maximum of 80 DBCS characters for OPM COBOL and 128 DBCS characters for ILE COBOL. Both may be continued from one line in a program to the next. The method of this continuation is consistent with that of VS COBOL II.

DBCS/SBCS literals cannot continue across lines and are restricted to AREA B on one line.

DBCS characters can be used only in literals and comments, and there are some restrictions for literals such as the CALL and CANCEL statements. You should refer to the COBOL/400 publications for complete details of these restrictions. (See “Books from related libraries” on page xiv.)

- Differences exist in the rules governing methods of calling subprograms, specifically with respect to the initialization of COBOL working storage in subprograms called by COBOL calls. See Table 2 on page 24.

ILE C

- CICS/400 does not support use of the #pragma XOPTS directive. All CICS translator options must be specified on the CRTICISC CL command used to invoke the ILE C translator.
- CICS/400 does not pass the TRANSID to main() in argv[0]. CICS conforms to the OSI C standard by providing the program name in argv[0]. Access to the TRANSID can be obtained through the EIB.

SQL

- SQL column names cannot be longer than 10 characters; this also affects Distributed Relational Database Architecture™ (DRDA®) applications.

iSeries

- CICS/400 does not support, and cannot be run, within a System/36™ or System/38™ emulated environment.
- Read Chapter 14, “Terminal control,” on page 169 for information about 5250 terminals. If you want to communicate with other CICS systems, read Chapter 15, “Intercommunication considerations,” on page 175.
- The OS/400 CL command manuals describe the details of the control language, which is different from JCL. “Books from related libraries” on page xiv gives a list of specific manual references.

Source code

- Imbedded hexadecimal characters with a value less than X'40' cannot be edited by the source entry utility (SEU). You must remove them from your program source code because the translator will not be able to handle them.

Migrating from another CICS/400 release

This section lists some points that you should consider when migrating from one release of CICS/400 to another. Further information on release-to-release compatibility can be found in the *CICS for iSeries Administration and Operations Guide*, SC41-5455-00.

BMS

- You cannot use BMS maps created under the current release on a previous release, but you can use BMS maps created under a previous release on the current release.

Application programs

- You can run application programs from a previous release under the current release, and those from the current release under a previous release, provided that you specify the TGTRLS(*PRV) option on the CL compilation command.

Resource definitions

- The previous releases that are supported for migration purposes are the last release of the previous version and the previous release of the current version.
- You cannot use current release resource definition files on previous releases, but you can migrate to a supported previous release using the SAVCICSGRP command.
- You can convert previous release resource definition files to the current release. You can use either the INZCICS command after the new version has been installed or the CONVERT parameter of the STRCICS command. See the *CICS for iSeries Administration and Operations Guide*.

Chapter 3. Preparing and writing CICS applications in COBOL

This chapter describes:

- “Preparing a COBOL application”
 - “Coding CICS statements in COBOL applications” on page 12
 - “Preprocessing” on page 13
 - “Translating a COBOL program” on page 13
 - “Compiling an application program” on page 16
- “Writing CICS programs in COBOL” on page 17
 - “Modular programming” on page 18
 - “Pointer-based addressing” on page 18
 - “Getting map set storage” on page 19
 - “Source code considerations” on page 20
- “Calling programs from COBOL” on page 22
- “Sample application programs” on page 27

Preparing a COBOL application

The iSeries provides two compilers for COBOL applications using CICS:

- COBOL/400. This is the Original Program Model compiler.
- ILE COBOL. This compiler creates programs that run in the Integrated Language Environment (ILE) environment.

To create a CICS COBOL application you select the appropriate compiler by using one of the following values for the Compile Type option in the CRTICSCBL command:

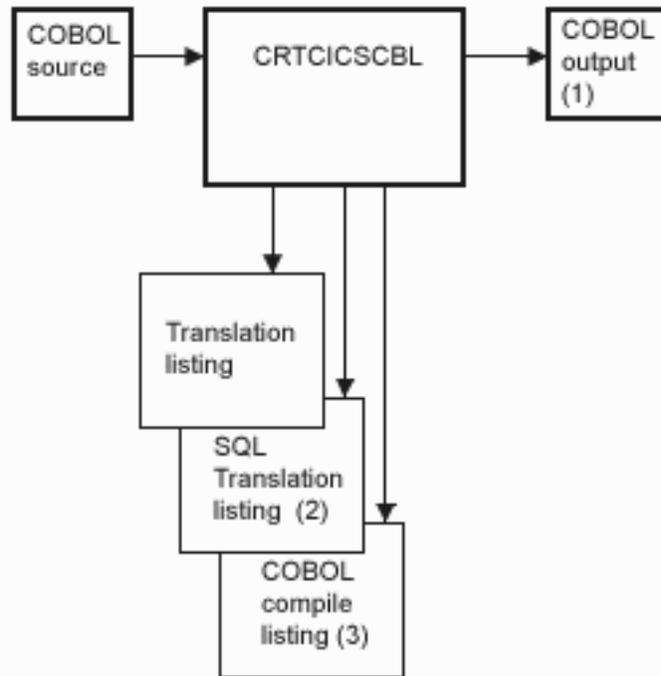
- The *PGM option is the default value and is used to create a CICS program using the COBOL/400 compiler.
- Either the *BNDPGM or *MODULE options may be used to create a CICS COBOL program or module using the ILE COBOL compiler.

You use the CRTICSCBL CL command to invoke the CICS/400 precompiler, which in turn invokes the following:

- The EXEC CICS command language translator, to convert the EXEC CICS commands in your applications into COBOL statements that the compiler understands. See “Translating a COBOL program” on page 13 for more information about how to do this.
- The SQL precompiler, if any SQL statements are encountered in your source program.
- The appropriate COBOL compiler, to produce a program object that is stored in an OS/400 library.

When your application program is executed, the statements inserted by the translator invoke the EXEC interface program. This program then provides the functions requested by each command by invoking one or more CICS service programs. The EXEC interface program also obtains and provides addressability to required areas of storage, and releases them automatically when they are no longer required.

As you can see from Figure 1, three listings can be produced. You can use these listings to check for any syntax errors.



- (1) Executable code if *NOGEN is not specified, compare with translated source in QTEMP/QACYCICS.
- (2) Listing is generated if SQL statements are found in the source.
- (3) Listing is generated if *NOGEN is not specified and no translation errors are found.

Figure 1. Preparing a COBOL application program. This picture shows the CICS/400 translator, CRTICSCBL, with a COBOL source file as input, a translator output file, and the three listings that may be produced; the translation listing, the SQL translation listing, and the COBOL compile listing.

Coding CICS statements in COBOL applications

When you need a CICS system service, for example when reading a record from a file, you include an EXEC CICS command in your code.

Each command specifies the function you want; for example, EXEC CICS READ to read a file. You then supply a number of options. Each option takes the form of a keyword, and may require an argument. For example, if you are reading a file, you use the FILE option, supplying the file name as the argument to the option. Some options do not require an argument; for example, the UPDATE option on the READ command simply tells CICS that you are updating the file. You mark the end of the command with the words END-EXEC.

For more information on the EXEC CICS command format, see Chapter 31, "Programming reference," on page 305. The commands and their options are described in Chapter 32, "Application programming commands - reference," on page 323.

When you specify an argument value, you can use a literal, or a data area where the value you want is stored. If you use a literal, follow the usual COBOL rules and put it in quotes unless it is a number. In other types of commands, these values may be paragraph names in your program, telling CICS where to go if a certain type of exception condition arises. Do not use quotes around paragraph names.

The statements generated by the translator never contain periods, unless you include one explicitly after the END-EXEC. This means you can use CICS commands within control statements (by leaving the period out of the command), or you can end a sentence with the command (by including the period). See “Translating a COBOL program” for more information about the translator.

Preprocessing

You may use a preprocessor to process source statements before the CICS translator is invoked. However, any preprocessor run before CICS translation must be able to ignore CICS statements.

Translating a COBOL program

The CICS application program interface translator translates COBOL programs with embedded EXEC CICS commands. The CICS translator program scans each statement and:

- **Verifies that each CICS statement is valid and free of syntax errors.** The validation procedure lists error messages in the output listing to help you correct any syntax errors.
- **Prepares each CICS statement for compilation in the host language.** For most EXEC CICS statements, the CICS translator inserts a comment, a series of COBOL MOVE statements, a CALL statement to the CICS interface program (AEGEIPGM), and possibly more COBOL MOVE statements.
- **Flags any SQL statements that are found within the source code.** When all CICS statements have been validated and prepared for compilation, the SQL translator is invoked if SQL statements are encountered within this source code, provided that the *GEN (default) option has been used on the CICSOPT parameter of the CRTICSCBL CL command. Otherwise the translation process stops at the end of translation of the CICS commands.

To obtain diagnostic information when you translate a program, specify the *SOURCE and *XREFCICS translator options of the CICSOPT parameter. This creates an output spool file showing you the results of the translation process.

The translator translates the EXEC commands into MOVE statements followed by a CALL statement in COBOL, and possibly more MOVE statements. The purpose of the MOVE statements is to assign constants to COBOL data variables; this enables constants and names to be specified as arguments to options in the commands.

Declarations for the generated variables are included automatically in working storage by the translator inserting a COBOL COPY statement. The variables included by this COPY statement are reserved and all begin with a “DFH” prefix.

Note: Do not use EXEC, CICS, END-EXEC, or names starting with “DFH”, as names for user variables.

The translator modifies the linkage section by inserting the EIB structure as the first parameter, and inserts a DFHCOMMAREA as the second parameter, if one is not already present. It also inserts declarations for the variables used as the receiving fields for the COBOL MOVE statements inserted by the translator.

You specify translator options using the OS/400 CL command, CRTICSCBL. The translator provides a number of optional facilities; for example, to specify what information is required on the listing. The translator options and their defaults are listed in Chapter 30, "OS/400 control language (CL) commands," on page 265.

Example

A command such as:

```
EXEC CICS RECEIVE MAP('MAPA') END-EXEC.
```

may be translated to:

```
MOVE 0 TO DFH-ARG-INDEX(6).
MOVE 'MAPA'
    TO DFH-STRING-VALUE(1).
MOVE 1 TO DFH-ARG-INDEX(9).
MOVE 8 TO DFH-ARG-CODE(1).
MOVE X"00008120" TO DFH-ARG-MASK(1).
MOVE X"00000000" TO DFH-ARG-MASK(2).
MOVE X"00000000" TO DFH-ARG-MASK(3).
MOVE X"00000000" TO DFH-ARG-MASK(4).
MOVE 3 TO DFH-ARG-COUNT.
MOVE -1 TO DFH-DEBUG-LINE.
MOVE 62 TO DFH-FN-CODE.
CALL "AEGEIPGM" USING DFHTTFTR,
    MAPAI.
IF DFH-EIBLABEL NOT EQUAL TO 0
    GO TO AEG-API-ERROR.
END-IF.
```

The CICS translator assumes that the host language statements are syntactically correct. If they are not, the translator may not correctly identify CICS statements. There are limits on the forms of source statements that can be passed through the translator. For example, literals and comments (which are not accepted by the application language compiler) can interfere with the translator source scanning process and cause errors.

Characteristics of the input source file

The translator reads input from the source physical file member specified. This source file contains language source statements in COBOL. It may contain the following items of relevance to the translator:

- EXEC CICS commands
- DFHRESP built-in functions
- DFHVALUE built-in functions

For COBOL the translator writes its output to the QACYCICS file of the QTEMP library. The specific member is given the same name as the source member. This contains the translated application program with the CICS functions commented out and followed by the equivalent language statements or function calls.

Note: If *NOGEN is specified as a CICSOPT option, the translation process ends at the end of the translation of the CICS commands.

Example

The following example uses the CRTICSCBL command to create a COBOL program named SAMPLE in library USERLIB1. See Chapter 30, “OS/400 control language (CL) commands,” on page 265 for more details.

```
CRTICSCBL PGM(USERLIB1/SAMPLE)
          CICSOPT(*SRC *XREFCICS)
```

CCSID of source files

The SQL translator reads the source records using the coded character set identifier (CCSID) of the source file. When processing SQL INCLUDE statements, the included source is converted to the CCSID of the original source file if necessary. If the included source cannot be converted to the CCSID of the original source file, an error occurs.

If double-byte character set (DBCS) literals are specified in the application program source, the CCSID, for converting DBCS characters, must indicate that the system supports DBCS literals. For more information about CCSID, see the Database and File and file systems topics in the iSeries Information Center. Before using the translator for DBCS purposes, you should refer to the *COBOL/400 User's Guide* or *WebSphere Development Studio: ILE COBOL Programmer's Guide* and become familiar with the DBCS support provided by COBOL.

If the second byte of a DBCS character has the code point X'7F', which is an SBCS double quotation mark, then if the string is surrounded by or contains double quotation marks, it will be misinterpreted by the translator. To avoid this, you are recommended to surround a string in a COBOL application, with single quotation marks instead of double quotation marks, if it contains any DBCS characters.

Output from the translator

The translator produces:

- Listings
- temporary source file members

Listings: The following listings are output by the translator to the printer file:

Translator options

Options specified in the CRTICSCBL CL command.

Translator source

Source statements, with record numbers assigned by the translator, if you specify the *SOURCE option.

Translator cross-reference

Cross-reference listing (if you specify the CICSOPT options *XREFCICS and *SRC) showing the translator line numbers of CICS statements in which host names and column names are referred to.

Translator diagnostics

Messages showing the translator record numbers of statements in error.

The output to the printer file uses a CCSID value of 65535. The data is not converted when it is output to the printer file.

Temporary source file members: Source statements processed by the translator are written to QACYCICS in the QTEMP library. In your translator-changed source code, CICS statements have been converted to a comment, MOVE, or CALL to the

CICS interface program AEGEIPGM. The name of the temporary source file member is the same as that of the original source file member. When CICS creates the QACYCICS file, it uses the CCSID value of the source file as the CCSID value for QACYCICS.

QACYCICS can be moved to a permanent library after translation, if you want to compile at a later time.

Compiling an application program

Having prepared your program source, and translated it to remove any EXEC CICS statements, you next need to compile it to create an executable program.

Unless *NOGEN is specified, the CICS precompiler automatically calls the relevant compiler after the successful completion of the translation.

- The CRTICSCBL CL program using the *PGM option automatically calls either CRTCLP or CRTSQLCBL depending on whether the input source contains EXEC SQL commands. The CRTCLP command is run specifying the program name, source file name, translator created source member name, text and USRPRF.

the following options under CICSOPT are passed to the COBOL/400 compiler:

- *SRC, *NOSRC
- *SOURCE, *NOSOURCE
- *APOST, *QUOTE
- SECLVL, *NOSECLVL

- The CRTICSCBL CL program using the *BNDCBL option automatically calls the CRTBNDCBL or the CRTSQLCBL compiler depending on whether the input source contains EXEC SQL commands. If the *MODULE option is selected to create a COBOL module rather than a program, CRTDBLMOD is called. If this option is used, then CRTPGM must be used to create an executable program.

The following options under CICSOPT are passed to the ILE COBOL compiler:

- *SRC, *NOSRC
- *SOURCE, *NOSOURCE
- *APOST, *QUOTE
- *SECLVL, *NOSECLVL
- *STDTRUNC, *NOSTDTRUNC
- *RANGE, NORANGE
- *PICXGRAPHIC, *NOPICXGRAPHIC, *PICGGRAPHIC, *NOPICGGRAPHIC

(See “Conventions and terminology used in this book” on page xiii for a discussion of the formation of these option names.) COBOL compiler defaults are used for all other parameters.

See page 266 for an explanation of the limitations on the SQL options that the COBOL compiler CL commands can pass to the SQL preprocessor.

Notes:

1. You must not change the translated source member in QTEMP/QACYCICS before issuing the CRTCLP, CRTSQLCBL, CRTBNDCBL or the CRTSQLCBLI command, or the compile may fail. The REPLACE option is passed on to the CRTCLP command.
2. When setting the compiler options, check that PIC S9(4) BINARY fields are regarded as having a range -32 767 through +32 767; otherwise you might

experience problems with LENGTH fields greater than 9999 bytes. If you have problems with this, refer to the ILE COBOL/400 manuals for more details about the NOTRUNC option.

Writing CICS programs in COBOL

This section describes some things you should be aware of when writing COBOL programs.

COBOL programs are usually coded as members of source type CICSCBL in source file QLBLSRC. See Figure 2 for an example of using source type CICSCBL.

```
Work with Members Using PDM
File . . . . . QLBLSRC
Library . . . . . QCICSSAMP          Position to . . . . .
Type options, press Enter.
 2=Edit      3=Copy      4=Delete      5=Display      6=Print
 7=Rename    8=Display description  9=Save        13=Change text ...
Opt Member  Type      Text
ACCTREC    CICSCBL   Account file record format
ACCTSET    CICSCBL   ACCT Sample Transaction
ACCT00     CICSCBL   ACCT00 CICS COBOL Source
ACCT01     CICSCBL   ACCT01 CICS COBOL Source
ACCT02     CICSCBL   ACCT02 CICS COBOL Source
ACCT03     CICSCBL   ACCT03 CICS COBOL Source
ACCT04     CICSCBL   ACCT04 CICS COBOL Source
ACIXREC    CICSCBL   Index file record format
                                                    More...

Parameters or command
===>
F3=Exit      F4=Prompt      F5=Refresh      F6=Create
F9=Retrieve   F10=Command entry  F23=More options  F24=More keys
```

Figure 2. Screen showing an example of using source type CICSCBL

Syntax checking for CICS COBOL applications is provided by the Source Entry Utility (SEU) when source type CICSCBL is used. All EXEC CICS commands are marked by the word CICS in the line number field, as shown in Figure 3 on page 18.

```

Columns . . . : 1 71          Edit          SAMPLE/QLBLSRC
SEU==>                SAMPLE
FMT CB .....-A+++B+++++
0006.00      ENVIRONMENT DIVISION.
0006.01
0007.00      DATA DIVISION.
0007.01
0008.00      WORKING-STORAGE SECTION.
0009.00
0010.00      01 MSG-TEXT      PIC X(23) VALUE IS
0011.00          ' 5722DFH CICS '.
0012.00
0013.00      PROCEDURE DIVISION.
0014.00
CICS          EXEC CICS SEND TEXT FROM(MSG-TEXT) LENGTH(23) ERASE
CICS          END-EXEC
CICS          EXEC CICS RETURN NOHANDLE END-EXEC.
0017.01
0018.00      EPILOG.
0019.00          GOBACK.
***** End of data *****
F3=Exit  F4=Prompt  F5=Refresh  F9=Retrieve  F10=Cursor
F16=Repeat find  F17=Repeat change  F24=More keys

```

Figure 3. Example of an SEU screen showing code containing CICS commands

For general information about writing COBOL programs, refer to the *WebSphere Development Studio: ILE COBOL Reference*.

Modular programming

If there is common code that is shared among many programs it can be saved as a member of a copybook. It must be translated before being stored, but do not edit the translated output. If you copy or manipulate statements originally inserted by the CICS translator in an application program, you may get unpredictable results.

Segments of programs to be copied into the procedure division can be translated by the command language translator, stored in their translated form, and later copied into the program to be compiled.

Note: Any command using a LABEL option can **not** be pre-translated and later copied into application programs. EXEC CICS HANDLE CONDITION, EXEC CICS HANDLE ABEND, and EXEC CICS HANDLE AID commands cannot be made common code because the code generated by the translator for these commands varies depending on their placement in the application program.

When using ILE, you can create modules for common code. These modules can then be combined with other modules to create an executable program. The source code for the module can contain EXEC CICS statements.

Pointer-based addressing

CICS application programs need to access data dynamically when it is in a CICS internal area, and only the address is passed to the program. Examples are:

- CICS areas such as the CWA, TWA, and TCTTE user area (TCTUA), accessed using the EXEC CICS ADDRESS command
- Input data, obtained by EXEC CICS commands such as READ and RECEIVE with the SET option

COBOL provides a simple method of obtaining addressability to the data areas defined in the linkage section, using pointer variables and the ADDRESS special

register. The ADDRESS special register holds the address of a record defined in the linkage section with level 01 or 77. This register can be used in the SET option of the EXEC CICS commands GETMAIN, LOAD, READ, and READQ.

Example of using pointer variables

Figure 4 shows how to obtain addressability to the data for pointer addressing. If the records in the EXEC CICS READ or EXEC CICS REWRITE commands are fixed-length, COBOL does not require a LENGTH option. This example assumes variable-length records. After the EXEC CICS READ command, you can get the length of the record from the field named in the LENGTH option (here, LRECL-REC1). In the EXEC CICS REWRITE command, you must code a LENGTH option if you want to replace the updated record with a record of a different length.

```
WORKING-STORAGE SECTION.  
  
77 LRECL-REC1    PIC S9(4) COMP-4.  
  
LINKAGE SECTION.  
  
01 REC-1.  
  02 FLAG1      PIC X.  
  02 MAIN-DATA  PIC X(5000).  
  02 OPTL-DATA  PIC X(1000).  
  
01 REC-2.  
  02 ...  
  
PROCEDURE DIVISION.  
  
  EXEC CICS READ UPDATE...  
    SET(ADDRESS OF REC-1)  
    LENGTH(LRECL-REC1)  
  END-EXEC.  
  
  IF FLAG1 EQUAL 'Y'  
  MOVE OPTL-DATA TO ...  
  .  
  
  EXEC CICS REWRITE...  
    FROM(REC-1)  
  END-EXEC.
```

Figure 4. Example of pointer-based addressing in a COBOL program

Note: In the above example, COMP-4 has been used. You can specify BINARY in preference to COMP-4. However, you must **not** specify COMP because the COBOL compiler treats this as equivalent to packed decimal (that is, COMP-3). COBOL compilers on other systems may treat COMP as equivalent to COMP-4.

Getting map set storage

If your basic mapping support (BMS) map is in the linkage section, you must acquire map storage dynamically with the EXEC CICS GETMAIN command. You may either release storage with the EXEC CICS FREEMAIN command or keep it until the task is complete. Early release helps you to optimize storage use, and can be useful in a long conversational transaction. See Chapter 12, "Introduction to basic mapping support (BMS)," on page 135 for further information about BMS.

With other COBOL compilers you must determine the necessary amount of storage, which must be sufficient for the largest map in your map sets. This can be difficult to determine, and probably involves examining all the map assemblies. With COBOL, use the LENGTH special register:

```
EXEC CICS GETMAIN
      SET(ADDRESS OF dataarea)
      LENGTH(LENGTH OF dataarea)
END-EXEC.
```

In COBOL, the actual processing of maps in the linkage section is simplified by the elimination of BLL cells.

Figure 5 shows the method of processing BMS maps in the linkage section. The highlighted material describes the contents of the MAPSET1 COBOL copybook. MAPSET1 was defined as follows:

```
MAPSET1 DFHMSD TYPE=DSECT,
          LANG=COBOL,
          STORAGE=AUTO,
          MODE=IN
```

In this example, it is assumed that the COBOL program has been compiled and that MAPSET1 has been included in the program.

```
WORKING-STORAGE SECTION.
77 FLD0 PIC X VALUE IS LOW-VALUE.

LINKAGE SECTION.

COPY MAPSET1.

01 MAP1I.
  02 FILLER PIC X(12).
  02 FILLER1L BINARY PIC S9(4).
  .
  .
  02 FIELD90 PIC X(20).

PROCEDURE DIVISION.

EXEC CICS GETMAIN
      FLENGTH(LENGTH OF MAP1I)
      SET(ADDRESS OF MAP1I)
      INITIMG(FLD0)
END-EXEC.
```

Figure 5. Example of processing BMS maps in a COBOL program

Source code considerations

You do not define your files in a CICS COBOL program, but in the CICS file-control table entries. At most sites this table is updated by the system administrator. See the *CICS for iSeries Administration and Operations Guide* for more information about setting up entries in the file control table.

- Do not use the entries in the environment division and the data division that are normally associated with files. In particular, the entire file section is omitted from the data division. However, you still need to code the headers for both of these divisions. Put the record formats that usually appear there in either the working storage or linkage sections.
- Do not use the COBOL READ, WRITE, OPEN, and CLOSE statements. Use the appropriate CICS commands for storing and retrieving data, and for communication with terminals.
- The translator expands all CICS commands to COBOL CALL commands, so the compiler expects a return to the calling program. Control returns to CICS after the EXEC CICS RETURN command.
- The COBOL compiler limits are given in Table 1. See the appropriate COBOL compiler reference book for other compiler limits.

Table 1. COBOL compiler limits

Language element	COBOL/400 Limit	ILE COBOL Limit
Table size—mixed length (bytes)	3 000 000	16 711 568
Table size—variable length (bytes)	32 767	16 711 568
Table element size (bytes)	32 767	16 711 568
GO TO proc-name DEPENDING ON	255	Virtually no limit

- If both the identification and procedure divisions are presented to the translator in the form of a source program or copybook, the following coding is produced or expanded:

DFHWRKSTART

inserted at the beginning of the working storage section. This indicates the beginning of WORKING STORAGE.

DFHTTFTR

inserted at the end of the working storage section. This brings in a 01 level of the same name containing a number of COBOL data areas used as receiver fields by COBOL statements inserted by the translator.

DFHWRKEND

inserted at the end of the working storage section. This indicates the end of working storage.

DFHEIBLK

inserted at the start of the linkage section as the first 01 level in the section. This brings in the EXEC interface block (EIB). See Appendix A, “EXEC interface block,” on page 529 for a description of the fields in the EIB.

DFHCOMMAREA

generated, if not specified, as the second 01 level in the linkage section. This brings in the communication area. See “COMMAREA in EXEC CICS LINK and EXEC CICS XCTL commands” on page 70 for more information about the communication area.

If no identification division is present, only the CICS commands are expanded.

If the identification division *only* is present, only DFHEIBLK and DFHCOMMAREA are produced.

- If a debugging line is to be used as a comment, it must not contain any unmatched quotation marks.
- Statements that produce variable-length areas, such as OCCURS DEPENDING ON, should be used with caution within the working storage section.
- Avoid invoking interactive CICS shells from within a COBOL program.
The STRCICSUSR CL command, which starts an interactive application shell, invokes an intermediate COBOL program before invoking the shell program. This intermediate program deactivates the COBOL run unit to enable CICS error handling.
When the shell terminates, the run unit is not reactivated. If the shell was invoked from a higher level COBOL program then unpredictable results could occur during exceptions.
- Using PERFORM to execute a COBOL subroutine is much more efficient than using CICS to link to, or transfer control to, another program. However, each PERFORM brings in a copy of the designated subroutine. Repeating the subroutine in each of your COBOL application programs uses much more storage. Like other COBOL compilers, the COBOL compiler allows a COBOL program to use a CALL to external routines. The called routines can issue CICS commands. This avoids the CICS overhead of transferring control between programs, but it does mean loading the routines for every calling program. Try to keep the code you PERFORM as near as you can to the controlling PERFORM statement, to minimize the risk of the two items being in separate pages of storage.
Using PERFORM with code that isn't a true "subroutine" (as in structured programming) may also affect response time. Consider these guidelines when using PERFORM:
 - Use PERFORM to help structure code, at the cost of increased paging.
 - Keep called code as close as possible to the PERFORM statement.
 - Use PERFORM for long code, or code used in a great many places.

Calling programs from COBOL

In a CICS system, there are two ways of transferring control to another program:

- Using EXEC CICS LINK and XCTL commands
- Using host language calls

Using CICS commands

In an COBOL CICS application, you can invoke CICS services to link or transfer control to an external program.

The calling program contains one of the following CICS commands:

- EXEC CICS LINK


```
EXEC CICS LINK PROGRAM('SUBPGM')
END-EXEC.
EXEC CICS LINK PROGRAM(subpgm)
END-EXEC.
```
- EXEC CICS XCTL


```
EXEC CICS XCTL PROGRAM('PGMNAME')
END-EXEC.
EXEC CICS XCTL PROGRAM(pgmname)
END-EXEC.
```

Note: The called program may be named explicitly as a non-numeric literal within quotation marks or as a COBOL data area with length equal to that required for the name of the program.

Figure 6 provides an overview of how control is transferred between programs when either the EXEC CICS LINK or the EXEC CICS XCTL (transfer control) command is used.

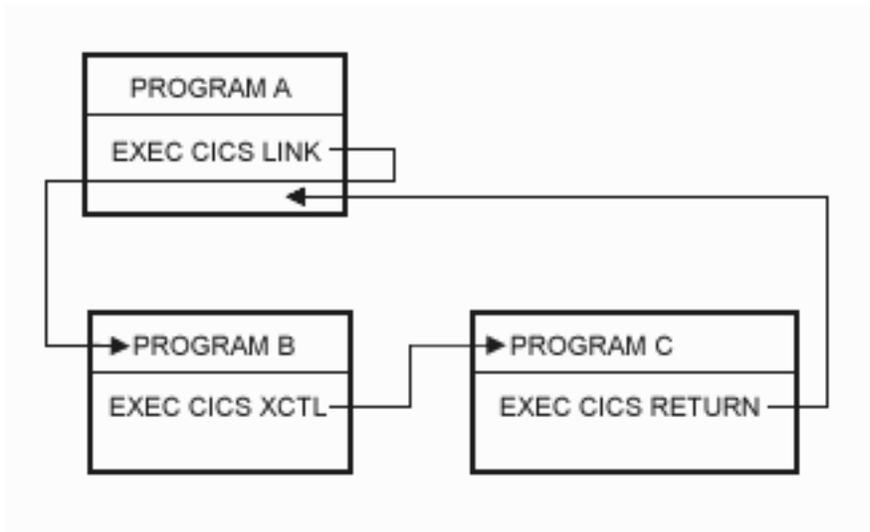


Figure 6. Control is returned to the next higher logical level.

The EXEC CICS LINK command transfers control to the specified program (B) at a new logical level. Program B could use the EXEC CICS RETURN command to cause control to return to the calling program (A). This is similar to the use of a COBOL CALL statement.

The EXEC CICS XCTL command in Program B transfers control to Program C at the same logical level. You *cannot* return control to the calling program (B) using an EXEC CICS RETURN command or a host language statement. When program C completes, control returns to program A.

In a CICS system, when control is transferred from an active program to an external program, but the transferring program remains active and control can be returned to it, the program to which control is transferred is called a subprogram.

Using COBOL CALL statements

In a COBOL application, there are two types of call that can be made during run time:

- Static call
- Dynamic call

Static COBOL call

The calling program contains a COBOL statement of the form:

```
CALL 'subpgname'
```

The called subprogram is explicitly named as a literal string and the reference to the called program is determined at compile time.

Dynamic COBOL call

The calling program contains a COBOL statement of the form:

```
CALL identifier
```

The identifier is the name of a COBOL data area that must contain the name of the called subprogram.

Rules governing calling CICS COBOL programs

Whether you use CICS commands or host language calls to call subprograms depends on a number of factors including functionality, portability, and performance. Table 2 outlines some of the requirements and conventions that you need to consider when choosing your implementation. This table defines requirements when a COBOL program is called as a main transaction program (called by CICS at logical level 1) or as a subprogram.

Table 2. Rules to be used with CICS COBOL programs

Programming requirement	Main program	Subprogram called by EXEC CICS LINK/XCTL	Subprogram called by language call
Install in control region Processing Program Table (PPT)	REQUIRED Transaction will abend APCT if not found or not authorized.	REQUIRED If not defined, not found, or disabled, a PGMIDERR condition is raised in the calling program.	OPTIONAL
Translation	REQUIRED Control is passed to the COBOL program entry point.		OPTIONAL
Source code	Any COBOL function supported by CICS, including precompiled EXEC SQL statements.		
Parameters and shared data	If the COMMAREA is used, it must be passed in the EXEC CICS LINK command.		If translated, may receive data passed by any of the standard CICS methods (COMMAREA, TCTUA, TS queues, TWA). When using language calls to call a translated CICS COBOL program, the calling program must supply pointers to the EXEC Interface Block and the COMMAREA as the first two parameters as expected.
Returning control to the calling program	Use EXEC CICS RETURN. Optionally, use GOBACK, EXIT PROGRAM, or STOP RUN. For ILE COBOL, use EXIT PROGRAM AND CONTINUE RUN UNIT.		Use GOBACK or EXIT PROGRAM. Use of EXEC CICS RETURN or STOP RUN may give unpredictable results.

Table 2. Rules to be used with CICS COBOL programs (continued)

Programming requirement	Main program	Subprogram called by EXEC CICS LINK/XCTL	Subprogram called by language call
Program activation	<p>On each entry, a CICS run unit is initialized and a new initialized copy of WORKING STORAGE is provided. In some circumstances, this can cause a performance degradation. CICS supports recursive program links. Native COBOL run-unit processing is suspended during a CICS application shell. If performance is unsatisfactory with LINK commands, COBOL calls may give improved results.</p>		<p>On first entry to the called subprogram, WORKING STORAGE is initialized. On subsequent entries to the called subprogram, WORKING STORAGE is provided in its last-used state; that is, no storage is freed, acquired, or initialized.</p> <p>The subprogram is deactivated only when the run unit ends or when an appropriate COBOL CANCEL statement is issued.</p> <p>Recursive calls may give unpredictable results.</p>
Condition AID and abend handling	<p>Condition AID and abend handling is initially defaulted.</p> <p>The system default action for unhandled conditions is usually to abend the task.</p> <p>EIBAID and EIBRESP fields may be tested if NOHANDLE or RESP options are coded on CICS commands.</p> <p>For more information on exception handling, see Chapter 6, "Dealing with exception conditions," on page 87.</p>	<p>On entry to the run unit, no condition or abend handling is active. Within the subprogram, the normal CICS rules apply. If an abend occurs while no abend handling is active, CICS searches successively higher logical levels (starting with the caller) and passes control to the label or program specified in the first active HANDLE ABEND command found. If none is found, the transaction abends.</p>	<p>If translated, CICS assumes that the calling program has issued a PUSH HANDLE command to suspend HANDLE processing. In order to establish an abend or condition handling environment that will exist for the duration of the subprogram, new HANDLE commands should be issued on entry to the subprogram. The environment created remains in effect until either another HANDLE command is issued, or the subprogram returns control to the caller. Upon return from the called subprogram, the calling program should issue a POP HANDLE command to restore HANDLE processing.</p>

Program activation

Activation is the process of preparing a program to run. It includes allocating the data or static storage needed by the program. There are rules that govern when a program is activated and deactivated.

A run unit is a running set of one or more programs that communicate with each other by COBOL CALL statements. In a CICS environment, a run unit is created by

the first program in a CICS task or by a program invoked by an EXEC CICS LINK or EXEC CICS XCTL command. The first program in a run unit is usually referred to as the *main program*.

Native COBOL run unit processing is suspended within a CICS/400 application shell. The run unit main program, which is a PPT resource defined to CICS, is activated by CICS. Subprograms called using COBOL CALL statements are activated the first time they are called. On subsequent calls, COBOL WORKING STORAGE will be in its 'last-used' state and the re-initialization of any items becomes the responsibility of the programmer. Such subprograms are deactivated only when the run unit ends or when an appropriate COBOL CANCEL statement is issued.

A called program must not directly or indirectly execute its caller (such as program X calling program Y; program Y calling program Z; and program Z then calling program X). This is called a recursive call. COBOL allows recursion in both main programs and subprograms. However, if you want your programs to conform to Systems Application Architecture® (SAA) standards, do not use recursive calls within a run unit.

Because each EXEC CICS LINK or EXEC CICS XCTL command initiates a new run unit, each time you link to a COBOL program working storage will be reinitialized.

Figure 7 on page 27 shows run units, logical levels, and the effects of using CICS commands and COBOL verbs to transfer control between programs. The dashed boxes marked A, B, and C show the scope of COBOL run units.

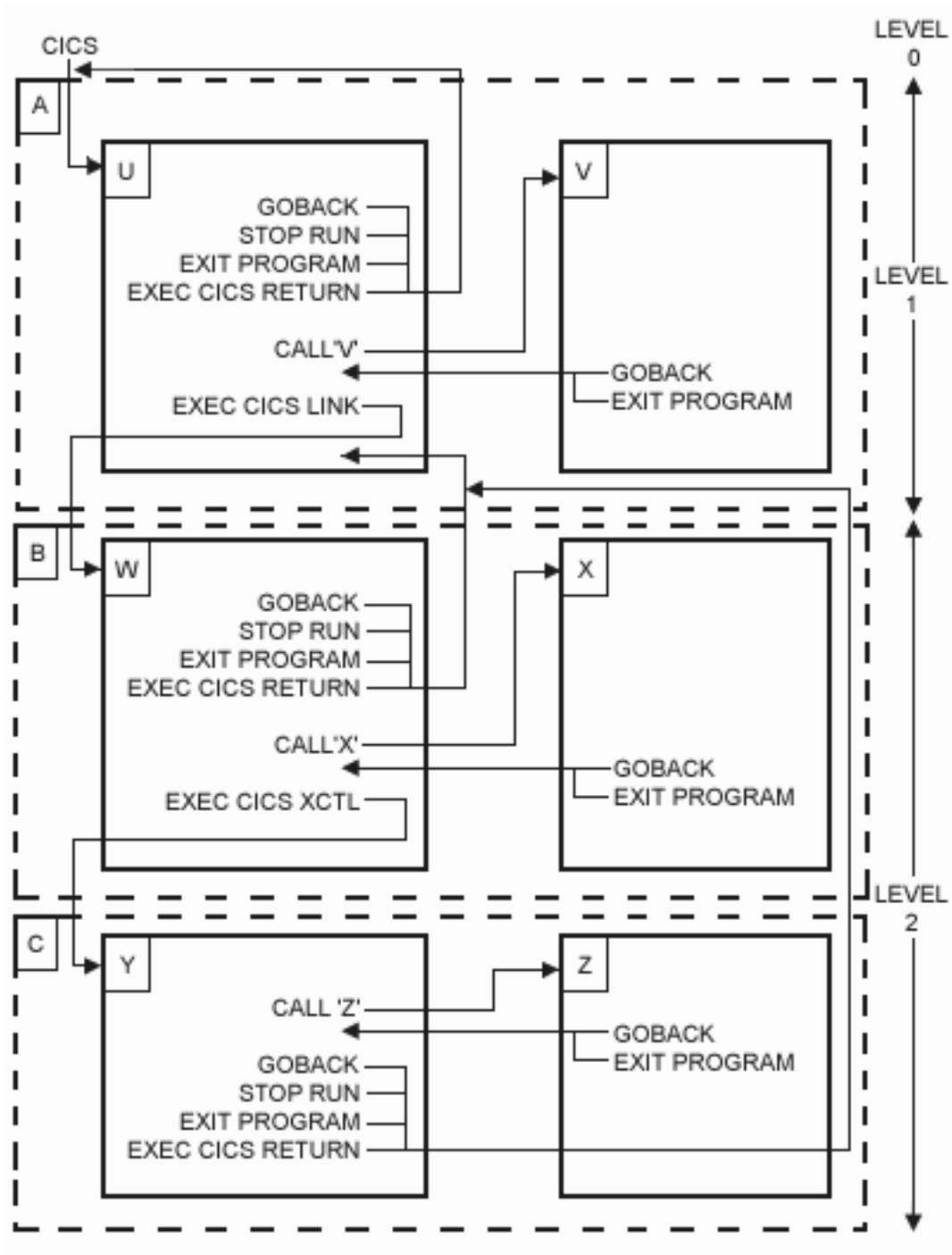


Figure 7. Flow of control between COBOL programs and run units in CICS/400

Sample application programs

Disclaimer:

The ACCT sample application contains programming source code for your consideration. This sample has not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, performance or function of the programs. All programs herein are provided to you “as is”. IBM EXPRESSLY DISCLAIMS ALL WARRANTIES, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

A set of sample application programs, called ACCT, is provided as an example of how CICS commands can be used in a program written in the COBOL language. All ACCT objects are in the QCICSSAMP library.

If the COBOL compiler is installed on your system, you can modify the ACCT sample programs. Figure 8 is a screen showing the members of file QLBSLRC that contain the source for the ACCT application programs. Note that the TYPE of the members containing CICS code is CICSCBL.

```

Work with Members Using PDM                               AS400A
File . . . . . QLBSLRC
Library . . . . . QCICSSAMP                               Position to . . . . .
Type options, press Enter.
2=Edit      3=Copy  4=Delete 5=Display  6=Print  7=Rename
8=Display descr. 9=Save 13=Change text 14=Compile 15=Create module.

Opt Member      Type      Text
ACCTREC      CBL      Account file record format
ACCTSET      CBL      ACCTSET CICS BMS Source
ACCT00       CICSCBL   ACCT00 CICS COBOL Source
ACCT01       CICSCBL   ACCT01 CICS COBOL Source
ACCT02       CICSCBL   ACCT02 CICS COBOL Source
ACCT03       CICSCBL   ACCT03 CICS COBOL Source
ACCT04       CICSCBL   ACCT04 CICS COBOL Source
ACIXREC      CBL      Index file record format

Parameters or command
====>
F3=Exit      F4=Prompt      F5=Refresh      F6=Create
F9=Retrieve   F10=Command entry F23=More options F24=More keys

Bottom

```

Figure 8. ACCT sample screen: Working with the ACCT samples

The BMS maps used by the ACCT sample are contained in a single mapset. The source for this mapset is in member ACCTSET of file QMAPSRC. The TYPE of the source members is CICS MAP.

Data declarations used by the ACCT sample

- The data for each account for the ACCT sample is stored in a physical file, ACCTFIL, in the QCICSSAMP library. The record format for this file is declared to a program by copying the copybook ACCTFIL into the working storage section. The source for the copybook is in member ACCTFIL of file QDDSSRC.
- The ACCT sample uses a physical file, ACCTIX, as an index of the account data by customer name. The copybook for the record format is called ACIXREC and the source is in ACCTIX; both ACIXREC and ACCTIX are members of file QDDSSRC. Note that logical files can be used to provide alternative indexes into a physical file.

Defining resources for the ACCT sample

A sample program, CRTSAMP, is provided to simplify resource definition for the ACCT sample. This program creates a CICS group, called ACCT, and other necessary resources for a control region. You specify the name of the control region and the library in which the resources are created as parameters for the program. When the CICS resources have been defined, the control region is started.

To run the CRTSAMP program, enter:

```
CALL CRTSAMP library-name sys-id
```

where `sys-id` is the name of the sample control region to be created and `library-name` is the library that the CICS resources are stored in. These are required parameters. The source for CRTSAMP is in member CRTSAMP of file QCLSRC in library QCICSSAMP.

Running the ACCT sample

1. Use CRTSAMP to create a control region and the CICS group, ACCT, that defines all the CICS resources needed to run the ACCT sample.
2. Start a CICS user shell for the control region created in step 1.
3. When the user shell has started, enter the transaction identifier ACCT to start the ACCT sample. The BMS map shown in Figure 9 is displayed at the terminal.

```
ACCOUNT FILE: MENU

TO SEARCH BY NAME, ENTER:                ONLY SURNAME
                                           REQUIRED. EITHER
SURNAME: _____ FIRST NAME: _____ MAY BE PARTIAL.

FOR INDIVIDUAL RECORDS, ENTER:           PRINTER REQUIRED
                                           ONLY FOR PRINT
REQUEST TYPE: _ ACCOUNT: _____ PRINTER: _____ REQUESTS.

REQUEST TYPES: D = DISPLAY      A = ADD      X = DELETE
                P = PRINT       M = MODIFY

THEN PRESS "ENTER"          -OR- PRESS "CLEAR" TO EXIT
```

Figure 9. ACCT sample screen: Menu

Note: The input fields in Figure 9 are marked with underscores, but these do not appear on the actual screen.

If you press the CLEAR key, the application ends and control passes back to CICS.

To request a function, fill in the appropriate fields on the map and press ENTER. If the request is invalid, the menu map is redisplayed, with a message that describes the error.

Displaying an account record

To display an account record, starting from the ACCT menu display, enter 'D' in the REQUEST TYPE field and the account number in the ACCOUNT field and press ENTER. The account details are displayed. Figure 10 on page 30 is an example of what the screen might look like following a request to display an account record.

```

ACCOUNT FILE: RECORD DISPLAY

ACCOUNT NO: 11111      SURNAME:  Matthews
                    FIRST:   Dan           MI:   TITLE: MR
TELEPHONE: 0001234567 ADDRESS:  1, The House
                                   The Road
                                   The City

OTHERS WHO MAY CHARGE:

NO. CARDS ISSUED: 1   DATE ISSUED: 02 04 93   REASON: N
CARD CODE: N         APPROVED BY: ISW       SPECIAL CODES: L

ACCOUNT STATUS: N     CHARGE LIMIT: 1000.00

HISTORY:  BALANCE    BILLED      AMOUNT      PAID         AMOUNT
          0.00      00/00/00    0.00        00/00/00    0.00
          0.00      00/00/00    0.00        00/00/00    0.00
          0.00      00/00/00    0.00        00/00/00    0.00

PRESS 'CLEAR' OR 'ENTER' WHEN FINISHED

```

Figure 10. ACCT sample screen: Displaying a record

Adding an account record

To add an account record, starting from the ACCT menu display, enter 'A' in the REQUEST TYPE field and the account number in the ACCOUNT field and press ENTER. If an account already exists with the account number specified, the application returns to the menu with a message that describes the error. Figure 11 on page 31 shows the map displayed on the screen, with details for a new account already entered. The ACCOUNT NUMBER field is already set to the value entered on the menu. To continue with the request, enter the account details into the map and press ENTER; otherwise press CLEAR to cancel the request. If any of the details added are invalid, the map is redisplayed with the current data, and the field containing the invalid data is highlighted with asterisks.

Note: The CARD CODE field indicates whether the card is lost (L), stolen (S), new (N), or reissued (R). No checking is done on the REASON and SPECIAL CODES fields.

```

ACCOUNT FILE: NEW RECORD

ACCOUNT NO: 11111      SURNAME:  Matthews
                    FIRST:    Dan           MI:    TITLE: MR
TELEPHONE: 01234567  ADDRESS:  1, The House
                                   The Road
                                   The City

OTHERS WHO MAY CHARGE:

NO. CARDS ISSUED: 1   DATE ISSUED: 02 04 93   REASON: N
CARD CODE: N         APPROVED BY: ISW       SPECIAL CODES: L

FILL IN AND PRESS 'ENTER,' OR 'CLEAR' TO CANCEL

```

Figure 11. ACCT sample screen: Adding a new record

Searching by account holder's name

You can search for account information by name, instead of by account number. To do this, you enter a name in the SURNAME field and, optionally, in the FIRST NAME field on the menu map, and press ENTER. If the name entered is not valid (not alphabetic), or no match can be found, the menu map is redisplayed with a message. Figure 12 shows an example of the display returned when you search for a name.

```

ACCOUNT FILE: MENU

TO SEARCH BY NAME, ENTER:                                ONLY SURNAME
                                                         REQUIRED. EITHER
                                                         MAY BE PARTIAL.

  SURNAME:          FIRST NAME:

FOR INDIVIDUAL RECORDS, ENTER:

REQUEST TYPE:      ACCOUNT:          PRINTER:          PRINTER REQUIRED
                                                         ONLY FOR PRINT
                                                         REQUESTS.

REQUEST TYPES:  D = DISPLAY  A = ADD    X = DELETE
                P = PRINT   M = MODIFY

THEN PRESS "ENTER"          -OR-  PRESS "CLEAR" TO EXIT

ACCT  SURNAME  FIRST  MI  TTL  ADDRESS          ST  LIMIT
11111  Matthews  Dan    MR  MR   1, The House     N  1000.00

```

Figure 12. ACCT sample screen: Searching by account holder's name

Modifying an account record

To modify an account record, starting from the ACCT menu display, enter 'M' in the REQUEST TYPE field and the account number in the ACCOUNT field and press ENTER. If the account does not exist or is already in use, the application returns to the menu display with a message.

Figure 13 shows an update map for an example account record. Changes are made to the record by typing over the current values. If an invalid change is made, the map is redisplayed and the field containing the invalid data is highlighted with asterisks.

```

ACCOUNT FILE: RECORD CHANGE

ACCOUNT NO: 11111      SURNAME:  Matthews
                    FIRST:   Dan           MI:   TITLE: MR
TELEPHONE: 0001234567 ADDRESS:  1, The House
                                         The Road
                                         The City

OTHERS WHO MAY CHARGE:

NO. CARDS ISSUED: 1   DATE ISSUED: 02 04 93   REASON: N
CARD CODE: N         APPROVED BY: ISW       SPECIAL CODES: L

ACCOUNT STATUS: N    CHARGE LIMIT: 1000.00

HISTORY:  BALANCE    BILLED      AMOUNT      PAID          AMOUNT
          0.00      00/00/00    0.00        00/00/00     0.00
          0.00      00/00/00    0.00        00/00/00     0.00
          0.00      00/00/00    0.00        00/00/00     0.00

MAKE CHANGES AND 'ENTER' OR 'CLEAR' TO CANCEL

```

Figure 13. ACCT sample screen: Modifying a record

When you have entered your request, the record is updated and you are returned to the menu with a message confirming that the operation was successful.

Deleting an account record

To delete an account record, starting from the ACCT menu display, enter 'X' in the REQUEST TYPE field and the account number to be deleted in the ACCOUNT field and press ENTER. If the account does not exist or is already in use, the application returns to the menu with a message informing you of the error. Figure 14 on page 33 is an account deletion map for an example account record.

```

ACCOUNT FILE: DELETION

ACCOUNT NO: 11111      SURNAME: Matthews
FIRST: Dan            MI:    TITLE: MR
TELEPHONE: 0001234567 ADDRESS: 1, The House
                               The Road
                               The City

OTHERS WHO MAY CHARGE:

NO. CARDS ISSUED: 1    DATE ISSUED: 02 04 93    REASON: N
CARD CODE: N          APPROVED BY: ISW        SPECIAL CODES: L

ACCOUNT STATUS: N     CHARGE LIMIT: 1000.00

HISTORY:  BALANCE    BILLED    AMOUNT    PAID      AMOUNT
           0.00      00/00/00    0.00      00/00/00    0.00
           0.00      00/00/00    0.00      00/00/00    0.00
           0.00      00/00/00    0.00      00/00/00    0.00

ENTER 'Y' TO CONFIRM OR 'CLEAR' TO CANCEL

```

Figure 14. ACCT sample screen: Deleting a record

You are asked to confirm the deletion request. When you have done so, the record is deleted and you are returned to the menu with a message confirming that the operation was successful.

Printing an account record

To do this, starting from the ACCT menu display, enter 'P' in the REQUEST TYPE field and the account number to be printed in the ACCOUNT field and press ENTER. If the account does not exist, or is already in use, the application returns to the menu with a message informing you of the error. When you have entered your request, the record is sent to the printer and you see a message confirming that the operation was successful.

To print the change log, enter the transaction identifier ACLG from a blank entry screen in CICS. The change log is sent to the device specified to CICS as L860 in the CICS terminal control table and you see a message confirming that the operation was successful.

Chapter 4. Preparing and writing CICS applications in ILE C

This chapter covers:

- “Preparing an ILE C application”
 - “Coding CICS statements in iSeries applications” on page 37
 - “Preprocessing” on page 38
 - “Translating an ILE C program” on page 38
 - “Compiling an application program” on page 40
- “Writing CICS programs in ILE C” on page 41
 - “Modular programming” on page 42
 - “Pointer-based addressing” on page 42
 - “Getting map set storage” on page 44
 - “Passing arguments by value” on page 44
 - “Exception handling” on page 46
 - “Data declarations needed for ILE C” on page 47
 - “Naming EIB fields” on page 47
 - “Source code considerations” on page 47
- “Calling programs and ILE procedures from ILE C” on page 48
- “Sample application programs” on page 53

Preparing an ILE C application

The tasks involved for preparing an ILE CICS application program are:

- Preprocessing an ILE C CICS program
- Translating an ILE C CICS program
- Compiling the output from the translator into an ILE module
- Binding of one or more modules into a program

ILE modules appear language independent to the program binder. Modules in any supported ILE language can be bound together into one program.

Note: For CICS/400 Version 5, the only supported ILE language is ILE C. You can also mix ILE and AD/Cycle applications across and within logical levels, using the EXEC CICS LINK and XCTL commands. See “Calling programs and ILE procedures from ILE C” on page 48 for details of logical levels.

You use the CRTICISC CL command to invoke the CICS/400 precompiler, which performs the following steps:

1. Calls the ILE C preprocessor to include header files and expand macro definitions, if selected by including the value *PP in the CICSOPT parameter list.
2. Calls the CICS C translator to convert the EXEC CICS commands in your application into C statements that can be compiled by the ILE C compiler. See “Translating an ILE C program” on page 38 for more information about the CICS C translator.

If conversion errors occur, or the *NOGEN option has been specified on the CICSOPT parameter list, processing stops at this point. The converted output will have been written to the file member specified by the OUTFILE and OUTMBR parameters.

3. If the program source contains EXEC SQL statements, the CRTSQLCI CL command is called to convert EXEC SQL statements.
If conversion errors occur, or the *NOGEN option has been specified on the SQLOPT parameter list, processing stops at this point.
4. Calls the CRTMOD CL command to invoke the ILE C compiler and create a module, if OBJTYPE(*MODULE) was specified.
5. If the *PGM option has been specified for the OBJTYPE parameter, the CRTPGM CL command is called to create a bound program.

When your application program is executed, the statements inserted by the translator invoke the EXEC interface program. This program then provides the functions requested by each command by invoking one or more CICS service programs. The EXEC interface program also obtains and provides addressability to required areas of storage, and releases them automatically when they are no longer required.

As you can see from Figure 15 on page 37, three listings can be produced. You can use these listings to check for any syntax errors.

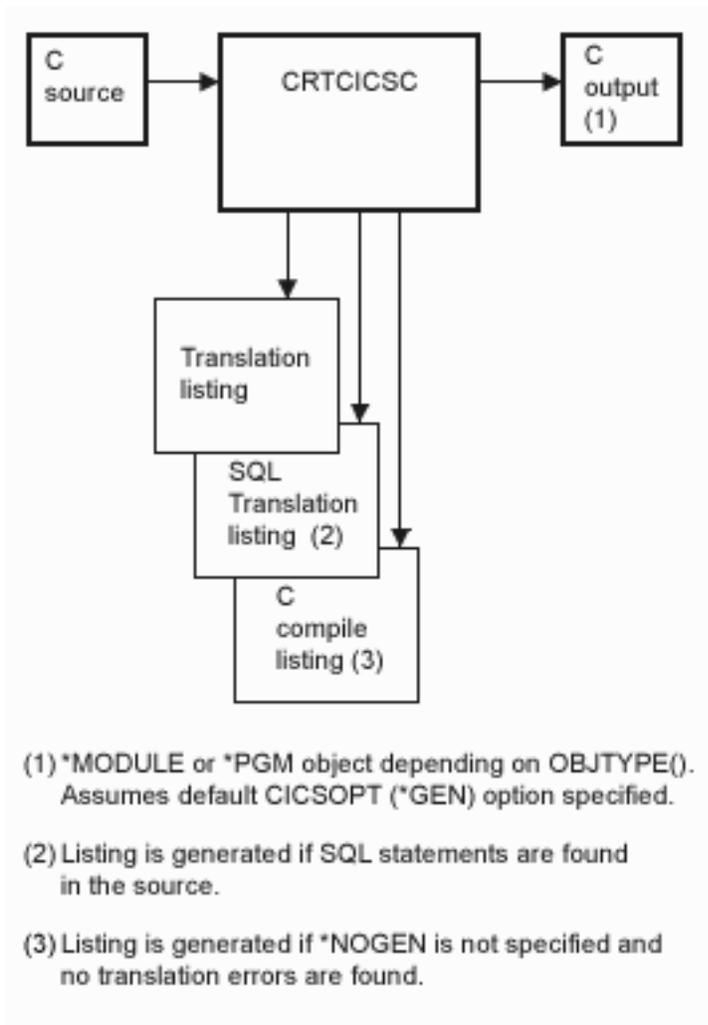


Figure 15. Preparing an ILE C application program. This picture shows the CICS/400 precompiler, CRTICISC, with a ILE C source file as input, a translator output file, and the three listings that may be produced: the translation listing, the SQL translation listing, and the ILE C compile listing.

Coding CICS statements in iSeries applications

When you need a CICS system service, for example when reading a record from a file, you include an EXEC CICS command in your code.

Each command specifies the function you want; for example, EXEC CICS READ to read a file. You then supply a number of options. Each option takes the form of a keyword, and may require an argument. For example, if you are reading a file, you use the FILE option, supplying the file name as the argument to the option. Some options do not require an argument; for example, the UPDATE option on the READ command simply tells CICS that you are updating the file. You mark the end of the command with a semi-colon (;).

For more information on the EXEC CICS command format, see Chapter 31, "Programming reference," on page 305. The commands and their options are described in Chapter 32, "Application programming commands - reference," on page 323.

When you specify an argument value, you can use a literal, or a data area where the value you want is stored. If you use a literal, follow the usual iSeries rules. See “Translating an ILE C program” for more information about the translator.

Preprocessing

If the CICSOPT parameter of the CRTICISC CL command is set to *PP, the ILE C preprocessor is called to resolve any **#define** and **#include** statements before the translator is called. This stage is optional and is omitted by default.

Translating an ILE C program

The CICS translator converts ILE C programs with embedded EXEC CICS commands. The CICS translator scans each statement and:

- **Verifies that each CICS statement is valid and free of syntax errors.** The validation procedure lists error messages in the output listing to help you correct any syntax errors.
- **Prepares each CICS statement for compilation by the ILE C compiler.** For most EXEC CICS statements, the CICS translator inserts comment limiters round the EXEC CICS commands, a series of C assignments in place of the EXEC CICS commands, a function call to the CICS interface module (AEGEIPGM), and more C assignments.
- **Flags any SQL statements that are found within the source code.** When all CICS statements have been validated and prepared for compilation, the SQL translator is invoked if SQL statements are encountered within this source code, provided that the *GEN (default) option is used on the CICSOPT parameter of the CRTICISC CL command. Otherwise, the translation process ends after the translation of the CICS commands.

To obtain diagnostic information when you translate, specify the *SOURCE and *XREFCICS translator options of the CICSOPT parameter. This creates an output spool file showing you the results of the translation process.

The translator comments out the EXEC commands and inserts ILE C statements that assign constants to ILE C data variables; this enables constants and names to be specified as arguments to options in the commands.

The file DFHTTFTR.H contains the declarations for the generated variables. The translator inserts a #include statement for this file at the beginning of the translated source file. The variables included by this #include statement are reserved and all begin with a “DFH” prefix.

Note: Do not use EXEC, CICS, or names starting with “AEG”, “DFH”, or “FAA”, as names for user variables.

The file DFHEIBLK.H contains the EXEC interface block (EIB) structure. The translator inserts a #include statement for this file at the beginning of the translated source file. See Appendix A, “EXEC interface block,” on page 529 for details of the EIB.

The translator provides a number of optional facilities. For example, you can specify the information you require on the listing. You specify translator options on the CICSOPT option of the CRTICISC CL command. See “CRTICISC” on page 286 for details.

The CICS translator assumes that the host language statements are syntactically correct. If they are not, the translator may not correctly identify CICS statements.

Characteristics of the input source file

The translator reads input from the source physical file member specified. This source file contains language source statements in ILE C. It may contain the following items of relevance to the translator:

- EXEC CICS commands
- DFHRESP built-in functions
- DFHVALUE built-in functions

You can specify the name of the library and file to be used for the translator output. By default, the translator writes its output to file QACYCICS in the QTEMP library, and the specific member is given the same name as the source member. This contains the translated application program with the CICS functions commented out and followed by the equivalent language statements or function calls.

Note: If *NOGEN is specified as a CICSOPT option, compilation terminates at the end of the CICS translation process.

Example

The following example uses the CRTICSC CL command to create a ILE C module named SAMPLE in library USERLIB1. See Chapter 30, “OS/400 control language (CL) commands,” on page 265 for more details.

```
CRTICSC OBJ(USERLIB1/SAMPLE)
        CICSOPT(*SRC *XREFCICS)
```

Output from the translator

The translator produces:

- Listings
- Temporary source files

Listings: The following listings are output by the translator to the printer file:

Translator options

Options specified in the CICSOPT parameter of the CRTICSC CL command.

Translator source

Source statements, with record numbers assigned by the translator, if you specify the *SOURCE option.

Translator cross-reference

Cross-reference listing (if you specify the CICSOPT options *XREFCICS and *SRC) showing the line numbers of CICS statements in which host names and column names are referenced.

Translator diagnostics

Messages showing the record numbers of statements in error.

Temporary source file members: Statements generated by the translator are written to the file and library specified in the OUTFILE option, and the member

specified in the OUTMBR option, of the CRTICSC CL command. By default, file QACYCICS in the QTEMP library is used.

In your translated source code, CICS statements have been converted to a comment, assignments, and a function call to the CICS interface module AEGEIPGM. The name of the temporary source file is given in the OUTMBR option of the CRTICSC CL command. When CICS creates the QACYCICS file, it uses the coded character-set identifier (CCSID) value of the source file as the CCSID value for QACYCICS.

Example

For an ILE C application program, each command is replaced by one or more assignment statements and function calls. For example, a command such as:

```
EXEC CICS RECEIVE MAP("MAPA");
```

may be translated to:

```
/*  
EXEC CICS RECEIVE MAP("MAPA");  
*/  
{\  
  cpyblap(dfhttftr.ArgData[0].StringValue,16, \  
    "MAPA", (short)7,' '); \  
  dfhttftr.ArgIndx[5] = 0; \  
  dfhttftr.ArgIndx[1].DataArea = &mapa.mapai; \  
  dfhttftr.ArgIndx[8] = 1; \  
  dfhttftr.ArgMask[0] = 0x00008120; \  
  dfhttftr.ArgMask[1] = 0x00000000; \  
  dfhttftr.ArgMask[2] = 0x00000000; \  
  dfhttftr.ArgMask[3] = 0x00000000; \  
  dfhttftr.ArgCount = 3; \  
  dfhttftr.FnCode = 62; \  
  dfhttftr.DebugLine = -1; \  
  AEGEIPGM(&dfhttftr); \  
}
```

Compiling an application program

Having preprocessed your program source code to resolve **#include** and **#define** statements, and translated it to remove any CICS and SQL statements, you next compile the program to create a program module. If the program is a simple one containing only one module, you can create the module and bind it in one step.

Unless the *NOGEN option is selected on the CICSOPT parameter list, the CICS precompiler calls the CRTMOD CL command to invoke the ILE C compiler and create a module. For simple, one-module programs you may specify OBJTYPE(*PGM), in which case the CRTPGM CL command is called to create a bound program. The program is bound to the CICS interface service program, AEGEIPGM. When preparing more complex applications, specify OBJTYPE(*MODULE). Then you can bind the compiled module with other modules or to service programs as required, using the CRTPGM or CRTSRVPGM CL commands.

Note: You must specify the value QCICS/AEGEIPGM in the BNDSRVPGM parameter list to resolve the static procedure calls made to the CICS application interface routines.

The following options under CICSOPT are passed to the ILE C compiler:

- *SRC, *NOSRC
- *SOURCE, *NOSOURCE
- *SECLVL, *NOSECLVL

(See “Conventions and terminology used in this book” on page xiii for a discussion of the formation of these option names.)

The REPLACE option is passed onto the CRTSQLCI, CRTCMOD, and CRTBNDC commands. See page 266 for an explanation of the limitations on the SQL options that the CRTICSC CL command can pass to the SQL preprocessor.

Note: You must not change the translated source member before compilation or the compilation may fail.

Writing CICS programs in ILE C

CICS/400 Version 5 supports ILE C programs. You can define and use both ILE C, ILE COBOL and COBOL/400 programs in a CICS/400 control region.

ILE C programs are normally coded as members of source type CICSC in source file QCSRC. See Figure 16 for an example of using source type CICSC.

```

Work with Members Using PDM                               AS4001
File . . . . . QCSRC
Library . . . . . QCICSSAMP                               Position to . . . . .

Type options, press Enter.
2=Edit      3=Copy  4=Delete 5=Display    6=Print    7=Rename
8=Display description  9=Save 13=Change text 14=Compile 15=Create module...

Opt  Member      Type      Text
    DFH$DALL     CICSC    FILEA Inquiry/Update program
    DFH$DBRW     CICSC    FILEA Browse program
    DFH$DCOM     CICSC    FILEA Order entry queue print program
    DFH$DMNU     CICSC    FILEA Operator instruction program
    DFH$DREN     CICSC    FILEA Order entry program
    DFH$DREP     CICSC    FILEA Low balance report program

                                                    Bottom

Parameters or command
===>
F3=Exit      F4=Prompt    F5=Refresh    F6=Create
F9=Retrieve   F10=Command entry  F23=More options  F24=More keys
This is a subsetted list.

```

Figure 16. An example of using source type QCSRC

See the *ILE Concepts* manual for information about ILE and the benefits of running applications under ILE.

For more general information about writing ILE C programs, refer to the *WebSphere Development Studio: ILE C/C++ Language Reference*, *WebSphere Development Studio: ILE C/C++ Programmer’s Guide*, and *WebSphere Development Studio: ILE C/C++ Compiler Reference*.

Modular programming

This section describes some considerations for modular programming.

Use of #include

Source code containing CICS statements that is shared among many programs can be included in the programs using the **#include** statement. There are two methods of handling include files containing CICS statements:

1. Translate the include file before it is stored, but do not edit the translated output. If you copy or manipulate statements inserted by the CICS translator into an application program, the results may be unpredictable.
2. Specify the *PP option in the CICSOPT parameter of the CRTICSC CL command. The CICS precompiler calls the ILE C compiler preprocessor to resolve the **#include** statements in the source code before calling the CICS C translator to convert EXEC CICS commands into C statements.

Use of modules

A module is an ILE object that is created when OBJTYPE(*MODULE) is specified on the CRTICSC CL command. A module can be run only if it is bound into an ILE program or service program using the Create Program (CRTPGM) or Create Service Program (CRTSRVPGM) command. Several modules are usually bound together, but a module can be bound by itself.

A service program provides a means of packaging externally-supported callable routines (functions or procedures) into a separate object. Bound programs and other service programs can access these routines by resolving their imports to the exports provided by a service program. The connections to these services are made when the calling programs are created. This improves call performance to these routines without including the code in the calling program.

When the Create Program (CRTPGM) or Create Service Program (CRTSRVPGM) command is used to bind CICS modules into a program or service program object, the CICS service program AEGEIPGM must be included in the BNDSRVPGM option of the command.

Pointer-based addressing

CICS application programs need to access data from CICS internal areas using only the address passed to the program. Examples are:

- CICS areas such as the EIB, CWA, TWA, and TCTTE user area (TCTUA), obtained using the EXEC CICS ADDRESS command
- Input data, obtained by EXEC CICS commands such as READ and RECEIVE with the SET option

Example of using pointer variables

The use of these pointers is illustrated in Figure 17 on page 43.

EXEC CICS ADDRESS EIB

Following CICS conventions, the address of the EXEC Interface Block (EIB) is not passed as an argument to an ILE C **main()** function. In order to obtain the address of the EIB, an EXEC CICS ADDRESS EIB command is required at the beginning of each program that requires access to the EIB. This includes any program that codes EXEC CICS commands with RESP or RESP2 options.

EXEC CICS ADDRESS COMMAREA

The address of the communication area is also not passed as an argument to a ILE C `main()` function. This means that ILE C functions must use `ADDRESS COMMAREA` to obtain the address of the communication area.

```
struct filea_struct {
    unsigned char stat;
    unsigned char numb[6];
    unsigned char name[20];
    unsigned char addrx[20];
    unsigned char phone[8];
    unsigned char datex[8];
    unsigned char amount[8];
    unsigned char comment[9];
};

static struct filea_struct filea;
static signed short int filea_size = sizeof(filea);
static struct filea_struct *commarea; /* COMMAREA Structure */
static short int comlen; /* Length of COMMAREA */
int rcode; /* RESP value */
main(int argc, char *argv[])
{
    EXEC CICS ADDRESS EIB(dfheiptr); /* Address of EIB */
    EXEC CICS ADDRESS COMMAREA(commarea); /* Address of COMMAREA */
    :
    EXEC CICS READ UPDATE FILE("FILEA ") INTO(&filea)
    LENGTH(filea_size) RIDFLD(commarea->numb) RESP(rcode);
    switch (rcode)
    {
        case DFHRESP(NORMAL) :
            break;
        case DFHRESP(NOTFND) :
            Notfound();
        default:
            Errors();
    }
    :
    EXEC CICS REWRITE FILE("FILEA ") FROM(&filea)
    LENGTH(filea_size) RESP(rcode);
    switch(rcode) {
        case DFHRESP(NORMAL) :
            break;
        case DFHRESP(DUPREC) :
            DupRec();
        default:
            Errors();
    }
    :
    EXEC CICS RETURN TRANSID(dfheiptr->eibtrnid)
    COMMAREA(commarea) LENGTH(comlen);
}
```

Figure 17. Example of using pointer-based addressing in a ILE C program

EXEC CICS READ/REWRITE

Figure 17 shows how to obtain addressability to the data for pointer addressing. Note that ILE C always requires a `LENGTH` option on `EXEC CICS READ` and `WRITE` commands. After the `EXEC CICS READ` command, you can get the length of the record from the field named in the `LENGTH` option (in Figure 17,

`filea_size`). In the EXEC CICS REWRITE command, you must code a LENGTH option if you want to replace the updated record with a record of a different length.

Getting map set storage

If you specify `BASE=name` in a DFHMSD macro, the logical map contains only a variable declaration for a variable of type *mapstruct**.

You must acquire map storage dynamically using the EXEC CICS GETMAIN command. You may either release storage with the EXEC CICS FREEMAIN command or keep it until the task is complete. Early release helps you to optimize storage use, and can be useful in a long conversational transaction. See Chapter 12, “Introduction to basic mapping support (BMS),” on page 135 for further information about BMS.

Figure 18 on page 45 shows a method of processing BMS maps. The highlighted material describes the contents of the MAPSET1.H ILE C header file. MAPSET1 was defined as follows:

```
MAPSET1 DFHMSD TYPE=DSECT,  
          LANG=C,  
          STORAGE=AUTO,  
          MODE=INOUT
```

In this example, `STORAGE=AUTO` has been specified. Therefore, the MAPSET1.H logical map declares an automatic global variable of type *mapstruct*. You do not need to acquire or release map storage.

Passing arguments by value

In ILE C, arguments are normally passed *by value* rather than *by reference*. *By value* means that when you pass arguments to a function, for example:

```
function(arg1, arg2, arg3);
```

the ILE C compiler builds a data area containing copies of the arguments to be passed to the function.

Other languages, such as COBOL/400, pass their arguments *by reference*, which means that the compiler passes a list of addresses pointing to the arguments to be passed. This is the call interface supported by CICS.

In order to provide a more compatible interface to the OS environment, the ILE C compiler provides a `#pragma` (an implementation-defined instruction to the compiler) to influence the linkage conventions used when generating code. Accordingly, the translator generates the following line of code in the translated output:

```
#pragma arguments(DFHEIPGM,OS,nowiden)
```

This designates the function DFHEIPGM as having its arguments passed conforming to the OS linkage conventions. Non-address arguments are copied to temporary locations, and the address of the copy is passed to the CICS procedure DFHEIPGM. Arguments that are addresses or pointers are passed directly to DFHEIPGM.

When you send values from a ILE C program to CICS, the translator takes the necessary action to generate code that will result in an argument list of the correct format being passed to CICS. To pass it The translator prefixes arguments (where possible) with the & (address of) operator. This generates a pointer to the data item; the address of the data item is placed directly into the parameter list.

However, the translator often cannot determine the data type being passed and therefore relies on your passing a suitable argument.

```
#include <stdio.h>
#include "MAPSET1.H"
union {
    _Packed struct {
        char    dfhms1[12];
        short int msg1;
        char    msgf;
        char    msgi[39];
        short int key1;
        char    keyf;
        char    keyi[6];
    } dfhdgai;
    _Packed struct {
        char    dfhms2[12];
        short int dfhms3;
        char    msga;
        char    msgo[39];
        short int dfhms4;
        char    keya;
        char    keyo[6];
    } dfhdgao;
} dfhdga;
main()
{
    EXEC CICS SEND MAP("DFHDGA ") MAPSET("MAPSET1")
            MAPONLY ERASE;
    EXEC CICS RETURN;
}
```

Figure 18. Example of processing BMS maps in an ILE C program

For example, if the translator is presented with the following line of C source:

```
EXEC CICS RECEIVE INTO(myarea) LENGTH(mylen);
```

it generates this ILE C replacement code:

```
DFHEIPGM( .... ,myarea,&mylen);
```

This works, provided the variable myarea is a pointer (or equivalent, for example, the name of a character array). If myarea is defined as a structure, for example:

```
struct {
    char header_info[8];
    char the_message[80];
} myarea;
```

the compiler makes a copy of myarea into some temporary storage and passes the address of the copy in the parameter list. The variable myarea remains unchanged, that is, CICS updates the copy instead.

Table 3 on page 46 shows the rules that apply when passing values as arguments in EXEC CICS commands.

Table 3. Rules for passing values as arguments in EXEC CICS commands

Data type	Usage	Handling the argument
Character array	Data area (R) Data value (S)	Only the name of the array should be passed. ILE C generates a pointer to the data location.
Integer variables (halfword)	Data area (R)	The translator prefixes the variable with &.
	Data value (S)	The translator does nothing.
Integer variables (fullword)	Data area (R)	The translator prefixes the variable with &.
	Data value (S)	The translator does nothing.
Integer literals (halfword)	Data value (S)	The translator does nothing.
Integer literals (fullword)	Data value (S)	The translator does nothing.
Pointers	Data area (R) Data value (S)	The translator prefixes the variable with &.
Structures	Data area (R) Data value (S)	You should prefix the structure name with & so that CICS is passed the address of your structure and not the address of a copy. Failure to do this will result in the loss of any data passed back in the structure.
Note: With the SEND and RECEIVE MAP BMS commands, the translator interprets that a structure is being passed in the FROM and INTO options and automatically prefixes the name with an &.		
Character variables	Data area (R)	The translator prefixes the variable with &.
	Data value (S)	The translator does nothing.
Character literals	Data value (S)	The translator does nothing.

Note: “(R)” indicates “Receiver”, where data is being received from CICS; “(S)” indicates “Sender”, where data is being passed to CICS.

Thoroughly review your programs for correct pointer usage.

Exception handling

In ILE C applications, you cannot use the EXEC CICS commands related to nonstructured exception handling. The commands are:

- EXEC CICS HANDLE CONDITION option (with or without a label)
- EXEC CICS HANDLE AID option (with or without a label)
- EXEC CICS IGNORE CONDITION option
- EXEC CICS PUSH HANDLE
- EXEC CICS POP HANDLE
- EXEC CICS HANDLE ABEND LABEL

Use of these commands is diagnosed by the translator.

EXEC CICS HANDLE ABEND PROGRAM commands are allowed, but you cannot use EXEC CICS PUSH HANDLE or POP HANDLE commands to suspend the active abend exit program.

In an ILE C application, every EXEC CICS command which does not explicitly specify either the NOHANDLE or the RESP option is treated as if it had the NOHANDLE option specified. This means that the set of “system action”

transaction abends that result from a condition occurring but not being handled, is not possible in a ILE C application. Control always flows to the next instruction, and it is up to the application to test for a normal response. For more information on condition handling, see Chapter 6, “Dealing with exception conditions,” on page 87.

Data declarations needed for ILE C

The following data declarations are provided by CICS for ILE C in file QCICS/H:

- Execution interface block (EIB) definitions.
The DFHEIBLK.H C header file includes a definition of the EIB.
- BMS screen attributes definitions.
ILE C versions of the DFHBMSCA, DFHMSRCA, and DFHAID files are supplied by CICS, and may be included by the application programmer when using BMS.

The EIB declarations are enclosed by `#indef` and `#endif`, and are included in all translated files. The ILE C compiler ignores duplicated declarations.

Naming EIB fields

Within a ILE C application program, fields in the EIB are referred to in lowercase and fully qualified as, for example, `dfheiptr->eibtrnid`, in contrast to EIBTRNID as used in other programming languages.

Data types

The following mapping of data types is used:

- Halfword binary integers are defined as “short int”
- Fullword binary integers are defined as “long int”
- Single-character fields are defined as “unsigned char”
- Character strings are defined as “unsigned char” arrays

Source code considerations

If at all possible, you should use CICS commands to request operating system functions, rather than the equivalent ILE C statements. For example, use the EXEC CICS GETMAIN command rather than `malloc()`, because CICS will free the storage for you at the end of the task. If you have to use ILE C statements rather than CICS commands, it is your responsibility to do the inverse operation to clean up. For example, if you open a file, you must also close it.

The following restrictions apply to a ILE C program that is to be used as a CICS application program:

- You can use upper, lower, or mixed case for keywords and arguments on EXEC CICS commands, with the exception of the words EXEC CICS. A name, such as a file name or a transaction id, must be coded in the same case as the external definitions.
- Do not omit the LENGTH option from commands that require a LENGTH option (for example, READ, READNEXT, READPREV, and WRITE).
- All strings passed to CICS/400 commands from ILE C programs must be delimited with double quotation marks.

ILE C string manipulation functions are allowed, but they result in a null byte being appended as an end-of-string marker. This is not acceptable if the string is to be passed to CICS/400. In particular, names of temporary storage queues

(which may contain null characters) must be supplied as eight characters. CICS/400 does not perform any padding to fill out short queue names supplied by ILE C applications.

- Where CICS expects a fixed-length character string such as a program name, map name, or queue name, any literals passed must be padded with blanks up to the required length.

Calling programs and ILE procedures from ILE C

In a CICS system, there are two ways of transferring control to another program:

- Using EXEC CICS LINK and XCTL commands
- Using ILE C language calls

Using EXEC CICS commands

In a ILE C CICS application you can invoke CICS services to link or transfer control to an external program.

The calling program contains one of the following CICS commands:

- EXEC CICS LINK

```
EXEC CICS LINK PROGRAM("SUBPGM") ;  
EXEC CICS LINK PROGRAM(SubPgmPtr) ;
```
- EXEC CICS XCTL

```
EXEC CICS XCTL PROGRAM("PGMNAME") ;  
EXEC CICS XCTL PROGRAM(PgmPtr) ;
```

Note: The called program may be stated explicitly as a constant string literal within quotation marks or as a variable of type **char ***.

Figure 19 provides an overview of how control is transferred between programs when either the EXEC CICS LINK or the EXEC CICS XCTL (transfer control) command is used.

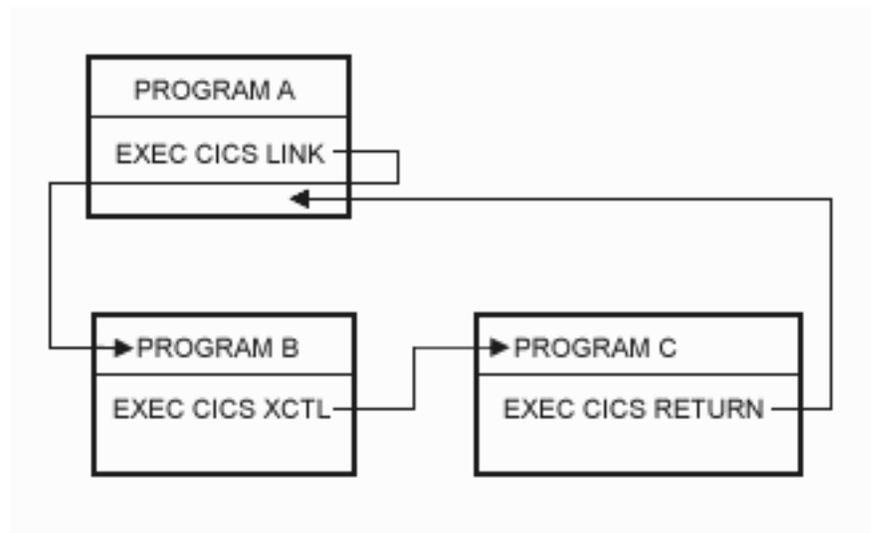


Figure 19. Control is returned to the next higher logical level.

The EXEC CICS LINK command transfers control to the specified program (B) at a new logical level. Program B could use the EXEC CICS RETURN command to cause control to return to the calling program (A). This is similar to the use of an ILE C dynamic program call.

The EXEC CICS XCTL command in Program B transfers control to Program C at the same logical level. You *cannot* return control to the calling program (B) using an EXEC CICS RETURN command or a host language statement. When Program C completes, control returns to Program A.

In a CICS system, when control is transferred from an active program to an external program, but the transferring program remains active and control can be returned to it, the program to which control is transferred is called a subprogram.

Using C language calls

In an ILE C application, there are three types of call that can be made during run time:

- Dynamic program calls
- Static procedure calls
- Procedure pointer calls

Dynamic program calls

A dynamic program call is a call made to a program object (*PGM). A call to an ILE C program or an OPM program are all examples of dynamic program calls.

If you have an ILE C program calling a program (*PGM) use the #pragma linkage (PGMNAME, OS) directive in your ILE C source to tell the compiler that PGMNAME is an external program, not a bound ILE procedure.

In contrast to static procedure calls, which are bound at compile time, symbols for dynamic program calls are resolved to addresses when the call is performed. As a result, a dynamic program call uses more system resources than a static procedure call.

Calling procedures

ILE C programs are called by dynamic program calls, but the procedures within an activated ILE C program can be accessed using static procedure calls or procedure pointer calls. ILE C programs that have not been activated yet must be called by a dynamic program call.

A call to an ILE procedure adds a new call stack entry to the bottom of the stack and passes control to the specified procedure. Examples include any of the following:

1. A call to a procedure in the same module
2. A call to a procedure in a different module in the same ILE program or service program
3. A call to a procedure that has been exported from an ILE service program

For a static procedure call, the called procedure must be bound to the calling procedure during binding. The call always accesses the same procedure. This contrasts with a call to a procedure through a pointer, where the target of the call can vary with each call.

The term procedure in ILE is equivalent to the term function in ILE C.

ILE C allows arguments to be passed between procedures that are written in different ILE high-level languages (HLLs). The calling function must make sure that the arguments are the size and type expected by the called function.

Procedure pointer calls

Procedure pointer calls provide a way to call a procedure dynamically. For example, by manipulating arrays, or tables, of procedure names or addresses, you can dynamically route a procedure call to different procedures.

Procedure pointer calls add entries to the call stack in exactly the same manner as static procedure calls. Any procedure that can be called using a static procedure call can also be called through a procedure pointer. If the called procedure is in the same activation group, the cost of a procedure pointer call is almost identical to the cost of a static procedure call. Procedure pointer calls can additionally access procedures in any ILE program that has been activated.

Note: Calls to bound procedures are not supported by the EXEC CICS LINK and XCTL commands. You must use static procedure or procedure pointer calls when calling bound procedures.

Rules governing calling CICS ILE C programs

Whether you use CICS commands or host language calls to call subprograms depends on a number of factors including functionality, portability, and performance. Table 4 outlines some of the requirements and conventions that you need to consider when choosing your implementation. This table defines requirements when a ILE C program is called as a main transaction program (called by CICS at logical level 1) or as a subprogram.

Table 4. Rules to be used with ILE C programs

Programming requirement	Main program	Subprogram called by EXEC CICS LINK/XCTL	Subprogram called by language call
Install in control region Processing Program Table (PPT).	REQUIRED Transaction abends APCT if not found or not authorized.	REQUIRED If not defined, not found, or disabled, a PGMIDERR condition is raised in the calling program.	OPTIONAL
Translation	REQUIRED Control is passed to the program entry point (PEP) of the called program.		OPTIONAL If translated and compiled with CRTICISC OBJTYPE(*MODULE), then bound into an ILE *PGM or *SRVPGM), may be called using a static procedure call.
Source code	Any ILE C function calls including precompiled EXEC SQL statements.		
Parameters and shared data	No automatic addressability to CICS control areas. Use EXEC CICS ADDRESS COMMAREA and ADDRESS EIB. When using language calls to call a translated CICS COBOL/400 program, the calling program must supply pointers to the EXEC Interface Block and the COMMAREA as parameters as expected.		If translated, may receive data passed by any of the standard CICS methods (COMMAREA, TCTUA, TS queues).

Table 4. Rules to be used with ILE C programs (continued)

Programming requirement	Main program	Subprogram called by EXEC CICS LINK/XCTL	Subprogram called by language call
Return control to the calling program	Use EXEC CICS RETURN Optionally, use <code>exit()</code> or <code>return()</code> from <code>main()</code> . Return codes specified by <code>exit()</code> or <code>return()</code> are stored in <code>dfheiptr->eibresp2</code> . Declare program as <code>int PGM(void)</code> .		Use <code>exit()</code> or <code>return()</code> . Use of EXEC CICS RETURN may give unpredictable results.
Program activation	Programs are activated according to the ACTGRP() attribute of the program as assigned at bind time.		
	Applications ported from other members of the CICS family normally require ACTGRP(*NEW) to ensure equivalent activation rules. See "Program activation."		Will normally be ACTGRP(*CALLER).
Exception handling	On entry to a new logical level, no abend handling is active. If an abend occurs while no higher abend handling is active, CICS searches successively higher logical levels (starting with the caller) and passes control to the program specified in the first active HANDLE ABEND PROGRAM found. If no active HANDLE ABEND PROGRAM is found, the transaction ends with that abend code. Note: EXEC CICS HANDLE AID and HANDLE CONDITION commands are not supported in CICS ILE C applications.		

Program activation

Activation is the process of preparing a program or service program to run. It includes allocating the data or static variables needed by the program. The necessary storage space is allocated from an *activation group*.

There are rules which govern when an ILE program is activated and deactivated. A program is activated the first time that it is called. A program may remain activated even when it completes normally and is no longer active on the call stack. On subsequent calls to an activated program, static storage will be in its 'last-used' state and reinitialization of any items becomes the responsibility of the programmer.

If a called program requires a new activation group - ACTGRP(*NEW) - or specifies a named activation group which has not been specified on any program previously activated in your OS/400 job, a new activation group will be started to manage allocation of storage and other data management facilities for the program. A program is only deactivated when its activation group ends.

The CICS/400 application developer can create ILE programs and service programs with activation group attributes of *NEW, *CALLER, or *username*. Use of activation groups will determine the run-time semantics of the CICS application. To provide equivalent semantics to those of other CICS platforms, you are recommended to compile with ACTGRP(*NEW) all programs that are defined in the PPT. This will ensure that static storage is in its first-use state on every call or link to the program and that recursive or reentrant calls are supported.

If your application design does not require static storage to be initialized on every call, you may use ACTGRP(*CALLER). This will result in improved performance. If you do not require support for a recursive program link, but do require static storage to be initialized on every LINK or XCTL to a program, you may specify the ACTGRP(*CALLER) and ALWRINZ(*YES) options on the OS/400 CRTPGM command. This use of the ALWRINZ option is peculiar to CICS/400 and allows you to gain the performance benefits of ACTGRP(*CALLER) with the static storage

reinitialization properties of standard CICS program activation after LINK or XCTL commands. Static storage of a program created with ALWRINZ(*YES) is not reinitialized if the program is invoked through a dynamic program call rather than through the CICS program control commands. Recursive links to a program compiled with ALWRINZ(*YES) may result in task abends or other unpredictable results.

For further information on controlling the effects of activation groups, refer to the *ILE Concepts* manual and the *WebSphere Development Studio: ILE C/C++ Programmer's Guide*.

Figure 20 on page 53 shows logical levels, activation groups, and the effects of using CICS commands and native C to transfer control between programs. The dashed boxes marked A, B, C, and D show the scope of activation groups.

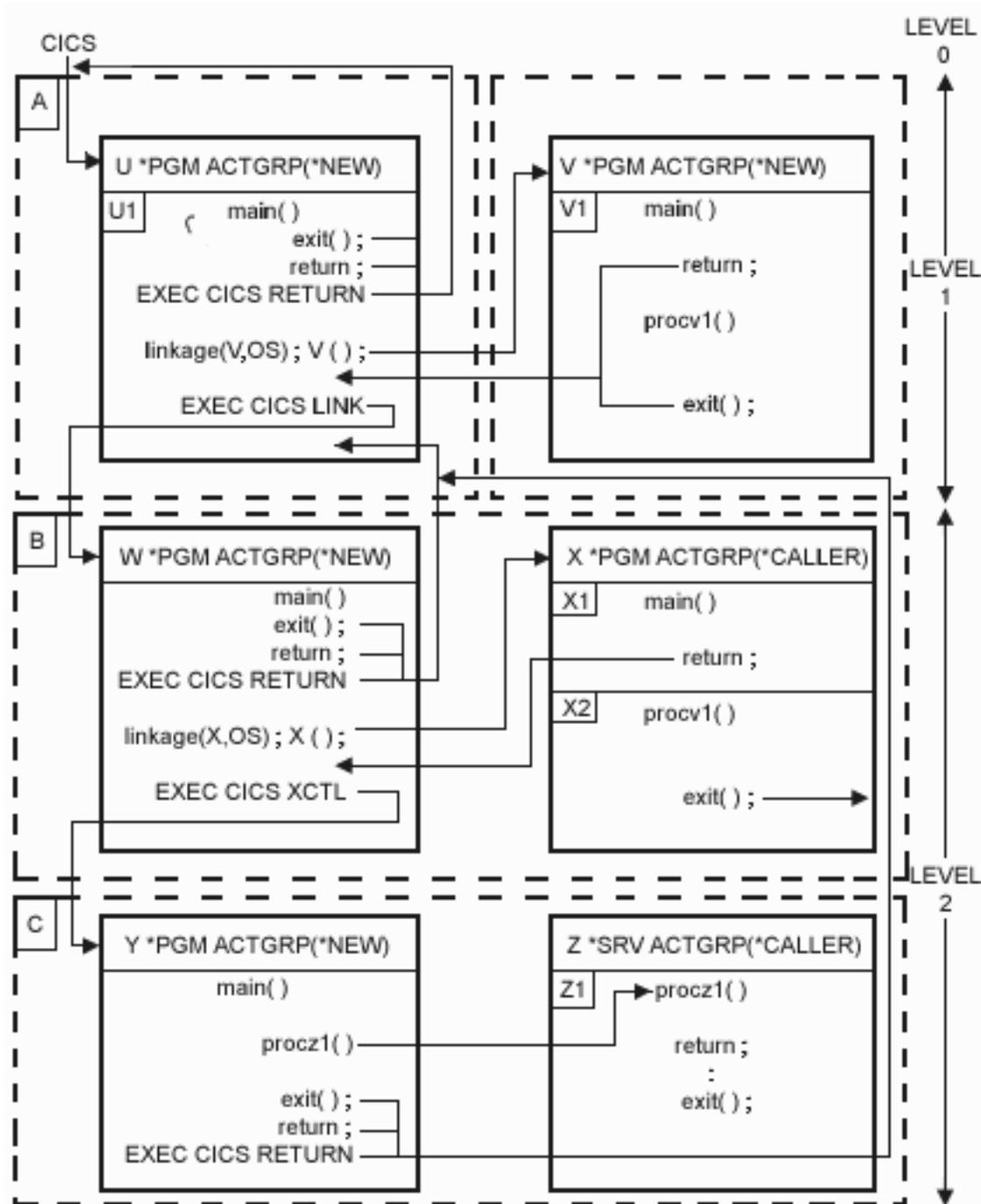


Figure 20. Flow of control between ILE C programs and activation groups in CICS

Sample application programs

Disclaimer:

The FILEA sample application contains programming source code for your consideration. This sample has not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, servcability, performance or function of the programs. All programs herein are provided to you "as is". IBM EXPRESSLY DISCLAIMS ALL WARRANTIES, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

A set of sample application programs, called FILEA, is provided as an example of how CICS commands can be used in a program written in the C language. All FILEA objects are in the QCICSSAMP library.

If the ILE C compiler is installed on your system then you can modify the FILEA sample programs. Figure 21 is a screen showing the members of file QCSRC that contain the source for the FILEA application programs. Note that the TYPE of the source members is CICSC.

```

Work with Members Using PDM                                AS4001
File . . . . . QCSRC
Library . . . . . QCICSSAMP          Position to . . . . .

Type options, press Enter.
2=Edit      3=Copy  4=Delete 5=Display    6=Print    7=Rename
8=Display description 9=Save 13=Change text 14=Compile 15=Create module...

Opt Member      Type      Text
DFH$DALL      CICSC    FILEA Inquiry/Update program
DFH$DBRW      CICSC    FILEA Browse program
DFH$DCOM      CICSC    FILEA Order entry queue print program
DFH$DMNU      CICSC    FILEA Operator intruction program
DFH$DREN      CICSC    FILEA Order entry program
DFH$DREP      CICSC    FILEA Low balance report program

Bottom

Parameters or command
===>
F3=Exit      F4=Prompt    F5=Refresh   F6=Create
F9=Retrieve   F10=Command entry F23=More options F24=More keys
This is a subsetted list.

```

Figure 21. Screen showing the members of file QCSRC containing the FILEA application programs

Figure 22 on page 55 is a screen showing the members of file QMAPSRC that contain the source for the BMS maps used in the FILEA sample. Note that the TYPE of the source members is CICSMAP.

```

Work with Members Using PDM
AS4001

File . . . . . QMAPSRC
Library . . . . . QCICSSAMP          Position to . . . . .

Type options, press Enter.
2=Edit      3=Copy  4=Delete 5=Display      6=Print      7=Rename
8=Display description 9=Save 13=Change text 14=Compile 15=Create module...

Opt Member      Type      Text
DFH$DGA      CICSMAP  Operator instruction map
DFH$DGB      CICSMAP  Record display/entry map
DFH$DGC      CICSMAP  Browse map
DFH$DGD      CICSMAP  Low balance report map
DFH$D GK     CICSMAP  Order entry map
DFH$DGL      CICSMAP  Order entry queue print map

Parameters or command
===>
F3=Exit      F4=Prompt      F5=Refresh      F6=Create
F9=Retrieve  F10=Command entry  F23=More options  F24=More keys
This is a subsetted list.

Bottom

```

Figure 22. Screen showing the members of file QMAPSRC containing FILEA sample maps

Data declarations used by the FILEA sample

- The data for each account for the FILEA sample is stored in a physical file, FILEA, in the QCICSSAMP library. The record format for this file is declared to the FILEA sample programs by including the header file DFH£DFIL.H. This header file is member DFH£DFIL in file H in library QCICSSAMP.
- The FILEA sample uses the L86O transient data queue for order processing. The data for each order is of a specific format and consists of several fields. This format is declared to the FILEA sample programs by including the header file DFH£DL86.H. This header file is member DFH£DL86 in file H in library QCICSSAMP.

The LOGA transient data queue also has an ILE C data structure associated with it, however this is declared in the source of the sample programs and hence no header file is required.

Defining resources for the FILEA sample

A sample program, CRTSAMP, is provided to simplify resource definition for the FILEA sample. This program creates a CICS group, called FILEA, and other necessary resources for a control region. You specify the name of the control region and the library in which the resources are created as parameters for the program. When the CICS resources have been defined, the control region is started.

To run the CRTSAMP program, enter:

```
CALL CRTSAMP library-name sys-id
```

where `sys-id` is the name of the sample control region to be created and `library-name` is the library that the CICS resources are stored in. These are required parameters. The source for CRTSAMP is in member CRTSAMP of file QCLSRC in library QCICSSAMP.

Running the FILEA sample

To run the FILEA sample:

1. Use CRTSAMP to create a control region and the CICS group, FILEA, that defines all the CICS resources needed to run the FILEA sample.
2. Start a CICS user shell for the control region created in step 1.

When the user shell has started, enter the transaction identifier DMNU to start the FILEA transaction.

The FILEA sample consists of six programs:

- The operator instruction sample program
- The browse sample program
- The inquiry and update sample program
- The low balance report sample program
- The order entry sample program
- The order entry queue print sample program

These samples are described in the following sections.

Operator instruction sample program

Program	Transaction Identifiers	BMS maps
DFH\$DMNU	DMNU	DFH\$DGA

The instruction program displays a map containing operator instructions. This map lists some of the FILEA sample application programs and the transaction identifiers that can be used to invoke them. To initiate the browse, add, update, or inquiry programs, the appropriate transaction identifier must be entered on the menu map.

Browse sample program

Program	Transaction Identifiers	BMS maps
DFH\$DBRW	DBRW	DFH\$DGA, DFH\$DGC

The browse program sequentially retrieves pages or sets of records for display, starting at a point in the FILEA data file specified by the operator. To start a browse, type the transaction identifier and the account number into the menu and press the Enter key. If the account number is omitted, browsing begins at the start of the file. Press the PF1 key, or type F and press the Enter key, to page forward through the data. Press the PF2 key, or type B and press the Enter key, to page backward through the data.

Inquiry and update sample program

Program	Transaction Identifiers	BMS maps
DFH\$DALL	DADD, DINQ, DUPD	DFH\$DGA, DFH\$DGB

The inquiry and update sample program lets you make an inquiry about, add to, or update records in a file. You can select one of these actions by entering the appropriate transaction identifier in the operator instruction menu.

To make an inquiry, enter the transaction identifier for the inquiry transaction, and an account number into the menu. The program maps in the account number and reads the record from the FILEA data file. The details of the requested record are displayed on the screen.

To add a record, enter the transaction identifier and the account number into the operator menu. A map is displayed with the title FILE ADD, the account number and a set of empty data fields. Once data has been entered, the addition is written to the FILEA data file and recorded on the LOGA transient data queue. The operator menu is then displayed with the message RECORD ADDED.

To update a record, enter the relevant transaction identifier and the account number into the menu. The program reads and displays the requested FILEA record. Once entered, the updated record is rewritten to the FILEA data file and the update is recorded on the LOGA transient data queue. The application then returns to the operator menu with the message RECORD UPDATED.

Low balance report sample program

Program	Transaction Identifiers	BMS maps
DFH\$DREP	DREP	DFH\$DGD

The low balance report sample program produces a report that lists all records in the FILEA data file for which the account balance is less than or equal to \$50.00. To run the report, enter the transaction identifier on the operator menu or on a clear screen. If an account number is specified, it is ignored.

Order entry sample program

Program	Transaction Identifiers	BMS maps
DFH\$DREN	DORD	DFH\$DGK

The order-entry sample program provides a data entry facility for customer orders for parts from a warehouse. Orders are recorded on a transient data queue that is defined so as to start the order entry queue print transaction automatically when a fixed number of orders have been accumulated. The queue print transaction sends the orders to a printer terminal at the warehouse. To begin order entry, type the transaction identifier on to a blank screen and press ENTER. The order entry program displays a map on the screen requesting the operator to enter order details. The customer number must be valid, that is, it must exist in the FILEA data file. The order details are mapped in and checked; an invalid order is redisplayed for correction. When valid, an order is written to the transient data queue L86O and the order entry screen is redisplayed ready for the next order to be entered. If CLEAR is pressed, the order entry program terminates. L86O, the name of the transient data queue, is also the name of the terminal where the order entry queue print transaction is to be triggered when the number of items on the queue reaches 30. The trigger level may be changed using the CEMT command as follows:

```
CEMT SET QUEUE(L86O) TRIGGER(n)
```

where *n* is the destination trigger level.

Order entry queue print sample program

Program	Transaction Identifiers	BMS maps
DFH\$DCOM	DORQ	DFH\$DGL

The order entry queue print sample program sends customer orders to a printer terminal at the warehouse. This program reads the transient data queue written to by the order entry sample program. The queue print transaction can be invoked in one of three ways:

- You can type the transaction identifier on to a clear screen. The program finds that the terminal identifier is not L86O and issues a START command to begin printing in one hour. The message PROCESSING COMPLETE is displayed and your terminal is available for other work.
- One hour after you enter this transaction identifier the queue print transaction is automatically invoked by CICS interval control. In this case the terminal identifier, specified by START, is L86O so the program prints the orders at the warehouse.
- The queue print transaction is started when the number of items (customer orders) on the transient data queue reaches 30. The trigger level is specified in the destination control table (DCT) entry for L86O. In this case, the terminal identifier is the same as the queue name (L86O) and the program prints the orders.

When invoked with a terminal identifier of L86O, the program reads each order, checks the customer's credit, and either prints the order at the warehouse or writes the rejected order to LOGA, the same transient data queue as used by the inquiry and update FILEA sample program. When all orders have been processed, or if there were no orders to process, the message ORDER QUEUE IS EMPTY is printed at the warehouse.

Part 2. Application design

Chapter 5. Designing efficient applications	61
Program size and structure	61
Choosing between pseudoconversational and conversational design	61
General programming techniques	63
Storage usage.	63
Minimizing memory requirements.	63
Processor usage	64
Recovery design implications	64
Terminal interruptibility	66
Summary of pseudoconversational and conversational design	66
Using resources effectively	66
Processor storage	66
Processor time	67
Exclusive-use resources	67
Line transmission capacity	67
Other suggestions	67
Auxiliary trace	67
Unnecessary commands	68
Resource retention	68
Data definition and manipulation considerations	68
Storing data within a transaction	68
Transaction work area (TWA)	69
User storage	69
COMMAREA in EXEC CICS LINK and EXEC CICS XCTL commands	70
Program storage.	70
Sharing data across transactions	70
Common work area (CWA)	71
TCTTE user area (TCTUA)	71
COMMAREA in EXEC CICS RETURN commands.	72
Display screen	72
Temporary storage	73
Intrapartition transient data	74
Your own files	74
Data operations	75
Emulating VSAM files.	75
Browsing files	76
Logging files	76
Sequential file access	76
Terminal operations	77
Data stream considerations	77
BMS considerations.	77
Avoid turning on modified data tags (MDTs) unnecessarily.	77
Use FRSET to reduce inbound traffic	78
Do not send blank fields to the screen	78
Use the MAPONLY option when possible	78
Send only changed fields to an existing screen	78
Design data entry operations to reduce line traffic	79
Compress data sent to the screen	79
Use nulls instead of blanks	79
Use methods that avoid the need for nulls or blanks	80
Sending messages to destinations other than the input terminal	80
Additional terminal control considerations	80
Use only one EXEC CICS SEND command per screen	80
Use the EXEC CICS CONVERSE command.	80
Avoid using unnecessary transactions	80
Send unformatted data without maps	81
Performance considerations	81
CICS and multiprocessor AS/400s.	81
CICS SIT parameters	81
CICS internal trace (INTTRCCTL) and auxiliary trace (AUXTRCCTL)	81
CICS files left open count and open timeout (FILECTL)	81
CICS interval control processing (ITVCTL)	82
BMS map set suffixing (DEVCTL)	82
COBOL application code	82
COBOL generation options	82
ILE C application code	83
ILE C generation options	83
*DEBUG or *NODEBUG	83
EXEC CICS LINK command or host language call	83
Terminal communication	84
BMS and terminal types	84
BMS DATAONLY option	84
Data stream compression.	84
Chapter 6. Dealing with exception conditions	87
Programs in any supported language.	87
How to use the RESP and RESP2 options	87
Example of coding and testing a RESP value	88
How to use the NOHANDLE option	90
COBOL programs only	91
How to use the EXEC CICS IGNORE CONDITION command	91
Passing control to a specified label	92
How to use the EXEC CICS HANDLE CONDITION condition command	92
How to use the EXEC CICS HANDLE CONDITION ERROR command	94
Relying on the system default action	95
How to use EXEC CICS PUSH HANDLE and POP HANDLE commands	95
How to use an EXEC CICS HANDLE CONDITION condition command	97
How CICS selects whether to take the system default action.	97
Mixing the methods	98
How CICS keeps track of what to do.	99
Chapter 7. Testing your application	101
Testing applications	101

Screen usage, checks and considerations	101
Types of problems	102
Levels of testing	103
Finding a problem in application code on a production system.	103
Chapter 8. Recovery considerations	105
CICS and OS/400 commitment control recovery	105
Defining recoverable files to CICS (an overview)	105
Syncpointing	106
User journaling.	106
Journal records	106
Journal output synchronization	107
Chapter 9. Abnormal termination recovery.	109
Creating a program-level abend exit	110
Restrictions on retrying operations	110
Trace	111
Trace entry points	112
System trace entry points	112
User trace entry points	112
Dump	112

Chapter 5. Designing efficient applications

In this chapter, design changes are suggested that can improve performance and efficiency without much change to the application program itself.

Program size and structure

The most efficient structure for a program is to have all the code in one program, with no subprograms. This structure can mean that programs are difficult to maintain. To overcome these potential problems, you might like to consider the following guidelines on program structure.

For modestly sized blocks of code that are processed sequentially, inline code is most efficient. The exceptions to this rule are blocks of code that are:

- Fairly long and used independently at several different points in the application
- Subject to frequent change (in which case, you balance the overhead of EXEC CICS LINK or EXEC CICS XCTL commands with ease of maintenance)
- Infrequently used, such as error recovery logic and code to handle uncommon data combinations

If you have a block of code that, for one of these reasons, has to be written as a subroutine, the best way of dealing with this from a performance viewpoint is to use a closed subroutine within the invoking program (for example, code that is dealt with by a PERFORM command in COBOL). If it is needed by other programs, it should be a separate program. A separate program can be invoked using either an EXEC CICS LINK or XCTL command, or by a host language call. Host language calls are more efficient than the EXEC CICS commands, but are functionally different in areas such as working storage initialization. See either “Calling programs from COBOL” on page 22 or “Calling programs and ILE procedures from ILE C” on page 48 for more information.

Choosing between pseudoconversational and conversational design

In a **conversational** transaction, the length of time spent in processing each of a user’s responses is extremely short when compared to the amount of time waiting for the input. A conversational transaction is one that involves more than one input from the terminal, so that the transaction and the user enter into a conversation. A **nonconversational** transaction has only one input (the one that causes the transaction to be invoked). It processes that input, responds to the terminal, and terminates.

Processor speeds are considerably faster than terminal transmission times, which, in turn, are considerably faster than user response times. This is especially true if users have to think about the entry or have to enter many characters of input. Consequently, conversational transactions tie up storage and other resources for much longer than nonconversational transactions.

A **pseudoconversational** transaction sequence contains a series of nonconversational transactions that look to the user like a single conversational transaction involving several screens of input. Each transaction in the sequence handles one input, sends back the response, and terminates.

Before a pseudoconversational transaction terminates, it can pass data forward to be used by the next transaction initiated from the same terminal, whenever that transaction arrives. A pseudoconversational transaction can specify what the next transaction is to be, and it does this by setting the transaction identifier of the transaction that handles the next input. However, you should be aware that if another transaction is started for that device, it may interrupt the pseudoconversational chain you have designed.

No transaction exists for the terminal from the time a response is written until the user sends the next input and CICS starts the next transaction to respond to it. Information that would normally be stored in the program between inputs is passed from one transaction in the sequence to the next using the COMMAREA or one of the other facilities that CICS provides for this purpose. (See “Sharing data across transactions” on page 70 for details.)

There are two major issues to consider in choosing between conversational and pseudoconversational programming. The effect of the transaction on **contention** resources, such as storage and processor usage. Storage is required for control blocks, data areas, and programs that make up a transaction, and the processor is required to start, process, and terminate tasks. Conversational programs have a *very* high impact on storage, because they last so long, relative to the sum of the transactions that would make up an equivalent pseudoconversational sequence. However, there is less processor overhead, because only one transaction is initiated instead of one for every input.

The second issue is the effect on **exclusive-use** resources, such as records in recoverable files, recoverable transient data queues, enqueue items, and so on. Again, a conversational transaction holds on to these resources for much longer than the corresponding sequence of nonconversational transactions.

CICS ensures that changes to recoverable resources (such as files, transient data, and temporary storage) made by a logical unit of work (LUW) are made completely or not at all. An LUW is equivalent to a transaction, unless that transaction issues EXEC CICS SYNCPOINT commands, in which case an LUW lasts between syncpoints.

When a transaction makes a change to a recoverable resource, CICS makes that resource unavailable to any other transaction that wants to change it until the original transaction has completed. In the case of a conversational transaction, the resources in question may be unavailable to other terminals for relatively long periods.

For example, if one user tries to update a particular record in a recoverable data set, and another user tries to do so before the first one finishes, the second user’s transaction is suspended. This has advantages and disadvantages. You would not want the second user to begin updating the record while the first user is changing it, because one of them would be working from what is about to become an obsolete version of the record, and these changes would erase the other user’s changes. On the other hand, you also do not want the second user to experience the long, unexplained wait that occurs when that transaction attempts to read for update the record that is being changed.

If you use pseudoconversational transactions, however, the resources are only briefly unavailable (that is, during the short component transactions). However, unless all recoverable resources can be updated in just one of these transactions,

recovery is impossible because LUWs cannot extend across transactions. So, if you cannot isolate updates to recoverable resources in this way, you must use conversational transactions.

The previous example poses a further problem for pseudoconversational transactions. Although you could confine all updating to the final transaction of the sequence, there is nothing to prevent a second user from beginning an update transaction against the same record while the first user is still entering changes. This means that you need additional application logic to ensure integrity. You can use some form of enqueueing, or you can have the transaction compare the original version of the record with the current version before actually applying the update.

There are factors other than performance overhead to consider when choosing between pseudoconversational and conversational design for CICS applications. The method you choose can affect how you write the application programs. You may need extra CICS requests for pseudoconversations, particularly if you are updating recoverable files. After you have done this, however, operational control (performance monitoring, capacity planning, recovery, system shutdown, and distributing system messages) may be much easier.

General programming techniques

Programming techniques can affect the performance and efficiency of the CICS system.

Storage usage

A truly conversational CICS task is one that converses with the terminal user for several or many interactions, by issuing a terminal read request after each write (for example, using either an EXEC CICS SEND command followed by an EXEC CICS RECEIVE command, or an EXEC CICS CONVERSE command). This means that the task spends most of its extended life waiting for the next input from the terminal user.

Any CICS task requires some storage throughout its life and, in a conversational task, some of this storage is carried over the periods when the task is waiting for terminal I/O. The storage areas involved include the TCA and associated task control blocks and the storage required for all programs that are in use when any terminal read request is issued. Also included are the work areas (such as copies of COBOL/400 working storage) associated with this task's use of those programs.

With careful design, you can sometimes arrange for only one very small program to be retained during the period of the conversation. The storage needed could be shared by other users. You must multiply the rest of the virtual storage requirement by the number of concurrent conversational sessions using that code.

By contrast, a pseudoconversational sequence of tasks requires almost all of its storage only for the period actually spent processing message pairs. Typically, this takes a period of 1–3 seconds in each minute (the rest being time waiting for operator input). The overall requirement for multiple concurrent users is thus perhaps five percent of that needed for conversational tasks. However, you should allow for data areas that are passed from each task to the next. This may be a COMMAREA of a few bytes or a large area of temporary storage.

Minimizing memory requirements

To improve performance, you should try to minimize the size of memory required. You can do this by:

- Writing modular programs and structuring the modules according to frequency and anticipated time of reference. Do not modularize merely for the sake of size; consider duplicate code inline as opposed to subroutines or separate modules.
- Using separate subprograms whenever the flow of the program suggests that execution will not be sequential.
- Not tying up memory awaiting a reply from a terminal user.
- Using command-level file control locate-mode input/output rather than move-mode. Use of multiple temporary storage queues is restricted. programs, using static storage for constant data.
- Avoiding the use of EXEC CICS LINK commands where possible, because they generate requests for memory and require additional processor time.
- Specifying constants as literals in the procedure division, rather than as data variables in the WORKING STORAGE SECTION. The reason for this is that there is a separate copy of working storage for every task executing the program, whereas literals are considered part of the program itself, of which only one copy is used in CICS.
- Using static storage in ILE C for data that is genuinely constant, for the same reason as in the point above. Static storage is the same between different invocations of the same transaction and is shared between them.
 - Reusing data areas in the program as much as possible. You can do this with the REDEFINES clause in COBOL and the union clause in ILE C. In particular, if you have a map set that uses only one map at a time, code the DFHMSD map set definition without specifying either the STORAGE=AUTO or the BASE operand. This allows the maps in the map set to redefine one another. Refer to “DFHMSD” on page 555 for more information.

Refer to data directly by:

- Avoiding long searches for data in tables
- Using data structures that can be addressed directly, such as arrays, rather than structures that must be searched, such as chain
- Avoiding methods that simulate indirect addressing
- Using binary or hash search algorithms

Processor usage

Pseudoconversational tasks require a new task to be created to process each message-pair, and to be deleted when that task has finished. These may include the cost of initializing a new work area for the program that is first entered. (In a conversational task, this area is retained permanently, as already mentioned.)

There may also be extra processor overhead because of extra requests needed to retrieve data passed from the previous task of the pseudoconversation, and possibly to pass data to the next task.

Recovery design implications

The fundamental and powerful recovery facilities that CICS provides have performance implications. CICS serializes updates to recoverable resources so that, if a transaction fails, its changes to those resources can be backed out independently of those made by any other transaction. Therefore, a transaction updating a recoverable resource gets control of that resource until it terminates or indicates with an EXEC CICS SYNCPOINT command that it wants to commit those changes. Other transactions requiring the same resource for update must wait until the first transaction finishes with it.

The primary resources that produce these enqueue delays are files, databases, temporary storage, and transient data queues. For transient data, the “read” end of the queue is considered a separate resource from the “write” end (that is, one transaction can read from a queue while another is writing to it, provided that the records are committed).

To reduce transaction delays from contention for resource ownership, the length of time between the claiming of the resource and its release (the end of the LUW) should be minimized. In particular, conversational transactions should not own a critical resource across a terminal read.

Note: Even for nonrecoverable files, CICS prevents two transactions from reading the same record for update at the same time. This enqueue ends as soon as the update is complete, rather than at the end of the LUW.

This protection scheme can also produce deadlocks as well as delays, unless specific conventions are observed. If two transactions update more than one recoverable resource, they must update the resources in the same order. If they each update two files, for example, file “A” should be updated before file “B” in all transactions. Similarly, if transactions update several records in a single file, they should always do so in some predictable order (low key to high, or conversely). Transient data and temporary storage must be included among such resources.

Many applications require a succession of interactions with the user to get all the data needed to create a file record. A conversational application might create a partial order record and then update it in stages, as the terminal operator enters items. If all the updates are to be committed and backed out together, this means retaining the protective enqueues on records throughout the conversation until the order is complete. You may need to protect both the current order being entered and the stock records that have been decreased by the number of items ordered. Thus, a whole series of enqueues could be carried forward through the conversation for several minutes, and any other user making a conflicting request might wait without warning until the end of the order. Lastly, if the conversation went on to another order, presumably a CICS syncpoint would be taken to commit the previous one.

In a pseudoconversational implementation, the above approach is not possible because updates on one task are committed independently of any other. Therefore, an order that must be complete “in one piece” must be created by just one task. However many interactions it takes to get all the necessary input, the final task has to be the one that creates the order. Data supplied earlier in the conversation must be saved somewhere between transactions—usually in temporary storage on disk. That is, you must incur extra overhead in I/O to TS while the order is built up.

If the operator is taking orders over the telephone, with no written backup material on paper, then maybe the TS data itself should be made recoverable to avoid the remote client having to dictate the order again later on.

To summarize the issue: recovery places separate design constraints on *both* conversational and pseudoconversational implementations, but the performance cost of the pseudoconversational approach is usually more acceptable.

Terminal interruptibility

When a conversational task is running, CICS allows nothing else to send messages to that task's terminal. This has advantages and disadvantages. The advantage is that unexpected messages (for example, broadcasts) cannot interrupt the user-machine dialogue and, worse, corrupt the formatted screen. The disadvantage is that the end user cannot then be informed of important information, such as the intention of the control operator to shut down CICS after 10 minutes. More importantly, the unwitting failure of the end user to terminate the conversation may in fact prevent or delay a normal CICS shutdown.

Pseudoconversational applications can allow messages to come through between message pairs of a conversation. This means that notices like shutdown warnings can be delivered. This might disturb the display screen contents, and can sometimes interfere with transaction sequences controlled by the EXEC CICS RETURN command with the TRANSID option. However, this can be prevented by forcing the terminal into NOATI status during the middle of a linked sequence of interactions (like building one order in the example above), or by judiciously allowing space at the top or bottom of the screen for use by any message being sent to the screen.

The main problem is that CICS shutdown could occur in mid sequence—in our example, when an order is only partly built. This is because CICS cannot distinguish between the last CICS task of a user transaction and any other. You can guard against this by ensuring that users are warned of any intended shutdown sufficiently far in advance, so they do not start work that they might not complete in time.

Summary of pseudoconversational and conversational design

Conversational tasks may be easier to write, and may require less processor time, but they have serious disadvantages in their need for more memory and in their effect on the overall operability of the CICS systems containing them.

Using resources effectively

This section gives advice about designing application programs to minimize the use of resources.

When you have decided which services you want an application to provide, you need to consider what resources you must conserve while providing these services. These are:

- Processor storage
- Processor time
- Exclusive-use resources (such as terminals, file records, and scratch-pad areas)
- Line transmission capacity

There will occasionally be conflict when trying to save one resource at the cost of another. The appropriate compromises will vary from one application to the next.

Processor storage

Your applications use up processor storage in two ways. First, there are the CICS control blocks associated with any transaction being processed. Second, there are the programs being executed to run the transaction. The programs, in turn, take up space both for executable code and for working storage areas. In an online system,

the storage needs for these purposes fluctuate, existing at most for the duration of a transaction. You should consider the following guidelines in your application programs:

- Keep programs short.
- Keep working storage short.
- Keep programs short in duration of use.
- Keep GOTOs to a minimum.
- Avoid long searches for data.
- Place subroutines near the code that calls them.

Processor time

Calls for operating system services take much longer than straight application code. This is true whether you are coding in CICS, where a call takes the form of a CICS command, or a programming language, where a call is implicit in your input-output statements (OPEN, READ, WRITE). Careful attention to CICS design can reduce processor time much more than fine tuning your code. In a similar manner you can also code a single input/output operation in a regular program.

Consider the following guidelines to use processor time effectively:

- Avoid unnecessary operating system services.
- Avoid excessively long calculations.

Exclusive-use resources

Some resources can be used by only one transaction at a time. An example of this is a file record, which CICS allows only one transaction to update at any one time. CICS does this by providing the enqueue mechanisms to prevent conflicts between transactions over such resources. When one user has update access to such a resource, everyone else who wants it has to wait. You should therefore minimize the duration of transactions that require exclusive-use of resources. For a more thorough discussion of enqueue, see Chapter 19, "Task control," on page 197.

Line transmission capacity

In an online system with terminals located a long way from the processor, signals carried over the network take time to reach their destination. Transmission time, especially over a congested line, may be a major component of the total response time.

To lessen the effect of this, you should avoid sending unnecessary data to and from screens.

For the most part, CICS does this for you automatically, using the terminal hardware features.

Other suggestions

Here are some other suggestions that can improve the performance of application code.

Auxiliary trace

Use auxiliary trace to review your application programs. For example, it can show up any obviously unnecessary code, such as a file browse from the beginning of a file instead of by key, too many or too large EXEC CICS GETMAIN commands,

failure to release storage when it is no longer needed, unintentional logic loops, and failure to unlock records held for exclusive control that are no longer needed.

Note: Any form of tracing incurs a significant performance overhead. Consider carefully whether you really need TRACE in a production system.

Unnecessary commands

Avoid unnecessary commands. For example, an EXEC CICS ASKTIME command may be unnecessary, because the start of transaction time in the EIB is adequate for most purposes.

Resource retention

Be aware that file control EXEC CICS READ UPDATE and EXEC CICS DELETE commands, and any activity on recoverable resources, imply a lock on the resource. Ensure that tasks cannot be deadlocked because of crossed locks—insist on a standard order that resources are acquired.

Where CICS recovery forces retention of resources to end-of-task (or to user syncpoint), reduce resource retention by acquiring as late as possible. That is, issue the EXEC CICS READ UPDATE command only after the completion of other work such as editing and table lookups. Similarly, release resources quickly by one of the following methods:

- Issue an EXEC CICS WRITE, an EXEC CICS REWRITE, an EXEC CICS DELETE or an EXEC CICS RETURN command.
- Use the EXEC CICS UNLOCK command if the update will not be completed.
- Issue an EXEC CICS SYNCPOINT command as soon as the data can be committed.

Data definition and manipulation considerations

Avoid browsing or updating too many records at one time.

Define data in its most readily usable form; that is, do not make array indexes in zoned decimal format.

Avoid repeated conversion of data. If the converted form is needed several times, use work fields to hold the data that has been converted.

Avoid repeated subscript references to the same source of data. It is better to transfer the data once to a work field, and refer to it there.

Storing data within a transaction

CICS provides a variety of facilities for storing data within and between transactions. Each one differs according to how available it leaves data to other programs within a transaction and to other transactions; in the way it is implemented; and in its overhead, recovery, and enqueueing characteristics.

Storage facilities that exist for the lifetime of a transaction include:

- Transaction work area (TWA)
- User storage (via an EXEC CICS GETMAIN command issued without the SHARED option)
- COMMAREA
- Program storage

All of these areas are main storage facilities and come from the same basic source—CICS storage services. However, program storage is allocated by the operating system. None of these storage areas is recoverable. They differ, however, in accessibility and duration, and therefore each meets a different set of storage needs.

Transaction work area (TWA)

The transaction work area (TWA) is allocated when a transaction is initiated. It lasts for the entire duration of the transaction, and is accessible to all local programs in the transaction. Any remote programs that are linked via a distributed program link command do not have access to the TWA of the client transaction. The size of the TWA is determined by the `TWASIZE` parameter in the `ADDCICSPCT CL` command. If this is entered as `TWASIZE=nnn`, then the TWA is always allocated, lasts for the entire duration of the transaction, and is accessible to all of the programs in a transaction. See the *CICS for iSeries Administration and Operations Guide* for more information about specifying the `TWASIZE`.

Processor overhead associated with using the TWA is minimal. You do not need an `EXEC CICS GETMAIN` command to access it, and you address it using a single `EXEC CICS ADDRESS` command.

The TWA is suitable for fairly small data storage requirements and for larger requirements that are both relatively fixed in size and are used more or less for the duration of the transaction. Because the TWA exists for the entire transaction, a large TWA size has much greater effect for conversational than for nonconversational transactions.

User storage

User storage is available to all the programs in a transaction, but some effort is required to pass it between programs using `EXEC CICS LINK` or `EXEC CICS XCTL` commands. Its size is not fixed, and it can be obtained (using `EXEC CICS GETMAIN` commands) just when the transaction requires it and returned as soon as it is not needed. Therefore, user storage is useful for large storage requirements that are variable in size or are shorter-lived than the transaction.

The `SHARED` option of the `EXEC CICS GETMAIN` command causes the acquired storage to be retained after the end of the task. The storage can be passed in the communication area from one task to the next at the same terminal. The first task returns the address of the communication area in the `COMMAREA` option of the `EXEC CICS RETURN` command. The second task accesses the address in the `COMMAREA` option of the `EXEC CICS ADDRESS` command. The second task does not necessarily have to issue the `EXEC CICS ADDRESS` command to get the address of the `COMMAREA`, since the `COMMAREA` address is automatically passed to the next program in a pseudoconversational sequence.

The amount of processor overhead involved in an `EXEC CICS GETMAIN` command means that you should not use it for a small amount of storage. You should use the TWA for the smaller amounts or group them together into a larger request. Although the storage acquired by an `EXEC CICS GETMAIN` command may be held somewhat longer when using combined requests, the processor overhead and the reference set size are both reduced.

COMMAREA in EXEC CICS LINK and EXEC CICS XCTL commands

A communication area (COMMAREA) is used to transfer information between two programs within a transaction or between two transactions from the same terminal. For information about using COMMAREA between transactions, see “COMMAREA in EXEC CICS RETURN commands” on page 72.

Information in COMMAREA is available only to the two participating programs, unless those programs take explicit steps to make the data available to other programs that may be invoked later in the transaction. When one program links to another, the COMMAREA may be any data area to which the linking program has access. It is often in the working storage or LINKAGE SECTION of that program. In this area, the linking program can both pass data to the program it is invoking and receive results from that program. When one program transfers control (using an EXEC CICS XCTL command) to another, CICS may copy the specified COMMAREA into a new area of storage, because the invoking program and its control blocks may no longer be available after it transfers control. In either case, the address of the area is passed to the program that is receiving control, and the CICS command-level interface sets up addressability. See “Program control commands” on page 259 for further information.

Program storage

CICS creates a separate copy of the variable area of a CICS command-level program for each transaction using the program. This area is known as **program storage**. This area is called the WORKING-STORAGE SECTION in COBOL/400 and automatic storage in ILE C. Like the TWA, this area is of fixed size and is allocated by CICS without you having to issue an EXEC CICS GETMAIN command. Unlike the TWA, however, this storage lasts only while the program is being run. This makes it useful for data areas that are not required outside the program and that are either small or, if large, are fixed in size and are required for all or most of the execution time of the program.

Sharing data across transactions

CICS facilities for sharing data across transactions include:

- Common work area (CWA)
- TCTTE user area (TCTUA)
- COMMAREA in EXEC CICS RETURN commands
- Display screen
- Temporary storage
- Intrapartition transient data
- Your own files

The last three items provide more flexibility and function than the other items in the list, and therefore involve somewhat more overhead. You can also use any of these methods within transactions.

With the exception of COMMAREA, the display screen, and the TCTUA, data stored in these facilities is available to any transaction in the system. Subject to resource-level security restrictions, any transaction may write to them and any transaction may read them.

Common work area (CWA)

The common work area (CWA) is a single control block that is allocated at system startup time and exists for the duration of that CICS session. The size is fixed, as specified in the `WRKARASIZE` parameter of the system initialization table. The CWA has the following characteristics:

- There is almost no overhead in storing or retrieving data from the CWA. Command-level programs must issue one `EXEC CICS ADDRESS` command to get the address of the area but, after that, they can access it directly.
- Data in the CWA is not recovered if a transaction or the system fails.
- CICS does not regulate use of the CWA. All programs in all applications that use the CWA must follow the same rules for shared use. These are usually set down by the system programmers, in cooperation with application developers, and require all programs to use the same “copy” module to describe the layout of the area.
- The CWA is especially suitable for small amounts of data, such as status information, that are read or updated frequently by multiple programs in an application.
- The CWA is not suitable for large-volume or short-lived data because it is always allocated.

You should ensure that data used in one transaction does not overlay data used in another. Do not exceed the length of the CWA.

TCTTE user area (TCTUA)

The TCTTE user area (TCTUA) is defined using the `USRARASIZE` parameter on the `ADDCICSTCT CL` command. Each entry in the TCT specifies whether the TCTUA is present and, if so, how long it is.

TCTUAs have the following characteristics in common with the CWA:

- Minimal processor overhead (only one `EXEC CICS ADDRESS` command needed)
- No recovery
- No regulation of use by CICS
- Fixed length
- Unsuitability for large-volume or short-lived data

Unlike the CWA, however, the TCTUA for a particular terminal is shared only among transactions using that terminal. It is therefore useful for storing small amounts of data of fairly standard length between a series of transactions in a pseudoconversational sequence. Another difference is that it is not necessarily permanently allocated, because the TCTUA only exists while the TCTTE is set up. For nonautoinstall terminals the TCTUA is allocated from system startup; for autoinstall terminals the TCTUA is allocated when the TCTTE is generated.

Using the TCTUA in this way does not require special discipline among using transactions, because data is always read by the transaction following the one that wrote it. However, if you use TCTUAs to store longer-term data (for example, terminal or operator information needed by an entire application), they require the same care as the CWA to ensure that data used in one transaction does not overlay data used in another. You should not exceed the length of the allocated TCTUA, because this produces a storage violation.

COMMAREA in EXEC CICS RETURN commands

COMMAREA is used to pass information between application programs. The pointer reference is set to the address of the communication area. If the communication area does not exist, the length of the COMMAREA in the EIB, EIBCALEN, is set to zero.

The COMMAREA option of the EXEC CICS RETURN command is designed specifically for passing data between successive transactions in a pseudoconversational sequence. It is implemented as a special form of user storage, although the EXEC interface, rather than the application program, manages COMMAREAs.

The COMMAREA is main storage allocated from the CICS nonshared user storage, and is pointed to by a TCTTE, for instance, between tasks of a pseudoconversational application. The COMMAREA is freed unless it is passed to the next task.

To summarize:

- It is not recoverable.
- It is not suitable for large amounts of data (because main storage is used, and it is held until the terminal user responds).
- As with using COMMAREA to transfer data between programs, it is available only to the first program in a transaction, unless that program explicitly passes the data or its address to succeeding programs.

Display screen

You can also store data between pseudoconversational transactions from a 3270 or 5250 display terminal on the display screen itself. For example, if users make errors in data that they are asked to enter on a screen, the transaction processing the input usually points out the errors on the screen (with highlights or messages), sets the next transaction identifier to point to itself (so that it processes the corrected input), and returns to CICS.

The transaction has two ways of using the *valid* data. It can save it (for example, in COMMAREA), and pass it on for the next time it is run. In this case, the transaction must merge the changed data on the screen with the data from previous entries. Alternatively, it can save the data on the screen by not turning off the modified data tags of the keyed fields.

Saving the data on the screen is easy to code, but has two limitations. Firstly, you must *not* use it with screens that contain large amounts of data if the likelihood of errors is high. This is because of the additional network traffic needed to resend the unchanged data. It does not apply to locally attached terminals.

Secondly, if the user presses the CLEAR key, the screen data is lost, and the transaction must be able to recover from this. You can avoid this by defining the CLEAR key to mean CANCEL or QUIT, if this is appropriate for the application concerned.

Data other than keyed data may also be stored on the screen. This data can be protected from changes (except those caused by CLEAR) and can be nondisplay, if necessary.

Temporary storage

Temporary storage is the primary CICS facility for storing data that must be available to multiple transactions.

Data items in temporary storage are kept in queues whose names are assigned dynamically by the program storing the data. A temporary storage queue containing multiple items can be thought of as a small file whose records can be addressed either sequentially or directly, by item number. If a queue contains only a single item, it can be thought of as a named scratch-pad area.

Temporary storage is implemented in two different ways. The one used for a particular queue is determined by what is specified on the command that creates the first item. Specifying the MAIN option means that the queue is kept in main storage, in space taken from the shared storage area. The AUXILIARY option means that the queue is written to a physical file. Whichever method you use, CICS maintains an index of items in main storage.

Both these methods have characteristics that you should bear in mind:

- Main temporary storage requires much more internal storage than auxiliary. In general, you should use it only for small queues that have short lifetimes or are accessed frequently. Auxiliary temporary storage is specifically designed for relatively large amounts of data that have a relatively long lifetime or are accessed infrequently.
- You can make queues in auxiliary storage recoverable, but not queues in main storage. Only one transaction at a time can update a recoverable temporary storage queue. So, if you choose to make queues recoverable, bear in mind the probability of enqueues.
- If a task tries to write to temporary storage and there is no space available, CICS normally suspends the task. The task is not resumed until another task frees the necessary space in storage or the auxiliary TS file. This can produce unexplained response delays, especially if the waiting task owns exclusive-use resources, in which case all other tasks needing those resources must also wait.
- It can be more efficient to use main temporary storage exclusively in low-volume systems that have no need for recovery.

The following points apply to temporary storage in general:

- You must use an EXEC CICS command every time data is written to or read from a temporary storage queue, and CICS must find or insert the data using its internal index. This means that the overhead for using main temporary storage is greater than for the CWA or TCTUA. With auxiliary storage there is usually file I/O, which increases overhead even more.
- You need not allocate temporary storage until it is required; you need keep it only as long as it is required, and the item size is not fixed until you issue the command that creates it. This makes it a good choice for relatively high-volume data and data that varies in length or duration.
- The fact that temporary storage queues can be named as they are created provides a powerful form of direct access to saved data. You can access scratch-pad areas for terminals, records, and so on, simply by including the terminal name or record key in the queue name.
- Resource-level protection for auxiliary temporary storage queues is provided by security authorization to physical files.

Intrapartition transient data

Intrapartition transient data has some characteristics in common with auxiliary temporary storage. (See “Sequential file access” on page 76 for information about *extrapartition* transient data.) Like temporary storage, intrapartition transient data consists of queues of data, kept together in a single file, with an index that CICS maintains in main storage.

You can use transient data for many of the purposes for which you would use auxiliary temporary storage, but there are some important differences:

- Transient data queue names must be defined in the destination control table (DCT) before CICS is started. You cannot define them arbitrarily at the time the data is created. Thus, transient data does not have the same random access characteristics as temporary storage.
- Transient data queues must be read sequentially. That is, after a transaction reads an item, that item is removed from the queue and is not available to any other transaction. In contrast, items in temporary storage queues may be read either sequentially or directly (by item number). They can be read any number of times and are not removed from the queue until the entire queue is purged. These two characteristics make transient data inappropriate for scratch-pad data but suitable for queued data such as audit trails and output to be printed. In fact, for data that is read sequentially once, transient data is preferable to temporary storage.
- Items in a temporary storage queue can be changed; items in transient data queues cannot.
- Transient data queues are always written to a file. (There is no form of transient data that corresponds to main temporary storage.)
- You can define transient data queues so that writing items to the queue causes a specific transaction to be initiated (for example, to process the queue). Temporary storage has nothing that corresponds to this “trigger” mechanism, although you may be able to use an EXEC CICS START command to perform a similar function.
- For recoverable transient data queues, uncommitted records can only be read by a second task when the writing task has completed a logical unit of work. For temporary storage queues, the writing task does not need to complete a logical unit of work before another task can read the records written.
- Because the commands for intrapartition and extrapartition transient data are identical, you can switch easily between the internal CICS facility (intrapartition) and an external file, described in “Sequential file access” on page 76. To do this, you need only change the DCT, not your application programs. Temporary storage has no corresponding function of this kind.

Your own files

You can also use your own files to save data between transactions. This method probably has the largest overhead in terms of instructions processed, buffers, control blocks, and user programming requirements, but does provide extra functions and flexibility.

Data operations

CICS supports:

- Emulation of VSAM files
- Browsing
- Logging
- Sequential data set access

Emulating VSAM files

The efficiency of database and file operations is an important factor in the performance of any CICS system.

To minimize contention delays using VSAM emulated files:

- Minimize the time that they are reserved for exclusive use. The **exclusive-use** enqueue is the way CICS and OS/400 prevent concurrent updates. It is held on the record level. The AS/400® also holds a lock on the file level while the file is opened. The hold for exclusive use ends when either the request is completed, at the next syncpoint, or when the task has completed, depending on whether or not the resource is recoverable. For nonrecoverable files, the exclusive use that starts with an EXEC CICS READ UPDATE command ends when OS/400 data management has completed the EXEC CICS REWRITE command. For recoverable files, the CICS exclusive use ends at either a syncpoint or end of task.

The table shows which requests require exclusive use and when that reservation terminates.

Table 5. Requests that require exclusive use and when the reservation terminates

Command	Released by
READ UPDATE	REWRITE/DELETE/UNLOCK/ SYNCPOINT
WRITE MASSINSERT ¹	Completion of WRITE
WRITE	Completion of WRITE
DELETE RIDFLD	Completion of DELETE

- Hold position in an emulated VSAM file for as short a time as possible. The table shows which commands hold position and when the hold is released.

Table 6. Commands that hold position and when the hold is released

Command	Released by
READ UPDATE	REWRITE/DELETE/UNLOCK/ SYNCPOINT
WRITE MASSINSERT ¹	Completion of WRITE
STARTBR	ENDBR

¹If a request is made to close a file (using SET™ FILE), the file is not closed until the task has terminated. For nonrecoverable files, EXEC CICS UNLOCK or EXEC CICS SYNCPOINT commands also close the file. For recoverable files, only the next EXEC CICS SYNCPOINT command also closes the file.

To minimize processor overhead in emulated VSAM files:

- Minimize the number of **open files** within a CICS shell. CICS opens files within each CICS shell when they are first referred to. Open processing on the iSeries is a relatively expensive operation in terms of processor resources and system storage requirements. CICS attempts to optimize the open processing by keeping files opened across task completion boundaries on the probability that a subsequent task will access the same file. The programmer should be aware that too many opened files may cause excessive demand of system storage and increase page faults.
- Ideally, you would want CICS to have the files used by an application already opened. However, if your users are executing a mixture of applications that access a multitude of files, you can reduce the number of files that CICS keeps open across tasks with the FILECTL system initialization parameter.
- Use **skip-sequential** processing if you are reading many nonadjacent records in sequence. (Skip-sequential processing begins with an EXEC CICS STARTBR command. Each record is retrieved with an EXEC CICS READNEXT command, but the key feedback area pointed to by the RIDFLD option is supplied with the key of the next requested record before the EXEC CICS READNEXT command is issued.)
- Use the GENERIC option on the EXEC CICS DELETE command when deleting a group of records in a KSDS whose keys start with a common character string. CICS internally optimizes an EXEC CICS DELETE GENERIC command.

Browsing files

CICS provides the ability to browse emulated VSAM files in an optimized manner that is very efficient. You should use the browse function when you are reading records sequentially.

Logging files

Both CICS and OS/400 provide options to log activity using either CICS journal control facilities or OS/400 journals against a file. Logging using CICS journal control facilities enables you to use the logged file for postprocessing, or you may want to log reads for security reasons. OS/400 journals are primarily used by OS/400 commitment control during syncpoint processing, thus providing data integrity for the file. You have to balance the need for postprocessing, security and data integrity, against the overhead of user journaling.

Sequential file access

CICS provides a number of different sequential processing options. Temporary storage and intrapartition transient data queues (already discussed in “Temporary storage” on page 73 and in “Intrapartition transient data” on page 74) are the most efficient to use, but they must be created and processed entirely within CICS. The following methods apply to sequential files that must be processed **externally** to CICS:

- Extrapartition transient data
- VSAM entry-sequenced files
- Journals

Extrapartition transient data is the CICS way of handling standard sequential files. It is the least efficient of the three forms of sequential support listed, because CICS has to synchronize access to the files. Moreover, extrapartition transient files are not recoverable.

Emulated VSAM ESDSs, on the other hand, are more efficient. CICS journals provide another good alternative to extrapartition transient data, although primarily for output files.

Journal records are written in a special format. Each record has a system prefix and an optional user-built prefix, and record length is variable. Journals are opened for output and many users can share one journal.

You can only use journals for output (online) while CICS is running. Reading records from a journal is possible only offline by means of a batch job.

Terminal operations

There are some design factors, related to communicating with terminals, that may affect performance.

Data stream considerations

Good screen design and effective use of display devices can significantly affect the number of bytes transmitted on a network link. It is particularly important to keep the number of bytes as small as possible because, in most cases, this is the slowest part of the path a transaction takes. The efficiency of the data stream therefore affects both response time and line usage.

BMS considerations

Basic Mapping Support (BMS) is discussed in detail in Chapter 13, “CICS/400 basic mapping support (BMS),” on page 141.

When building a formatted data stream with BMS, you should bear in mind, the factors described in the following sections.

Avoid turning on modified data tags (MDTs) unnecessarily

The MDT is the bit in the attribute byte that determines whether a field should be transmitted on a READ MODIFIED command (the command used by CICS for all but copy operations).

The MDT for a field is normally turned on by the 5250 or 3270 hardware when the user enters data into a field. However, you can also turn on the tag when you send a map to the screen, either by specifying FSET in the map or by sending an override attribute byte that has the tag on. You should never set the tag on in this way for a field that is constant in the map, or for a field that has no label (and therefore is not sent to the program that receives the map).

Also, you do not normally need to specify FSET for an ordinary input field. This is because, as already mentioned, the MDT is turned on automatically in any field in which the user enters data. The MDT is then received in the next EXEC CICS RECEIVE MAP command. These tags remain on, no matter how many times the screen is sent, until explicitly turned *off* by the program (by the FRSET, ERASEAUP, or ERASE option, or by an override attribute with the tag off).

You can store information, between inputs, that the user did not enter on the screen. This is an intended reason for turning the MDT on by a program. However, this storage technique is appropriate only to small amounts of data, and is more suitable for local than for remote terminals, because of the transmission overhead involved. For example, this technique is particularly useful for storing default values for input fields. In some applications, the user must complete a screen in

which some fields already contain default values. A user who does not want to change a default just skips that field. The program processing the input has to be informed what these defaults are. If they are always the same, they can be supplied as constants in the program. If they are variable, however, and depend on earlier inputs, you can simply save them on the screen by turning the MDT on with FSET in the map that writes the screen. The program reading the screen then receives the default value from a user who does not change the field and the new value from a user who does.

Note: The saved values are not returned to the screen if the CLEAR, PA1, PA2, or PA3 key is pressed.

Use FRSET to reduce inbound traffic

If you have a screen with many input fields, which you may have to read several times, you can reduce the length of the input data stream by specifying FRSET when you write back to the screen in preparation for the next read. FRSET turns off the MDTs, so that fields entered before that write are not present unless the user reenters them the next time. If you are dealing with a relatively full screen and a process where there may be a number of error cycles (or repeat transmissions for some other reason), this can be a substantial saving. However, because only *changed* fields are sent on subsequent reads, the program must save input from each cycle and merge the new data with the old. This is not necessary if you are not using FRSET, because the MDTs remain on, and all fields are sent regardless of when they were entered.

Do not send blank fields to the screen

Sending fields to the screen that consist entirely of blanks or that are filled out on the right by trailing blanks usually wastes line capacity. The only case in which BMS requires you to do this is when you need to erase a field on the screen that currently contains data, or to replace it with data shorter than that currently on the screen, without changing the rest of the screen.

This is because, when BMS builds the data stream representing your map, it includes blanks (X'40') but omits nulls (X'00'). This makes the output data stream shorter. BMS omits any field whose first data character is null, regardless of subsequent characters in the field.

BMS requires you to initialize to nulls any area to be used to build a map. This is done by moving nulls (X'00') to the mapnameO field in the symbolic map structure. See "Output map data structures" on page 153. BMS uses nulls in attribute positions and in the first position of data to indicate that no change is to be made to the value in the map. If you are reusing a map area in a program or in a TIOA, you should take special care to clear it in this way.

Use the MAPONLY option when possible

The MAPONLY option sends only the *constant* data in a map, and does not merge any variable data from the program. The resulting data stream is not always shorter, but the operation has a shorter path length in BMS. When you send a skeleton screen to be used for data entry, you can often use MAPONLY.

Send only changed fields to an existing screen

Sending only changed fields is important when, for example, a message is added to the screen, or one or two fields on an input screen are highlighted to show errors. In these situations, you should use the DATAONLY option to send a map that consists of nulls except for the changed fields. For fields in which only the attribute byte has changed, you need send only that byte, and send the remaining

fields as nulls. BMS uses this input to build a data stream consisting of only the fields in question, and all other fields on the screen remain unchanged.

For example, when a program that is checking an input screen for errors finds one, there are two options. It can simply add the error information to the input map (highlighted attributes, error messages, and so on) and resend it, or it can build an entirely new screen, consisting of just the error and message fields. The former is slightly easier to code (you do not need to have two map areas or move any fields), but it may result in much longer transmissions because the output data stream contains the correct input fields as well as the error and message fields. In fact, it may even be longer than the original input stream because, if there were empty or short fields in the input, BMS may have replaced the missing characters with blanks or zeros.

For 5250 terminals, additional processing is required to support the DATAONLY option. In a network of predominantly 5250 devices, you may want to avoid this option.

With the 3270 hardware, if the input stream for a terminal exceeds 256 bytes, the terminal control unit automatically breaks it up into separate transmissions of 256 bytes maximum. This means that a long input stream may require several physical I/O operations. Although this is not apparent to the application program, it does cause additional line and processor overhead. The *output* stream is generally sent in a single transmission.

Design data entry operations to reduce line traffic

Often, users are required to complete the same screen several times. Only the data changes on each cycle; the titles, field labels, instructions, and so on remain unchanged. In this situation, when an entry is accepted and processed, you can respond with an EXEC CICS SEND CONTROL ERASEAUP command (or a map that contains only a short confirmation message and specifies the ERASEAUP option). This causes all of the **unprotected** fields on the screen (that is, all of the input data from the last entry) to be erased and to have their MDTs reset. The labels and other text, which are in protected fields, are unchanged, the screen is ready for the next data-entry cycle, and only the necessary data has been sent.

Compress data sent to the screen

When you send unformatted data to the screen, or create a formatted screen outside BMS, you can compress the data further by inserting set buffer address (SBA) and repeat-to-address (RA) orders into the data stream. SBA allows you to position data on the screen, and RA causes the character following it to be generated from the current point in the buffer until a specified ending address. SBA is useful whenever there are substantial unused areas on the screen that are followed by data. RA is useful when there are long sequences of the same character, such as blanks or dashes, on the screen. However, you should note that the speed with which RA processes is not uniform across all display devices. You should check how it applies to your configuration before using it.

Use nulls instead of blanks

You should note that, outside BMS, nulls have no special significance in an *output* data stream. If you need a blank area on a screen, you can send either blanks or nulls to it; they take up the same space in the output stream. However, if the blank field is likely to be changed by the user and subsequently read, use nulls, because they are not transmitted back.

Use methods that avoid the need for nulls or blanks

For any *large* area of a screen that needs to be blank, you should consider methods other than transmitting blanks or nulls; for example, using BMS, putting SBA and RA orders directly into the data stream, or using the ERASE and ERASEAUP options.

Sending messages to destinations other than the input terminal

You have a choice of two other methods of delivering output to a terminal not associated with the transaction.

1. You can use an EXEC CICS START command, with the TERMID option, to specify the terminal to which you want to write and the FROM option to specify the data you want to send. The started transaction issues an EXEC CICS RETRIEVE command for the message and then sends it to its own terminal.
2. Similarly, you can put messages destined for a particular terminal on a transient data queue that has the same name as the terminal. The DCT entry for the queue must specify DEVTYPE(*TERMINAL), a trigger level (usually 1), and a transaction. Your own transaction reads the transient data queue and sends the message to its terminal. It repeats this sequence until the queue is empty, and then terminates. The trigger level you specified means that it is invoked every time messages are placed on the queue.

Note: This requires coding a DCT entry for each possible terminal destination, with the resulting storage overhead.

Additional terminal control considerations

The following sections describe additional points to consider when using the CICS terminal control services.

Use only one EXEC CICS SEND command per screen

It is important to send the screen in a single physical output to the terminal. It is *very* inefficient to build a screen in parts and send each part with a separate command, because of the additional processor overhead of using several commands and the additional line and access method overhead. If your program is pseudoconversational, it has only one EXEC CICS SEND command, by definition. (See “Choosing between pseudoconversational and conversational design” on page 61) Unless you require notification to this program of an error on the EXEC CICS SEND command, omit the WAIT option. This allows CICS task control to reclaim the control blocks and user storage for your program long before it would otherwise be able to do so. Use of the WAIT option reduces substantially the savings effected by pseudoconversational programming.

Use the EXEC CICS CONVERSE command

Use the EXEC CICS CONVERSE command rather than the EXEC CICS SEND and EXEC CICS RECEIVE commands (or an EXEC CICS SEND, EXEC CICS WAIT, EXEC CICS RECEIVE command sequence if your program is conversational). They are functionally equivalent, but the EXEC CICS CONVERSE command crosses the CICS services interface only once, which saves processor time.

Avoid using unnecessary transactions

Avoid situations that may cause users to enter an incorrect transaction or to use the CLEAR key unnecessarily, thus adding to terminal input, task control processing, terminal output, and overhead. Good screen design and standardized PF and PA key assignments should minimize this.

Send unformatted data without maps

If your output to a terminal is entirely or even mostly unformatted, you can send it using terminal control commands rather than BMS commands (that is, using an EXEC CICS SEND command without the MAP or TEXT options). However, these commands can be less efficient when using 5250 terminals. See “Terminal communication” on page 84.

Performance considerations

This section covers general performance considerations for CICS, including advice relating specifically to CICS in the AS/400 environment. Where this affects application design, you should also consider any possible effects on the portability of the program to CICS in other environments.

CICS and multiprocessor AS/400s

The CICS/400 architecture is generally able to exploit each of the processors in a multiprocessor AS/400 without the need for multiple CICS regions. This is because the CICS/400 architecture uses individual AS/400 jobs for user shells and the dispatch of the jobs is controlled by internal OS/400 dispatching mechanisms.

A simplified explanation follows. An *individual task* is likely to be executed on a specific processor due to internal affinities. Another task running from the same user shell could execute on a different processor. The choice of which processor is dispatched to run the task is controlled by the operating system, not CICS, and uses a combination of priority, eligibility, and cache affinity. Cache affinity is used to dispatch tasks to the processor on which they are most likely to have residual data in cache and, therefore, the processor on which they are likely to experience the best performance.

CICS SIT parameters

There are performance considerations that relate to specific SIT parameter settings.

CICS internal trace (INTTRCCTL) and auxiliary trace (AUXTRCCTL)

CICS trace is a powerful debugging aid, but there is a significant performance overhead incurred when trace is enabled. This is because of the additional processor time required to generate and record trace data.

The CICS supplied defaults have trace switched off, and it is recommended that it is left off in stable production systems for best performance. If you experience problems that you believe are caused by IBM code however, you will need to provide debugging information which will include CICS traces.

The overhead of CICS auxiliary tracing is greater than that for an internal trace.

CICS files left open count and open timeout (FILECTL)

In the SIT parameter FILECTL:

Element 1 controls the maximum number of files left open to a user shell at task termination.

Element 2 controls how long a file can be kept open by a shell without being referenced.

Each user shell separately opens and closes files, unlike mainframe CICS where the single CICS job opens the files on behalf of all the tasks in the system.

Opening, and to a lesser extent, closing files is a relatively expensive operation in terms of processor cycles. The default setting for this parameter is therefore zero, which means that CICS keeps files open for the user shell once they have been referenced. They remain open until the shell shuts down or the open inactivity limit expires.

Leaving files open saves processor time in opening and closing files, but increases memory use in relation to the number of open files.

The impact on memory use depends on the configuration of your AS/400 and the nature of your application. As an initial estimate, you might regard up to 20 open files per user shell as a threshold. For larger numbers of files, consider using the maximum file open and time out close parameters in conjunction.

Setting the maximum number of open files too low could cause additional opening and closing of files, which then increases the processor use and response time of the transaction.

Note: CICS allows a task to access more than the maximum number of files defined in the first FILECTL parameter during the processing of that task. A check is made at task termination to see whether the maximum is exceeded; if it has, those files least recently used will be closed. Any files opened to the shell, including those for temporary storage and transient data are eligible for automatic closure.

CICS interval control processing (ITVCTL)

In the interval control parameter (ITVCTL):

Element 1 controls the maximum number of batch shells concurrently active for interval control.

Element 2 controls how many batch shells will be kept active when there is no ongoing interval control processing.

Starting and stopping the batch shells used for interval control processing uses relatively high amounts of processor and elapsed time. To get good response time and efficient processing, you should ensure that sufficient shells exist and are available for reuse.

BMS map set suffixing (DEVCTL)

In the SIT parameter DEVCTL:

Element 3 is set to *DDS by default, enabling map set suffixing when suffixed map sets are in use. When map sets are not suffixed, you can improve performance by setting this element to *NODDS as unnecessary searches are avoided.

COBOL application code

There are performance considerations that relate to the design and implementation of COBOL applications.

COBOL generation options

The performance of a CICS application is influenced by how efficient your application code is, and how the program is compiled. The *COBOL/400 User's Guide* describes COBOL/400 compilation and generation options.

Note that, if the program contains SQL statements, the generation options are passed to the SQL precompiler but not to the COBOL compiler.

The CICS/400 CL command CRTICSCBL allows the use of two generation options that affect performance.

***NORANGE**

The default generation option used by the CICS/400 translator is *RANGE. This causes the compiler to generate code to check that subscripts are within range. For example, it ensures that you are not attempting to reference the 21st element of a 20-element array. If you specify the *NORANGE option this additional code is not generated. Specifying the *NORANGE option can give worthwhile performance improvements in programs with significant subscript activity,

***OPTIMIZE**

The default generation option used by the CICS/400 translator is *NOOPTIMIZE. This causes the compiler to perform only standard optimization for the program. If you specify the *OPTIMIZE option, the program object created may run more efficiently and may require less storage. However, specifying the *OPTIMIZE option can substantially increase the time required to compile a program, so you may decide to use this only for production programs.

See page 266 for the format of the CRTICSCBL command.

ILE C application code

There are performance considerations that relate to the design and implementation of C applications.

ILE C generation options

The CRTICSC CL command does not support the ILE C optimization parameter (OPTIMIZE) on the CRTCMOD CL command. You can, however, use the CHGMOD, CHGPGM, or CHGSRVPGM CL commands to select OPTIMIZE(*BASIC) or OPTIMIZE(*FULL) before using a module, program, or service program in a production environment.

See page 286 for the format of the CRTICSC command.

***DEBUG or *NODEBUG**

Setting the *DEBUG translator option could affect the performance of your program. The *NODEBUG translator option has no effect on the performance of your program.

EXEC CICS LINK command or host language call

There is a greater processor cost in using an EXEC CICS LINK command to pass control between programs than using a host language call. If your application makes extensive use of, for example, an interface program to access various different file access methods, you may wish to consider the use of a host language call rather than an EXEC CICS LINK command.

When using an EXEC CICS LINK command, the linked to program will have its working or automatic storage initialized every time it is invoked, but this will only occur on the first entry in a run time unit when a host language call is used. In COBOL/400 applications, this could mean you would need to change the application code to use MOVE statements in the procedure division to initialize fields rather than VALUE clauses in the WORKING STORAGE SECTION.

Terminal communication

These performance considerations relate to terminal communication.

BMS and terminal types

CICS/400 supports both 3270 and 5250 terminals, and there is logic within the Terminal Control and BMS functions to deal with the differing needs of these devices.

For BMS, CICS creates the appropriate 3270 or 5250 data stream to communicate with the attached device, when that device is connected to the local AS/400. Terminals attached to remote AS/400 or other processors are dealt with as 3270 devices, and data stream conversion is avoided.

The CICS logic for locally attached devices is designed to avoid data stream conversion wherever possible. The matrix of when data stream conversion is required is shown in Figure 23.

		CICS REQUEST	
		Terminal Control SEND	Basic Mapping Support SEND
Terminal type	3270	No data stream conversion	No data stream conversion
	5250	Needs conversion	No data stream conversion

Figure 23. Data stream conversion

In the case of terminal control, the CICS application sends a 3270 data stream, and CICS terminal control determines from the device characteristics of the destination unit whether conversion is required.

BMS DATAONLY option

3270 devices can benefit from reduced line traffic by use of the DATAONLY option of the EXEC CICS SEND MAP command.

5250 devices need additional processing to process a DATAONLY request in merging the new data with the existing screen. This option therefore incurs a processor overhead and may not significantly reduce line traffic.

Data stream compression

Depending on definitions in the system initialization table (SIT) and terminal control table (TCT), CICS can compress outbound data being sent to 3270 and 5250 devices. This is of particular relevance to terminals connected to the AS/400 by low-speed synchronous data link control (SDLC) links, where the outbound data stream is typically up to ten times longer than the inbound data stream.

It is possible to reduce the size of the outbound data stream by replacing repeated characters with a Repeat to Address (RA) order. This technique can substantially reduce the amount of outbound data and therefore give quicker transaction response time.

The **CICS/400 data stream compressor** causes all outbound CICS data streams to be parsed for repeating sequences of the same character, replacing them with equivalent RA orders. These are automatically expanded to the correct size at their destination as this is an integral part of data-stream architecture.

This facility is only effective for CICS generated data streams. You may also choose to use the **AS/400 3270 data stream optimizer**. This optimizes data streams generated by the AS/400 system through non-CICS activity. It uses similar methods to the CICS/400 compressor, and makes additional size reductions by maintaining details of screen images and avoiding unnecessary sending of unchanged data. Using this facility may cause an increase in processor activity.

The AS/400 data stream optimizer is only effective for AS/400 generated 3270 data streams targeted for remotely attached 3270's, Distributed Host Command Facility (DHCF) devices, Network Routing Facility (NRF) devices, and 3270 TELNET devices.

Chapter 6. Dealing with exception conditions

Every time you process an EXEC CICS command in one of your applications, CICS automatically raises a condition to tell you how the command worked. This condition (which is usually NORMAL) is passed back by the CICS EXEC interface program to your application. If something out of the ordinary happens, you get an “exception condition”, which simply means a condition other than NORMAL. By testing this condition, you can find out what has happened, and possibly why. Each condition has a name (such as LENGERR, for length error) and a matching number. There is a general condition named ERROR that you can use as a “catch-all” and whose default action is to terminate the task abnormally. There is also a NOTAUTH condition—a general condition that is raised when a resource security check on a command has failed.

Not all conditions denote an error situation, even if they are not NORMAL. (For example, if you get an ENDFILE condition on an EXEC CICS READNEXT command during a file browse, it might be exactly what you expect.) For information about all possible conditions and the commands on which they can occur, see Chapter 32, “Application programming commands - reference,” on page 323..

Programs in any supported language

When a condition is raised, you should let the program continue, with control coming straight back from CICS to whatever instruction in your program immediately follows the EXEC CICS command that has just been executed. You can then find out what happened by testing the RESP and RESP2 values. The result of this test enables you decide what to do next. For details, see “How to use the RESP and RESP2 options.”

All ILE C applications, and any new CICS applications written in COBOL, should use this method of handling conditions. It lends itself to structured code and removes the need for implied GOTOs that CICS required in the past.

You do not need to change any existing applications that are currently working perfectly well using one of the older methods of handling conditions.

How to use the RESP and RESP2 options

CICS sets return codes in the EXEC interface block (EIB), so you can test for particular conditions right after each CICS command, by executing the command and then immediately testing, for example, the RESP value to check whether it did what you wanted.

CICS makes it easy to test the RESP value by using a built-in function named DFHRESP. With this, your code can examine RESP values symbolically. This is a lot easier than looking at hexadecimal values that are less meaningful to someone reading the code.

Simply code the RESP option on your CICS command and follow it immediately with tests on the returned RESP value.

You can use the RESP option with any command to test whether an exception condition was raised during its processing. With some commands, a condition may be raised for more than one reason. If you have already specified a RESP option, then you can use the RESP2 option to determine exactly why a condition was raised.

RESP(xxx)

“xxx” is a user-defined fullword binary data area. On return from the command, it contains a value corresponding to the condition that may have been raised, or to a normal return, that is, xxx=DFHRESP(NORMAL). You can test this value using DFHRESP, as follows:

```
EXEC CICS WRITEQ TD FROM(abc)
        QUEUE(qname)
        RESP(xxx)
END-EXEC.

:
IF xxx=DFHRESP(QIDERR) THEN ...
```

For ILE C, the test is:

```
EXEC CICS WRITEQ TS FROM(abc)
        QUEUE(qname)
        RESP(xxx);

:
if (xxx==DFHRESP(NOSPACE))
{ ...
```

RESP2(yyy)

“yyy” is a user-defined fullword binary data area. On return from the command, it contains a value that further qualifies the response to certain commands. RESP2 values are given in the description of each command that returns them.

RESP2 values are included in the “Exception conditions” section of the command descriptions in Part 7, “Programming reference,” on page 263, where applicable. Where no RESP2(yyy) value is noted, the RESP2 field is reserved and the returned value is undefined.

Example of coding and testing a RESP value

Consider the section of COBOL code from program ACCT01 shown in Figure 24 on page 89:

```

* GET INPUT AND CHECK REQUEST TYPE FURTHER.
EXEC CICS RECEIVE MAP('ACCTMNU')
      MAPSET('ACCTSET')
      RESP(RESPONSE)
END-EXEC.
IF RESPONSE = DFHRESP(MAPFAIL)
  GO TO NO-MAP.
IF RESPONSE NOT = DFHRESP(NORMAL)
  GO TO OTHER-ERRORS.
/* Resume processing */
:
:
* PROCESSING FOR UNEXPECTED ERRORS.
OTHER-ERRORS.
MOVE EIBFN TO ERR-FN.
MOVE EIBRCODE TO ERR-RCODE.
MOVE EIBFN TO ERR-COMMAND.
MOVE EIBRESP TO ERR-RESP.
EXEC CICS HANDLE CONDITION ERROR END-EXEC.
EXEC CICS LINK PROGRAM('ACCT04')
      COMMAREA(COMMAREA-FOR-ACCT04)
      LENGTH(14) END-EXEC.
GOBACK.

```

Figure 24. An extract from COBOL program ACCT01

An equivalent piece of code in ILE C might be:

```

* Get input and check request type further.
EXEC CICS RECEIVE MAP("ACCTMNU")
      MAPSET("ACCTSET")
      RESP(RESPONSE);

switch(response)
{
case DFHRESP(MAPFAIL):
  /* Code to handle MAPFAIL condition */
:
:
  break;
case DFHRESP(NORMAL):
  break;
default:
  /* Code to handle any other condition */
:
:
EXEC CICS LINK PROGRAM("ACCT04")
      COMMAREA(COMMAREA-FOR-ACCT04)
      LENGTH(14);
  break;
}

```

In this example, the first thing to do after the EXEC CICS RECEIVE MAP command is to test the value CICS puts into RESPONSE to check whether it worked. You start by looking explicitly for condition MAPFAIL because it can occur without there being any serious error (if, for example, the user presses CLEAR at this point in the application) and you have to be able to recover. Note, however, that MAPFAIL is by no means the only condition that can arise on an EXEC CICS RECEIVE MAP command.

You can either test explicitly for all possible conditions after each command, or test for some subset of those conditions and somehow deal with all other possibilities

elsewhere in your program. (Usually, “somehow” means taking the system default action if all else fails. However, in most cases, because you are using the RESP option, you *must* make sure that you allow for all possible conditions somewhere in your code. This is because there is no system default in this case, because CICS returns straight to the application program—see “How CICS keeps track of what to do” on page 99.)

Here, the decision is that any value other than NORMAL is to be dealt with by a general error processing subroutine. In the COBOL example, this involves a branch to the paragraph at label OTHER-ERRORS. The code at paragraph OTHER-ERRORS is to catch *all* other conditions. ACCT01 picks up what information it can about what has happened, and then links to the error-handling program ACCT04 where a user abend is issued, and displays a final error message to the user.

Finally, with all condition testing out of the way, you can resume normal processing.

Specifying RESP on a command implies the NOHANDLE condition. See “How to use the NOHANDLE option” for information about NOHANDLE. For more information about the RESP and RESP2 options, see page 313.

How to use the NOHANDLE option

Note: The NOHANDLE option is implicit on all EXEC CICS commands in ILE C programs. The application code has to test for exception conditions and raise any abends.

You can code a NOHANDLE option on any command to ensure that no action is taken for any condition resulting from the execution of that command.

The NOHANDLE option suspends the error handling that was specified in previous EXEC CICS HANDLE CONDITION commands (or with the CICS defaults) but *only* for the command on which you put the NOHANDLE option. It has no effect on later commands, or on the error handling set by other EXEC CICS HANDLE commands.

To use the NOHANDLE option on an EXEC CICS RECEIVE MAP command, in ILE C you write:

```
EXEC CICS RECEIVE MAP("ACCTMNU")
          MAPSET("ACCTSET") NOHANDLE;
:
if (EIBRESP==DFHRESP(MAPFAIL))
{ ...
```

The equivalent in COBOL is:

```
EXEC CICS RECEIVE MAP('ACCTMNU')
      MAPSET('ACCTSET') NOHANDLE
END-EXEC.

:
IF EIBRESP=DFHRESP(MAPFAIL) THEN ...
```

Note: Using RESP implies NOHANDLE, so be careful when using RESP with the EXEC CICS RECEIVE command, because NOHANDLE overrides the EXEC CICS HANDLE AID command in addition to the EXEC CICS HANDLE CONDITION command. (This means that PF key responses are ignored, and is the reason for testing them earlier in the ACCT code.) See “The EXEC CICS HANDLE AID command” on page 163.

COBOL programs only

The methods described in this section may be used in COBOL programs only. For new applications you are recommended to use the methods described in “Programs in any supported language” on page 87, but for portability of existing applications between CICS platforms, you can:

1. **Let the program continue**, which means allowing control to return from CICS to whatever instruction in your program immediately follows the EXEC CICS command that has just been executed. You have three ways of doing this:
 - Put the RESP and RESP2 options on the command. For details, see “How to use the RESP and RESP2 options” on page 87.
 - Put the NOHANDLE option on the command. For details, see “How to use the NOHANDLE option” on page 90.
 - Use an EXEC CICS IGNORE CONDITION command. For details, see “How to use the EXEC CICS IGNORE CONDITION command” on page 91.
2. **Pass control to a specified label** if a named condition arises. You do this by using an EXEC CICS HANDLE CONDITION command to name both the condition and the label of a routine in your code to deal with it. For details, see “Passing control to a specified label” on page 92.
3. **Rely on the CICS system default action**, which is a perfectly sensible option in some cases, and means that you do nothing by way of testing or handling conditions. The default action is normally (but not always) to abend the task¹. For details, see “Relying on the system default action” on page 95.
4. **Mix the methods** in any way you choose. For details, see “Mixing the methods” on page 98.

How to use the EXEC CICS IGNORE CONDITION command

In COBOL, just as you can arrange for control to pass to a particular label for a specific condition with an EXEC CICS HANDLE CONDITION command, so you can have the program continue when a specific condition occurs. You do this by setting up an EXEC CICS IGNORE CONDITION command to ignore one or more of the conditions that can potentially arise on a command. The EXEC CICS IGNORE CONDITION command means that no action is to be taken if any of the conditions specified on the EXEC CICS IGNORE CONDITION command occur.

1. For the conditions ENQBUSY, NOSTG, QBUSY, and SYSBUSY, the default is to force the task to “wait” until the required resource, for example, storage, becomes available, and then resume processing the command. The NOSUSPEND option can be used to cause processing to resume immediately following the command.

Control returns to the instruction following the command and return codes are set in the EIB. The following example ignores the MAPFAIL condition:

```
EXEC CICS IGNORE CONDITION MAPFAIL  
END-EXEC.
```

While a single EXEC CICS command is being processed, it can raise several conditions. For example, you may have a file control command that is not only invalid but also applies to a file not defined in the file control table. CICS checks these and passes back to your application program the first one that is not ignored (by your EXEC CICS IGNORE CONDITION command). CICS passes back only one exception condition at a time to your application program.

An EXEC CICS IGNORE CONDITION command for a given condition applies only to the program you put it in, and it remains active while the program is running, or until a later EXEC CICS HANDLE CONDITION command naming the same condition is met, in which case the EXEC CICS IGNORE CONDITION command is overridden.

You can choose an EXEC CICS IGNORE CONDITION command if you have a program reading records that are sometimes longer than the space you provided, but you do not consider this an error and do not want anything done about it. You might, therefore, code EXEC CICS IGNORE CONDITION LENGERR before issuing your EXEC CICS READ commands.

You can also use an EXEC CICS IGNORE CONDITION ERROR command to enable the application to handle any condition considered as an error for which there is no currently active EXEC CICS HANDLE CONDITION command that includes a label. When an error occurs, control is passed to the next statement and it is up to the program to check for return codes in the EIB. See the footnote 3 on page 91 for examples of conditions that are **not** considered as errors.

You cannot code more than 16 conditions in the same command; the conditions must be separated by at least one space. You may specify additional conditions in further EXEC CICS IGNORE CONDITION commands.

Passing control to a specified label

You have two ways of passing control to a specified label:

1. Use an EXEC CICS HANDLE CONDITION condition(label) command, where condition is the name of an exception condition
2. Use an EXEC CICS HANDLE CONDITION ERROR(label) command

How to use the EXEC CICS HANDLE CONDITION condition command

You use the EXEC CICS HANDLE CONDITION condition command to specify the label to which control is to be passed if a condition occurs. You must include the name of the condition and, optionally, a label to which control is to be passed if the condition occurs. You must ensure that the EXEC CICS HANDLE CONDITION condition command is executed before the command that may give rise to the associated condition.

You cannot include more than 16 conditions in the same command; the conditions must be separated by at least one space. You may specify any additional conditions in further EXEC CICS HANDLE CONDITION commands. You can also use the

ERROR condition within the same list to specify that all other conditions are to cause control to be passed to the same label.

The EXEC CICS HANDLE CONDITION command for a given condition applies only to the program in which it is specified. The EXEC CICS HANDLE CONDITION command:

- Remains active while the program is running, or until:
 - An EXEC CICS IGNORE CONDITION command for the same condition is met, in which case the EXEC CICS HANDLE CONDITION command is overridden
 - Another EXEC CICS HANDLE CONDITION command for the same condition is met, in which case the new command overrides the previous one
- Is temporarily deactivated by the NOHANDLE or RESP and RESP2 options on a command

When control passes to another program, via an EXEC CICS LINK or EXEC CICS XCTL command, the EXEC CICS HANDLE CONDITION commands that were active in the calling program are deactivated. When control returns to a program from a program at a lower logical level, the EXEC CICS HANDLE CONDITION commands that were active in the higher-level program before control was transferred from it are reactivated, and those in the lower-level program are deactivated. (Refer to Chapter 20, “Program control,” on page 199 for information about logical levels.)

The following example shows you how to handle conditions, such as DUPREC, LENGERR, and so on, that can occur when you use an EXEC CICS WRITE command to add a record to a file. Suppose that you want DUPREC to be handled as a special case; that you want standard system action (that is, to terminate the task abnormally) to be taken for LENGERR; and that you want all other conditions to be handled by the error routine ERRHANDL. You would code:

```
EXEC CICS HANDLE CONDITION
      ERROR(ERRHANDL)
      DUPREC(DUPRTN) LENGERR
END-EXEC.
```

The *same* condition can arise, in some cases, on many different commands, and for a variety of reasons. For example, you can get an IOERR condition during file control operations, interval control operations, and others. One of your first tasks in the error handling routine, therefore, is to sort out *which command* has raised a particular condition; only when you have discovered that, can you begin to investigate why it has happened. This, for many programmers, is reason enough to start using the RESP option in their new CICS applications. (Although you need only one EXEC CICS HANDLE CONDITION command to set your error-handling for several conditions, it can sometimes be awkward to pinpoint exactly which of several EXEC CICS HANDLE CONDITION commands is currently active when a CICS command fails somewhere in your code.)

If you do not name the condition and it arises, you get the default action for it, unless this is to abend the task, in which case you get the ERROR condition. (If you do not name the ERROR condition either, the task will abend.) If you name the condition but leave out its label, any EXEC CICS HANDLE CONDITION command for that condition is deactivated, and you revert to the default action for it, if and when it occurs.

Bearing in mind the distinction between an error condition and a condition that merely causes a “wait” (see the footnote 3 on page 91), an EXEC CICS HANDLE CONDITION command is active after an EXEC CICS HANDLE CONDITION condition(label), or EXEC CICS HANDLE CONDITION ERROR(label) command has been run in your application.

The need to deal with *all* conditions can be a common source of errors when using the EXEC CICS HANDLE CONDITION command. Even if you then issue EXEC CICS HANDLE commands for all of these, you may not finish all the error-handling code adequately. The outcome is sometimes an error-handling routine that, by issuing an EXEC CICS RETURN command, allows incomplete or incorrect data changes to be committed.

The best approach is not to use the EXEC CICS HANDLE CONDITION command, but let the system default action take over if you cannot see an obvious way round a particular problem.

If you use EXEC CICS HANDLE CONDITION commands, or are maintaining an application that uses them, do not include any commands in your error routine that can cause the same condition that gave you the original branch to the routine, because you will cause a loop. Notice, also, that one EXEC CICS HANDLE CONDITION command can name up to 16 conditions, and that one of these could be the ERROR condition to deal with all remaining conditions.

Take special care not to cause a loop on the ERROR condition itself. You can avoid a loop by reverting temporarily to the system default action for the ERROR condition. Do this by coding an EXEC CICS HANDLE CONDITION ERROR command with no label specified. At the end of your error processing routine, you can reinstate your error action by including an EXEC CICS HANDLE CONDITION ERROR command with the appropriate label. If you know the previous EXEC CICS HANDLE CONDITION state, you can do this explicitly. In a general subroutine, which might be called from several different points in your code, the EXEC CICS PUSH HANDLE and EXEC CICS POP HANDLE commands may be useful—see “Relying on the system default action” on page 95.

How to use the EXEC CICS HANDLE CONDITION ERROR command

Figure 25 shows the first of only two EXEC CICS HANDLE CONDITION commands used in the sample program ACCT01:

```
PROCEDURE DIVISION.  
*  
*   INITIALIZE.  
*   TRAP ANY UNEXPECTED ERRORS.  
*   EXEC CICS HANDLE CONDITION  
*   ERROR(OTHER-ERRORS)  
*   END-EXEC.  
*
```

Figure 25. Trapping the unexpected with the EXEC CICS HANDLE CONDITION ERROR command

It passes control to the paragraph at label OTHER-ERRORS if any condition arises for a command that does not specify NOHANDLE or RESP.

The EXEC CICS HANDLE CONDITION ERROR command is the first command executed in the procedure division of this COBOL program. This is because an

EXEC CICS HANDLE CONDITION command must be processed before any CICS command is processed that can raise the condition being handled. Otherwise, the EXEC CICS HANDLE CONDITION command does not take effect. Note, however, that your program does not see the effects when it processes the EXEC CICS HANDLE CONDITION command; it only sees them later, if and when it issues a CICS command that actually raises one of the named conditions.

In this, and the other ACCT programs, you generally use the RESP option. All the commands specifying the RESP option have been written with a “catch-all” test (IF RESPONSE NOT = DFHRESP(NORMAL) GO TO OTHER-ERRORS) *after* any explicit tests for specific conditions. So any exceptions, other than those you might particularly “expect”, take control to the paragraph at OTHER-ERRORS in each program. Those relatively few commands that do not have RESP on them take control to exactly the same place if they result in any condition other than NORMAL because of this EXEC CICS HANDLE CONDITION ERROR command.

Relying on the system default action

You have three ways of relying on the system default action:

- Use an EXEC CICS PUSH HANDLE command.
- Use an EXEC CICS HANDLE CONDITION command without a label.
- Do nothing about the condition.

You may choose quite deliberately to ignore the condition because you want the default action to happen. However, this can be a potential source of program maintenance problems, especially if the CICS system defaults ever change over a period of time.

If you omit to cater for an obscure condition on an unfamiliar command and you do not have an EXEC CICS HANDLE CONDITION command to cope with that condition, you get the standard system (default) action. With the exception of “wait” conditions, or the EOC condition on the EXEC CICS RECEIVE or CONVERSE commands, the system abends the task.

How to use EXEC CICS PUSH HANDLE and POP HANDLE commands

The EXEC CICS PUSH HANDLE command allows you to “nest” your condition-handling code. For example, when calling a subroutine you may want a completely different set of EXEC CICS HANDLE CONDITION commands while in the subroutine. (EXEC CICS PUSH HANDLE and EXEC CICS POP HANDLE commands enable you to suspend all current EXEC CICS HANDLE CONDITION, EXEC CICS IGNORE CONDITION, EXEC CICS HANDLE AID, and EXEC CICS HANDLE ABEND commands. This can be useful, for example, during a branch to a subroutine embedded in a main program.) For information about these commands, see Chapter 32, “Application programming commands - reference,” on page 323.

The EXEC CICS PUSH HANDLE and EXEC CICS POP HANDLE commands are not supported for applications written in ILE C.

Normally, when a CICS program passes control other than by an EXEC CICS LINK or EXEC CICS XCTL command to a subroutine, the program or routine that receives control inherits the current EXEC CICS HANDLE commands. These commands may not be appropriate within the called program. The called program can use EXEC CICS PUSH HANDLE to suspend existing EXEC CICS HANDLE commands.

Use EXEC CICS PUSH HANDLE, therefore, to save your present set of EXEC CICS HANDLE CONDITION commands unaltered while you use a new set in the routine. On exit, you can reinstate the original set of EXEC CICS HANDLE CONDITION commands by using a corresponding EXEC CICS POP HANDLE command.

You can nest EXEC CICS PUSH HANDLE...EXEC CICS POP HANDLE command sequences within a task. Each EXEC CICS PUSH HANDLE command stacks a set of specifications; the EXEC CICS POP HANDLE that follows it restores them. To give you some idea of what is involved, look at Figure 26.

```

*
* PROCESSING FOR UNEXPECTED ERRORS.
  OTHER-ERRORS.
* FIRST, STACK THE CURRENT CONDITION HANDLING
  EXEC CICS PUSH HANDLE
  END-EXEC.
* NOW, SET THE SYSTEM DEFAULT ACTION
* FOR A "NOSPACE" ERROR
  EXEC CICS HANDLE CONDITION NOSPACE
  END-EXEC.
* PROCEED WITH PROCESSING
  MOVE EIBFN TO ERR-FN.
  MOVE EIBRCODE TO ERR-RCODE.
  MOVE EIBFN TO ERR-COMMAND.
  MOVE EIBRESP TO ERR-RESP.
  MOVE LOW-VALUES TO ACCTERRO.
  MOVE EIBTRNID TO TRANEO.
  MOVE ERR-PGRMID TO PGME0.
  PERFORM REASON-LOOKUP
    THROUGH REASON-END
    VARYING I FROM 1 BY 1 UNTIL I NOT < IXR.
  MOVE ERR-MSG (IXR) TO RSNEO.
  IF IXR < 12 MOVE EIBDS TO DSN,
    MOVE DSN-MSG TO FILEEO.
  PERFORM COMMAND-LOOKUP
    THROUGH COMMAND-END
    VARYING I FROM 1 BY 1 UNTIL I NOT < IXC.
  MOVE COMMAND-NAME (IXC) TO CMDEO.
  IF ERR-RESP < 94
    MOVE RESPVAL (ERR-RESP) TO RESPEO
    ELSE MOVE RESPVAL (94) TO RESPEO.
  EXEC CICS SEND MAP('ACCTERR')
    MAPSET('ACCTSET') ERASE FREEKB
  END-EXEC.
  EXEC CICS WRITEQ TS QUEUE('ACERLOG')
    FROM(ACCTERRO)
    LENGTH(ERR-LNG)
  END-EXEC.
* IF CONDITION NOSPACE OCCURS,
* WAIT FOR TS TO BECOME AVAILABLE
* NOW RESET THE PREVIOUS CONDITION HANDLING
  EXEC CICS POP HANDLE
  END-EXEC.

```

Figure 26. Using EXEC CICS PUSH HANDLE and EXEC CICS POP HANDLE commands

Suppose that, in the code in Figure 24 on page 89, instead of linking to the error-handling program ACCT04, you choose to include the error lookup and error message display in one of the other programs in your application.

You might do this if you do not want to link to a separate program, and you arrive at this piece of code from many different points in the application. There are several points to note:

- EXEC CICS PUSH HANDLE commands can be nested. The EXEC CICS PUSH HANDLE command suspends the current set of EXEC CICS HANDLE commands (saving them for later use), and the EXEC CICS POP HANDLE command restores the most recent set suspended.
- When you link to another program, an EXEC CICS PUSH HANDLE command is implied.

That is, an EXEC CICS PUSH HANDLE command occurs between the EXEC CICS LINK command and the first instruction of the linked-to program, which begins with the system defaults. It may or may not do some of its own EXEC CICS HANDLE commands and EXEC CICS PUSH HANDLE or EXEC CICS POP HANDLE commands, but afterwards, the stack is popped back to the point where the EXEC CICS LINK command occurred, to restore the HANDLE status of the linking program when control is returned there. That is, CICS pushes at each EXEC CICS LINK command and pops at each EXEC CICS RETURN command.

- When you transfer control to another program (using the EXEC CICS XCTL command), the current table of conditions that instructs CICS what to do is cleared, except for EXEC CICS HANDLE ABEND commands, which remain active. (See “How CICS keeps track of what to do” on page 99.)

One flaw in this example, of course, is that you do not know where to go at the end of the code and you do not need such sophistication if you are not going back.

(You could have all your EXEC CICS HANDLE commands sent to different labels, each of which would consist of PERFORM OTHER-ERRORS, GO TO “back”, which would be wherever this particular error occurred. This shows some of the limitations of EXEC CICS HANDLE commands, even with EXEC CICS PUSH HANDLE and EXEC CICS POP HANDLE commands.)

How to use an EXEC CICS HANDLE CONDITION condition command

The easiest way to restore the system default action for a given condition is to code an EXEC CICS HANDLE CONDITION command without a label on a named condition.

You might do this when, having “exhausted” the more likely conditions, you decide that anything else is either so unlikely, or so disastrous, that the only feasible option is the abend that the system default action generally gives you.

How CICS selects whether to take the system default action

CICS selects whether to take the system default action for a given condition according to the sequence of tests shown in the flowchart shown in Figure 27 on page 98.

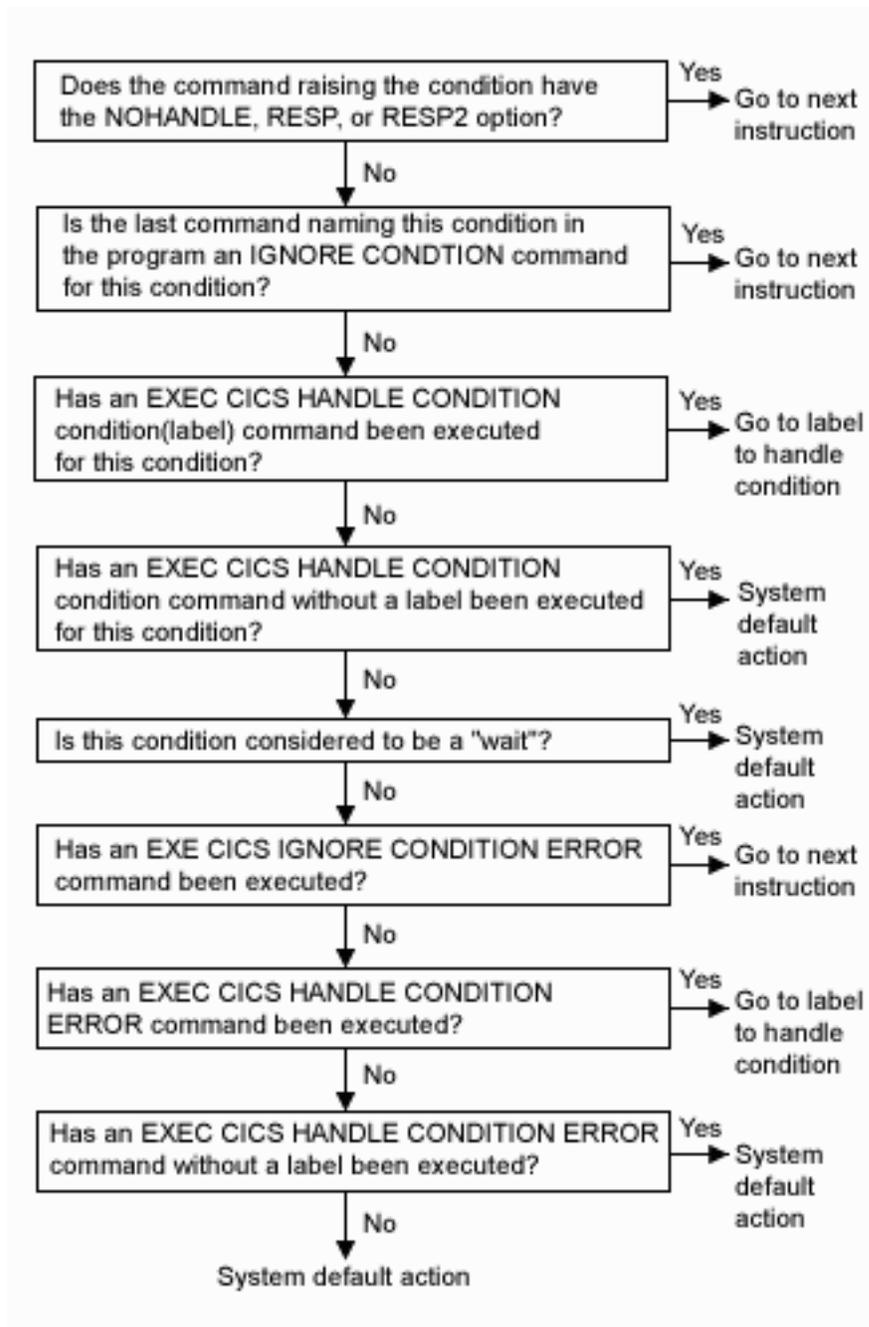


Figure 27. How CICS selects whether to take the system default action

Mixing the methods

You can temporarily deactivate the effect of any EXEC CICS HANDLE CONDITION command by using the RESP or NOHANDLE option on a command. If you do this, you lose the ability to use any system default action for that command. In other words, you have to do your own “catch-all” error processing.

You can also switch from ignoring a condition to handling it, or to using the system default action. For example, you could code:

```

*   MIXED ERROR PROCESSING
    EXEC CICS IGNORE CONDITION LENGERR
    END-EXEC.
:
    EXEC CICS HANDLE CONDITION DUPREC(DUPRTN)
    LENGERR
    ERROR(ERRHANDL)
    END-EXEC.

```

This code initially ignores condition LENGERR, so if the program raises a LENGERR condition, nothing happens; the application simply continues its processing. However, your program might not be able to continue if this condition has arisen.

Later in the code, you can explicitly set condition LENGERR to the system default action by naming it in an EXEC CICS HANDLE CONDITION command without a label. When this command has been executed, the program no longer ignores condition LENGERR, and if it subsequently occurs, it now causes the system default action.

The point is, you can mix methods and each condition is treated separately.

How CICS keeps track of what to do

CICS has a table of the conditions referred to by EXEC CICS HANDLE CONDITION and EXEC CICS IGNORE CONDITION commands in your application. Each execution of one of these commands either updates an existing entry in this table, or causes CICS to make a new entry if this is the first time the condition has been quoted in such a command. Each entry tells CICS what to do by indicating one of the three exception-handling states your application can be in, namely:

1. **Taking no action**, where control returns to the next instruction following the command that has failed
2. **With an EXEC CICS HANDLE CONDITION or EXEC CICS HANDLE CONDITION ERROR command active**, where control goes to the appropriate label in your program defined earlier by the command
3. **Taking the system default action**, where for most conditions, this is to terminate the task abnormally

CICS keeps a table of these conditions for each link level and each unpopped EXEC CICS PUSH HANDLE command. Essentially, therefore, each program level has its own HANDLE state table governing its own condition handling.

When each condition occurs, CICS performs the sequence of tests shown in Figure 27 on page 98.

The commands EXEC CICS ALLOCATE, EXEC CICS ENQ, EXEC CICS GETMAIN, EXEC CICS READQ TD, and EXEC CICS WRITEQ TS can all raise conditions for which the default action is to suspend your application program until the specified resource becomes available. So, on these commands, you have the NOSUSPEND option to inhibit this waiting and return immediately to the next instruction in your application program.

Some conditions can occur during the execution of a number of unrelated commands. If you want the same action for all occurrences, code a single EXEC CICS HANDLE CONDITION command at the start of your program.

Chapter 7. Testing your application

This chapter describes ways of making your applications more error-free.

Often two systems that run perfectly by themselves, when run together cause performance degradation and you begin experiencing “lockouts” or waits. In this case, the scope of each system has not been defined adequately.

The key points in a well-designed application system are:

- At all levels, each function is defined clearly with inputs and outputs well stated.
- Resources that the system uses are well defined.
- Interactions with other systems are known.

Testing applications

The following general rules apply to testing applications:

- Don't test on a production CICS system—use a test system, where you can isolate errors without affecting “live” databases.
- Have the testing done by someone other than the application developer, if possible.
- Document the data you use for testing.
- Test your applications several times, exercising as much program logic as is practical.
- Use the OS/400 and CICS/400 facilities to trace and debug your application during initial testing. See “Trace” on page 111.
- Starting the CICS/400 user shell before using OS/400 debug facilities can reduce the size and complexity of the output.
- Use stress or volume testing to catch problems that may not arise in a single-user environment.
- Test whether the application can handle *correct* data and *incorrect* data.
 - When correct data is entered, does your program acknowledge its receipt?
 - When incorrect data is entered, do you get an appropriate error message?
- Check whether files and databases are updated correctly.
- Test against complete copies of the related databases.
- Before you move an application to the production system, it is also good practice to run a *final* set of tests against a copy of the production database to catch any errors.

Screen usage, checks and considerations

The following lists some screen considerations for CICS programmers:

- Check whether screens are displayed in the expected sequence.
- All headings and captions should be placed correctly.
- Check for any misspellings.
- Ensure that your screen is easy to read and to use.

- Check whether the cursor is positioned correctly initially, and that the cursor moves correctly from field to field.
- Check whether each field has the desired attributes set.
- Check whether PF keys function as required; and that their functions are indicated to the user.
- Check whether all work fields are cleared properly after each valid transaction.
- If incorrect data is entered, you should ensure that an appropriate error message is returned to the screen; and that the cursor is positioned at the first field in error and the field is highlighted.
- You need to ensure that the error message is no longer issued when the user corrects the error.

Types of problems

The problems you meet when testing your code can be grouped by symptom into four general types. This classification is useful, because you need to take a slightly different approach to solve each type. Also, it is the same problem-classification scheme used by IBM programming support representatives (PSRs), so if you require assistance, it will help you in identifying your problem to IBM. The four types are:

- Abends
- Loops
- Waits
- Incorrect output

Abends: When a transaction terminates abnormally, CICS sends this message both to the terminal associated with the transaction and to the transient data message destination CSMT. (At most CICS installations, this message destination is directed to a printer used by the system administrator, to provide a second immediate notification of the event.)

Loops: A loop can contain COBOL/400, ILE C, or CICS code, but CICS/400 may not detect it. The problem symptom is that the transaction never ends. It usually produces less than all of the expected output and may leave the keyboard locked as well.

Waits: The symptoms of a transaction in the “wait” state are the same as those described for a loop containing a CICS command: the transaction never ends and may not produce all of its outputs. If your transaction behaves like this, you can tell whether you have a loop or a wait by using the CEMT transaction. Display the task:

```
CEMT INQUIRE TASK FACILITY(tttt)
```

(“tttt” is the name of the terminal from which the transaction was entered.) If the task still exists and is active, wait a minute and repeat the inquiry. If the same task is still there, the program is probably in a loop that contains a CICS command. Using EDF from a second terminal will help you to discover the CICS command in the loop. For more information about EDF, see Chapter 27, “Execution diagnostic facility (EDF),” on page 229.

If the task is not active but suspended, repeat the display once or twice. If the task remains suspended, it’s probably waiting for some event that’s never going to happen.

Incorrect output: The last category of problem covers those situations in which the transaction appears to run successfully but produces the wrong results. It includes wrong answers, missing or extra records in files, screens filled with what appear to be random characters, or no output at all.

Levels of testing

A **single-thread** or **unit** test takes one application transaction at a time, in an otherwise “empty” CICS system, and checks how it behaves. This enables you to test the program logic, and also shows you whether or not the basic CICS information (such as resource definitions) is correct. It’s quite feasible to test this single application in a test CICS control region while your normal, online production CICS system is active in another control region.

A **multithread** or **string** test involves several, concurrently active transactions. Naturally, all the transactions are running under the same CICS control region, so you can readily test the ability of a new transaction to coexist with other transactions.

You may find that a transaction that works perfectly in its single-thread testing fails in the multithread test. Also, it may cause other transactions to fail, or even terminate a task.

A **regression** test is used to make sure that all the transactions in a system continue to do their processing in the same way both before and after changes are applied to the system. This is to ensure that fixes that have been applied to solve one problem don’t go on to cause further problems. It’s often a good idea to build a set of miniature files to perform your tests, because it’s much easier to examine a small data file for changes.

A good regression test exercises all the code in every program—that is, it explores all tests and possible conditions. As your system develops to include more transactions, and more possible conditions, add these to your test system. The results of each test should match those from the previous round of testing. Any discrepancies are grounds for suspicion. You should compare terminal output, file changes, and log entries for validity.

Finding a problem in application code on a production system

If an offending piece of application code slips through the net of testing and into the production system, try the following:

- Monitor the JOBLOG and CSMT log (if defined in your system), check for transaction errors, and use the trace facilities to find failing programs.
- Check for incorrectly addressed data areas or data areas that are too small.

Chapter 8. Recovery considerations

There are two techniques available to help recover or reconstruct events or data changes during CICS execution:

- Standard recovery (syncpoints)
- CICS user journals

But you may not need to use CICS user journals for data recovery purposes. By properly defining recoverable files to CICS, by accurately maintaining this information, and by using OS/400 commitment control, you may find that the OS/400 facilities are adequate for recovery.

Before discussing recovery techniques, some information is needed about how CICS uses the **commitment control** facilities of OS/400.

CICS and OS/400 commitment control recovery

CICS uses the OS/400 commitment control facilities to control the backout and recovery of CICS resources defined as recoverable. CICS/400 is no different from any native OS/400 application running under commitment control.

To facilitate recovery in the event of abnormal termination of a CICS task or of a failure of a CICS/400 control region, the CICS system administrator can, during CICS table generation, define specific CICS resources (mainly file resources) as recoverable.

Each recoverable CICS resource is opened within CICS under the commitment control facilities of OS/400. Each recoverable CICS file must be registered by the system administrator to an OS/400 journal and journal receiver. These OS/400 objects are used by OS/400 to retain the information necessary to perform commitment and rollback on files opened under commitment control.

Note: The OS/400 journal and journal receiver objects are not to be confused with a CICS user journal. While the use of OS/400 journals as a recovery mechanism may be similar to the use of a CICS user journal, they are not the same. Only CICS user journals are accessed by CICS journal control commands. See “User journaling” on page 106.

For a more complete description of commitment control, refer to the *Backup and Recovery*. For further information about defining CICS recoverable resources, refer to the *CICS for iSeries Administration and Operations Guide*.

Defining recoverable files to CICS (an overview)

The system administrator creates an OS/400 journal receiver (or a number of them) using the CRTJRNRCV CL command. Once a journal receiver is created, it is attached to a specific OS/400 journal by using the CRTJRN CL command. Each file to be opened under commitment control is registered to an OS/400 journal using the Start Journal Physical File (STRJRNPF) CL command. Using the STRJRNPF command, the system administrator indicates which file images are to be captured (before update, after update, or both) and whether open and close operations are to be journaled. When the STRJRNPF command has been run successfully, any

changes to the file by any process is logged in the journal receiver. Commitment control of the transactions for a particular process using that file does not start until the process starts commitment control. More information about recoverability can be found in the *CICS for iSeries Administration and Operations Guide*.

Syncpointing

To facilitate recovery in the event of abnormal termination of a CICS task or of failure of the CICS system, the system programmer can, during CICS table generation, define specific resources (for example, files) as recoverable. If a task is terminated abnormally, these resources are restored to the condition they were in at the start of the task, which can then be rerun. The process of restoring the resources associated with a task is termed **backout**.

If an individual task fails, backout is performed by the commitment control facilities of OS/400 and by CICS component modules. If the CICS system fails, backout is performed as part of the emergency restart process.

However, for long-running programs, it may be undesirable to have a large number of changes, accumulated over a period of time, exposed to the possibility of backout in the event of task or system failure. This possibility can be avoided by using the EXEC CICS SYNCPOINT command to split the program into logically separate sections known as logical units of work (LUWs); the end of an LUW is referred to as a synchronization point (**syncpoint**).

If failure occurs after a syncpoint but before the task has been completed, only changes made after the syncpoint are backed out.

LUWs must be entirely logically independent, not merely with regard to protected resources, but also with regard to execution flow. Typically, an LUW comprises a complete conversational operation bounded by EXEC CICS SEND and EXEC CICS RECEIVE commands. A browse is another example of an LUW; an EXEC CICS ENDBR command must therefore precede the syncpoint.

User journaling

CICS provides facilities for defining and managing CICS **user journals** during CICS processing. CICS user journals are special-purpose nonrecoverable sequential files.

CICS user journals may be used as an audit trail, or as a change file of file updates and additions, or for any other purpose needed by a CICS application program.

Journal control commands are provided to allow the application programmer to create a journal record (EXEC CICS WRITE JOURNALNUM).

Exception conditions that occur during execution of a journal control command are handled as described in Chapter 6, "Dealing with exception conditions," on page 87..

Journal records

CICS user journals are defined to CICS in the journal control table (JCT). Data may be written to any journal specified in the JCT. The JCT may define one or more journals on direct access storage. Each journal is identified by a number known as the journal identifier. This number is in the range from 1 through 99. Generally the

system administrator is responsible for defining the JCT entries. For more information about defining journals, see the *CICS for iSeries Administration and Operations Guide*.

Each journal record begins with a standard length field (LL), a user-specified identifier, and a system-supplied prefix. This data is followed in the journal record by any user-supplied prefix data (optional), and finally by the user-specified data. Journal control is designed so that the application programmer requesting output services need not be concerned further with the detailed layout and precise contents of journal records. The programmer needs to know which journal to use, what user data to specify, and what user-identifier to supply.

The layout of the journal records is defined in the *CICS for iSeries Administration and Operations Guide*. CICS user journals are defined as variable-length record physical files. To use these files in a batch application, you must know the physical file information for the journal you want, and the format of the CICS user journal records.

The use of the information recorded on a CICS user journal is left up to the needs of the application. This information can be used by reading the physical file records on the journal. For example:

- An application could use any before and after record images recorded on a CICS user journal to recreate updates to a file if desired.
- The application may create an audit trail report indicating the updates that took place and the areas of the records that had changed.
- The application could also report the date and time of the update, the id of the user performing the update, and the CICS terminal ID where the changes were made.

Journal output synchronization

References to journals in this section mean CICS user journals.

Journal output takes place at the time of the EXEC CICS WRITE JOURNALNUM command. The WAIT and STARTIO options are allowed within CICS/400, but are treated as null operations (no-ops).

If an out-of-space condition occurs while trying to execute an EXEC CICS WRITE JOURNALNUM command, the action taken by CICS varies based on how the journal file is defined in the JCT. See the *CICS for iSeries Administration and Operations Guide* for details.

If the requested journal file is defined as switchable, CICS switches to a new journal file and writes the journal record to the newly created journal file automatically when the first journal file is full. The task requesting the write waits until the journal record is written to the new journal file. If the second file becomes full, another switch to a new journal file occurs. Journal switching occurs when necessary, until the maximum number of switched journal files has been reached. If this occurs, the JCT entry for that journal file is marked CLOSED and DISABLED.

If the journal file is defined as nonswitchable, CICS closes the journal file, marks the JCT entry as CLOSED and returns an IOERR condition to the application program. If the program has an active EXEC CICS HANDLE CONDITION command set up to deal with an IOERR condition, control is returned to the program (to the location requested in the HANDLE CONDITION request). If there

is no `HANDLE CONDITION` command set for an IOERR condition, the default action for an IOERR condition is taken; that is, to abend the transaction.

Chapter 9. Abnormal termination recovery

A program-level abend exit facility is provided in CICS so that you can write exits of your own that can be given control during abnormal termination of a task. An example of a function performed by such an abend exit is the “cleanup” of a program that has started but not completed normally.

Abnormal termination can occur because of:

- A user request by, for example:
`EXEC CICS ABEND ABCODE(...)`
- A CICS request as a result of an invalid user request.

The `EXEC CICS HANDLE ABEND` command activates or reactivates a program-level abend exit within your application program; you can also use this command to cancel a previously activated exit.

When activating an exit, you must use the `PROGRAM` option to specify the name of a program to receive control, or (in COBOL) the `LABEL` option to specify a routine label to which control will branch when an abnormal termination condition occurs.

An `EXEC CICS HANDLE ABEND` command overrides any preceding such command in any application program at the same logical level. Each application program of a transaction can have its own abend exit, but only one abend exit at each logical level can be active. (Logical levels are explained in Chapter 20, “Program control,” on page 199.)

When a task is abnormally terminated, CICS searches for an active abend exit, starting at the logical level of the application program in which the abend occurred, and proceeding, if necessary, to successively higher levels. The first active abend exit found, if any, is given control. This procedure is shown in Figure 28 on page 111, which also shows how subsequent abend processing is determined by the user-written abend exit.

If no abend exit is found, CICS terminates the task abnormally.

To prevent recursive abends in an abend exit, CICS deactivates the exit upon entry to the exit routine or program. If a retry of the operation is desired, the application programmer can branch to a point in the program that was in control at the time of the abend and issue an `EXEC CICS HANDLE ABEND RESET` command to reactivate the abend exit. This command can also be used to reactivate an abend exit (at the logical level of the issuing program) that was canceled previously by an `EXEC CICS HANDLE ABEND CANCEL` command. You can suspend the `EXEC CICS HANDLE ABEND` command by means of the `EXEC CICS PUSH HANDLE` and `EXEC CICS POP HANDLE` commands as described in “How to use `EXEC CICS PUSH HANDLE` and `POP HANDLE` commands” on page 95.

The `EXEC CICS HANDLE ABEND` command cannot intercept abends resulting from OS/400 machine event errors.

Creating a program-level abend exit

Abend exit programs can be coded in any supported language, but abend exit routines must be coded in the same language as their program.

Upon entry to an abend exit program, no addressability can be assumed other than that normally assumed for any application program coded in that language.

There are three means of terminating processing in an abend exit routine or program, as listed below. Note, however, that all abend routines should terminate with an abend, except for those handling abends generated as a result of application program logic.

- Using an EXEC CICS RETURN command to indicate that the task is to continue running with control passed to the program on the next higher logical level. If no such program exists, the task is terminated normally.
- Using an EXEC CICS ABEND command to indicate that the task is to be abnormally terminated with control passed either to an abend exit specified for a program on a higher logical level or, if there is not one, CICS for abnormal termination processing.
- Branching to retry an operation. When you are using this method of retrying an operation, and you want to reenter the original abend exit routine or program if a second failure occurs, the abend exit routine or program should issue the EXEC CICS HANDLE ABEND RESET command before branching. This is because CICS will have disabled the exit routine or program to prevent it reentering the abend exit.

Restrictions on retrying operations

If an abend occurs during the invocation of a CICS service, you should be aware that issuing a further request for **the same service** may cause unpredictable results, because the reinitialization of pointers and work areas, and the freeing of storage areas in the exit routine, may not have been completed.

You should not try to recover from ATNI or AEXI abends by attempting further I/O operations. Either of these abends results in a TERMERR condition, requiring the session to be terminated in all cases.

If intersystem communication (ISC) is being used, an abend in the remote system may cause a branch to the specified program or label, but subsequent requests to use resources in the remote system will fail.

If an abend occurs as a result of a BMS command, control blocks are not tidied up before control is returned to the BMS program, and results are unpredictable if the command is retried.

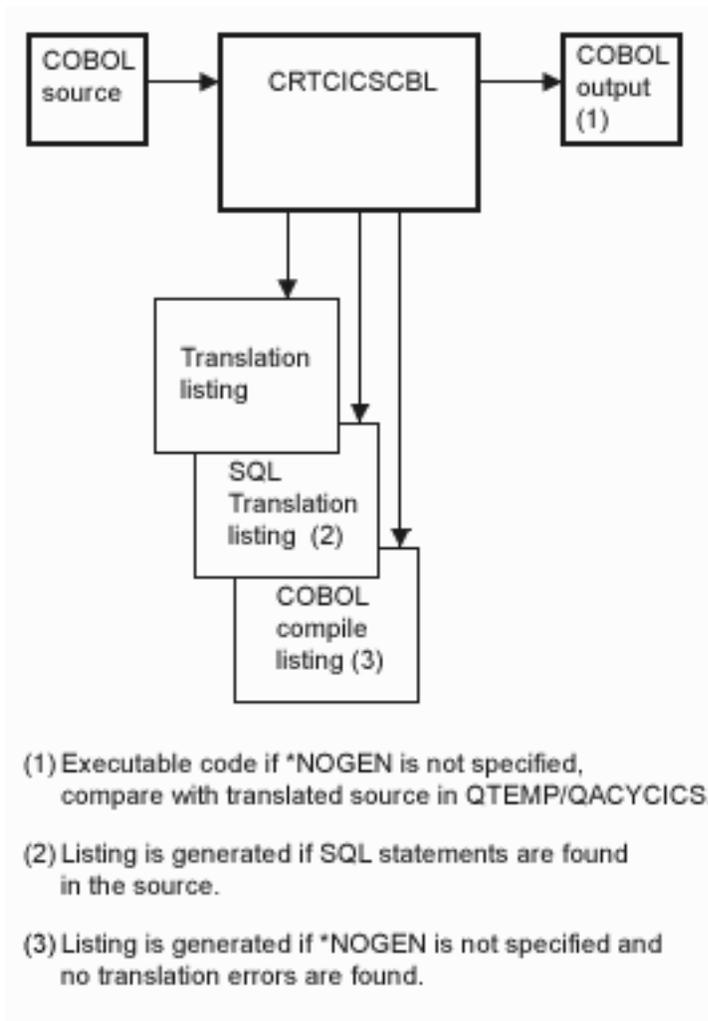


Figure 28. ABEND exit processing

Trace

CICS trace is a debugging aid for application programmers, system programmers, and IBM field engineers. It produces trace entries in response to trace commands. The trace entries can be sent to any trace destination that is currently active. The destinations are:

- Internal trace table
- Auxiliary trace user spaces

For information about trace destinations, see the *CICS for iSeries Problem Determination*.

You can:

- Specify user trace entry points (EXEC CICS ENTER TRACENUM).
- Switch CICS internal trace on or off using the EXEC CICS SET TRACEDEST command. For information about this command, see “SET TRACEDEST” on page 522.

- Switch auxiliary trace user spaces using the EXEC CICS SET TRACEDEST command. The EXEC CICS SET TRACEDEST command is also used to set auxiliary trace on and off.

Trace entry points

There are two types of trace entry produced during CICS operations: system trace entry points and user trace entry points.

System trace entry points

These are points within CICS at which trace control requests are made. The most important system trace entry points for application programmers are for the EXEC interface program. These produce entries in the trace table whenever a CICS command is processed.

Two trace entries are made: the first when the command is issued, and the second when CICS has performed the required function and is about to return control to your application program. Between them, these two trace entries allow you to trace the flow of control through an application, and to check which exception conditions, if any, occurred during its execution.

User trace entry points

These are additional points within your application program that you can include in the trace table to allow complete program debugging. For example, you could specify an entry for a program loop containing a counter value showing the number of times that the loop had been entered.

A trace entry is produced wherever the EXEC CICS ENTER TRACENUM command is run. Each trace entry request, which can be given a unique identifier, causes data to be placed in the trace table.

Dump

CICS/400 provides a diagnostic dump facility that takes advantage of dump facilities provided within OS/400. Dump output is directed to a DPFHDMP spool file. CICS/400 sets the user data (USRDTA) attribute of the spool file to indicate the CICS abend code that caused the dump.

Dump support is provided for:

- EXEC CICS DUMP command
- EXEC CICS ABEND command
- CEMT PERFORM SNAP command
- Internal dump requests; for example, an unhandled transaction abend

Transaction abends result in dump information only if the **Dump on abend** field of the PCT entry for the active transaction is set to *YES. When the dump is run for a program that has a PPT **Calling convention (PGMLNG)** parameter value of *COBOL, CICS/400 provides a COBOL/400 formatted dump. In other circumstances, CICS/400 uses the DMPJOB CL command to dump the appropriate level of information.

Note: Use of the DMPJOB CL command may be restricted at your installation.

Part 3. Files and databases

Chapter 10. File control	115
Emulated VSAM files.	115
Key-sequenced file (KSDS)	116
Entry-sequenced file (ESDS)	116
Relative record file (RRDS)	116
VSAM-like logical views.	116
Reading records	117
Direct reading (using EXEC CICS READ)	118
Direct reading from a KSDS	118
Direct reading from an ESDS	118
Direct reading from an RRDS	119
Direct reading by way of a path	119
Sequential reading (browsing)	119
Browsing through a KSDS	119
Browsing through an ESDS.	120
Browsing through an RRDS	120
Browsing using a path	120
Ending the browse	121
Simultaneous browse operations	121
Skip-sequential processing	121
Updating records	122
Specifying record length.	122
Deleting records	123
Deleting groups of records (generic delete)	123
Adding records.	123
Adding to a KSDS.	123
Adding to an ESDS	124
Adding to an RRDS	124
Specifying record length.	124
Review of file control command options	124
The RIDFLD option	124
The INTO and SET options.	125
The FROM option.	125
Avoiding transaction deadlocks	126
KEYLENGTH option for remote files	127
Record identification	127
Identifying records by key	127
Relative byte address (RBA) and relative record number (RRN)	128
RBA	128
RRN	128
CICS locking of emulated VSAM records in recoverable files	128

Chapter 10. File control

This chapter discusses CICS file control under the following headings:

- “Emulated VSAM files”
- “Reading records” on page 117
- “Updating records” on page 122
- “Deleting records” on page 123
- “Adding records” on page 123
- “Review of file control command options” on page 124
- “Avoiding transaction deadlocks” on page 126
- “KEYLENGTH option for remote files” on page 127
- “Record identification” on page 127
- “CICS locking of emulated VSAM records in recoverable files” on page 128

CICS file control lets you read, update, add, browse, and delete data in emulated VSAM files.

The following API commands provide basic operations against files within CICS:

- EXEC CICS WRITE command (adds a file record)
- EXEC CICS REWRITE command (updates a file record)
- EXEC CICS READ command (retrieves a file record)
- EXEC CICS DELETE command (deletes one or more file records)
- EXEC CICS UNLOCK command (releases a record lock)
- EXEC CICS STARTBR, EXEC CICS READNEXT, EXEC CICS READPREV, EXEC CICS RESETBR, and EXEC CICS ENDBR commands (read sequentially or browse records in a file)

For detailed guidance on the use of these commands, see Chapter 32, “Application programming commands - reference,” on page 323. In general, you need not worry about the physical organization of data in the file.

A CICS application program reads and writes individual records. Each read or write request is made by a CICS command. To access a record, the application program must identify both the record and the file that contains it. Unless you specify the SET option, it must also specify the storage area into which the record is to be read, or from which the record is to be written.

CICS file control offers you access to OS/400 physical files that are managed by an emulated virtual storage access method (VSAM). These files must be defined within the file control table (FCT).

Emulated VSAM files

CICS supports access to the three types of emulated VSAM files:

- Key-sequenced file (KSDS)
- Entry-sequenced file (ESDS)
- Relative record file (RRDS)

Key-sequenced file (KSDS)

A **key-sequenced file** has each of its records identified by a key. (The **key** of each record is simply a field in a predefined position within the record.) Each key must be unique in the file.

When the file is initially loaded with data, or when new records are added, the logical order of the records depends on the collating sequence of the key field. This also fixes the order in which you retrieve records when you browse through the file. The **physical** order in which records are written to the file is determined by OS/400.

To find the physical location of a record in a KSDS, the iSeries database facility creates and maintains an **index**. This relates the key of each record to the record's relative location in the file. When you add or delete records, this index is updated accordingly.

Entry-sequenced file (ESDS)

An **entry-sequenced file** is one in which each record is identified by its relative byte address (RBA).

Records are held in an ESDS in the order in which they were first loaded into the file. New records added to an ESDS always go after the last record in the file. You may not delete records or alter their lengths. After a record has been stored in an ESDS, its RBA remains constant. When browsing, records are retrieved in the order in which they were added to the file.

Relative record file (RRDS)

A **relative record file** has fixed-length slots, predefined to emulated VSAM, in which records may be stored. An RRDS record is always fixed-length, equal to the slot size. A record in an RRDS is identified by the **relative record number** (RRN) of the slot that holds it. When a new record is added to an RRDS, emulated VSAM uses the number you supply with the file control request.

VSAM-like logical views

Sometimes you want to access the same set of records in different ways. For example, you may have records in a personnel file that have as their key an employee number. No matter how many Smiths you had, each of them would have its own, unique employee number. Think of this as the primary key.

If you were producing a telephone directory from the file, you would want to list people by name rather than by employee number. You can identify records in a file with a secondary (alternate) key instead of the primary key described above. So the primary key is the employee number, and the employee name is the **alternate key**. Alternate keys are just like the primary key in a KSDS—fields of fixed length and fixed position within the record. You can have any number of alternate keys per base file and, unlike the primary or base key, alternate keys need not be unique.

To continue the personnel example, the employee's department code might be defined as a further alternate key.

Emulated VSAM allows KSDS and ESDS (but not RRDS) files to have alternate keys. When the file is added to or amended on the base file, one secondary or **alternate index** is built for each alternate key in the record and is related to the

primary or base key. To access records using an alternate key, you must define a logical view, using the OS/400 create logical file (CRTLF) command, which provides a **path** through the file. This view behaves as if it were a KSDS where records are accessed using the alternate key. When you add or update a record that has an alternative view, whether you use the primary key or a path, the record is maintained on the base file, and the corresponding alternate index is updated to reflect the change.

A CICS application program determines whether it is accessing a file by means of its primary key (base), or by means of a path, by the file name you specify. In a running CICS system, accesses to a single base file can be made by way of the base and by any of the paths defined to it, if each such access route is defined in the file control table (FCT).

Reading records

There are several methods of reading records: direct reading, sequential reading (browsing), and skip-sequential browsing.

A file can be defined in the FCT as containing either fixed-length or variable-length records.

Fixed-length records should be defined only if:

- The definition of the emulated VSAM file (using OS/400 database creation facilities) specifies no variable-length fields within the file's associated data definition services (DDS) description, or if the file was created without a DDS description

or

- The definition of the file contains multiple variable-length fields as specified within the file's associated DDS, or contains a variable-length field that is not the last field defined within the file's record.

Variable-length records should be defined only if:

- The definition of the file contains one variable-length field as specified within the file's associated DDS, and it is the last field defined within the record
- and**
- The definition of the emulated VSAM file is not a logical view (alternate path) that either contains multiple record formats or is a join of multiple files.

Note: Unless you are concerned about portability to other CICS platforms, files with variable-length records can be defined to CICS in the FCT as having fixed-length records and each variable-length field will be preceded with a 2-byte binary length field. If defined to CICS as a variable-length record, the variable-length field would not be preceded with a 2-byte binary length field.

For direct reading and browsing, if the file contains fixed-length records, and if the application program provides an area where the record is to be read, that area must be of the defined length. If the file contains variable-length records, the command must also specify the length of the area provided to hold them (normally the maximum length of records in the file).

For fixed-length records and for records retrieved into CICS-provided SET storage, you need not specify the length argument. However, you may like to do

so, to check that the record being read is not too long for the available data area. If you provide the length argument, CICS uses the length field to return the actual length of the record retrieved.

Direct reading (using EXEC CICS READ)

You read a record in the file with the EXEC CICS READ command. This command must identify the record you want and say whether it is to be read into an area of storage provided by your application program (EXEC CICS READ INTO), or into CICS SET storage acquired by file control (EXEC CICS READ SET). If the latter, the address of the data in the CICS SET storage is returned to your program. For further information, see the SET option on “READ” on page 390.

The length of time that the CICS SET storage remains valid depends on whether the EXEC CICS READ command is for update or not. The SET storage for a nonupdate EXEC CICS READ command survives until another nonupdate or EXEC CICS READ UPDATE command (either INTO or SET) is encountered, regardless of syncpoint. The SET storage for an EXEC CICS READ UPDATE command survives until the next EXEC CICS REWRITE, EXEC CICS UNLOCK, EXEC CICS DELETE (without the RIDFLD option), or EXEC CICS SYNCPOINT command, whichever is encountered first.

For both update and nonupdate commands, you must identify the record to be retrieved by the record identification field specified in the RIDFLD option. Immediately after completion of an EXEC CICS READ UPDATE command, the RIDFLD data area is available for reuse by the application program.

You can specify only one update operation at a time for each file within a transaction. To avoid deadlock when accessing an emulated VSAM file, your next command to the file must be an EXEC CICS REWRITE, EXEC CICS DELETE without RIDFLD, or EXEC CICS UNLOCK command.

Direct reading from a KSDS

When reading from an emulated VSAM KSDS, the record you want is usually identified by specifying its full key. You can, however, also specify a partial (**generic**) key. Emulated VSAM retrieves from the file the first record whose leftmost characters match the partial key. Or you can retrieve the record in the file whose key is greater than or equal to the full key provided with the command. Finally, you can also identify the record you want by providing a generic key together with the “greater than or equal” (**GTEQ**) option.

An EXEC CICS READ command raises the NOTFND condition if no record with the key specified is found; or with the GTEQ option, if no record is found with a key greater than or equal to the specified key. EQUAL requests a record with exactly the key specified. GTEQ requests the first record with a key greater than or equal to the key specified.

Direct reading from an ESDS

When reading from an emulated VSAM ESDS, the individual record you want is identified by an RBA. Because the RBA of a record in an ESDS cannot change, unless the ESDS has another logical view that allows it, or the file is updated outside of the CICS-provided facilities, your application program can keep track of the values of the RBAs corresponding to the records it wants to access. An access to an emulated VSAM ESDS specifying an incorrect RBA, or an RBA where there is no record, returns the ILLOGIC condition.

Direct reading from an RRDS

When reading from an emulated VSAM RRDS, the record to be retrieved is identified by its relative record number. The application program must know the RRN values of the records it wants. For records not present in the file, the NOTFND condition is returned.

Direct reading by way of a path

If a KSDS or an ESDS has an alternate index and an alternate index path (and an appropriate entry in the FCT), you can retrieve a record in the file by using the alternate key that you set up in the alternate index. The GENERIC option and the GTEQ (greater than or equal to) option still work in the same way as for a read from a KSDS using the primary key.

If the alternate key in an EXEC CICS READ command isn't unique, the first record in the file with that key is read and you get the DUPKEY condition. To retrieve other records with the same alternate key, you have to start a browse operation at this point.

Sequential reading (browsing)

You start a browse with the EXEC CICS STARTBR command, identifying a particular record in the same way as for a direct read. However, the EXEC CICS STARTBR command only identifies the starting position for the browse; it doesn't retrieve a record.

You can reset a browse at any time using the EXEC CICS RESETBR command. Use an EXEC CICS RESETBR command to define a new starting position for the browse, or to change the type of search argument used.

The EXEC CICS READNEXT command reads records sequentially from the file, from the starting point set by the EXEC CICS STARTBR command. CICS updates the field specified in the RIDFLD option on the EXEC CICS READNEXT command with the complete key, relative byte address, or relative record number of the record retrieved each time an EXEC CICS READNEXT command is processed.

The record (as in the case of a direct read) may be read into an area of storage supplied by the application program (the EXEC CICS READNEXT INTO command), or into CICS-provided SET storage (the EXEC CICS READNEXT SET command). In the latter case, the CICS SET storage remains valid until the next EXEC CICS ENDBR command for this REQID, or an EXEC CICS SYNCPOINT command is issued.

The EXEC CICS READPREV command is like the EXEC CICS READNEXT command, except that records are read sequentially *backward* from the starting point set by the EXEC CICS STARTBR command.

Note that if you change from an EXEC CICS READNEXT command to an EXEC CICS READPREV command, or the converse, the same record is retrieved twice, because these commands also serve to change the current browse position in the file.

Browsing through a KSDS

You can use a generic key on the EXEC CICS STARTBR command when browsing through an emulated VSAM KSDS. However, the browse can only continue forward through the file. If you process an EXEC CICS READPREV command during such a browse, you get the INVREQ condition.

You can use the options “key equal to” (EQUAL) and “key greater than or equal to” (GTEQ) on the EXEC CICS STARTBR command. The default is the “key greater than or equal to” option. If you specify the “key equal to” option and no record matches the key specified, you get the NOTFND condition.

An EXEC CICS READNEXT or EXEC CICS READPREV command works only after the successful execution of an EXEC CICS STARTBR command.

You can start a forward browse through an emulated VSAM KSDS at the start of the file by specifying a key of hexadecimal zeros and the GTEQ option, or by specifying options GENERIC, GTEQ, and KEYLENGTH(0) on the EXEC CICS STARTBR or EXEC CICS RESETBR commands. (In the latter case, you need the RIDFLD keyword although its value isn’t used and, after the command completes, CICS will be using a generic key length of one.)

You can start from the end of the file by specifying a complete key of X'FF' characters on the EXEC CICS STARTBR or EXEC CICS RESETBR command. This points to the last record in the file ready for a backward browse.

An EXEC CICS STARTBR, EXEC CICS RESETBR, or EXEC CICS READNEXT command having the option KEYLENGTH(0) is always treated as if KEYLENGTH(1) and a partial key of one byte of binary zeros have been specified.

Browsing through an ESDS

You cannot use the GTEQ option on the EXEC CICS STARTBR command when browsing through an emulated VSAM ESDS. If no record matches the RBA specified in the EXEC CICS STARTBR command, you get the ILLOGIC condition. Like emulated VSAM KSDS, keys of X'00' characters and X'FF' characters on the EXEC CICS STARTBR command enable browses to start at the first or last record respectively.

Browsing through an RRDS

You can use the GTEQ option on an EXEC CICS STARTBR command when browsing through an RRDS. It is the default, even though on a direct EXEC CICS READ this option has no effect. A direct read command with the GTEQ option that specifies an RRN that doesn’t exist returns the NOTFND condition, because only the EQUAL option is taken. However, an EXEC CICS STARTBR GTEQ command using the same RRN completes successfully, if there is a record with an RRN higher than that specified. It also sets a pointer to the relevant position in the file for the start of the browse. The first record in the file is identified using an RRN of 1, and the last record by X'FF' characters.

Browsing using a path

Browsing can also use an alternate index path to an emulated VSAM KSDS or an ESDS. The browse is just like that for an emulated VSAM KSDS, but using the alternate key. The records are retrieved in alternate key order.

When nonunique alternate keys are involved, a browse operation retrieves all records with the same alternate key. The EXEC CICS READNEXT command retrieves those records in the order that they were added to the file. (You could use the EXEC CICS READPREV command, but these records are returned in the same order as for the EXEC CICS READNEXT command.)

If you have nonunique keys and switch from an EXEC CICS READNEXT command to an EXEC CICS READPREV command, or from an EXEC CICS READPREV command to an EXEC CICS READNEXT command, the next record obtained is the first occurrence of this key. The DUPKEY condition is returned for

each retrieval operation except the last. For example, if there are three records with the same alternate key, the DUPKEY condition is raised for retrieval of the first two, but not the third. You can design the application program to revert from browsing to direct reading, when the DUPKEY condition no longer occurs.

Ending the browse

Trying to browse past the last record in a file raises the ENDFILE condition. Stop a browse with the EXEC CICS ENDBR command. You must issue the EXEC CICS ENDBR command before performing an update operation on the same file (an EXEC CICS READ UPDATE, EXEC CICS DELETE with RIDFLD, or EXEC CICS WRITE command), before a syncpoint, or before task termination. If you don't, you get unpredictable results, possibly including deadlock within your own transaction.

Simultaneous browse operations

CICS allows a transaction to perform more than one browse on the same file at the same time. You distinguish between browse operations by including the REQID option on each browse command.

Skip-sequential processing

For quick direct access to records, you can browse using skip-sequential processing. This reduces index search time. It is useful if the records are retrieved in ascending or descending order, relatively close to each other, but not necessarily adjacent.

CICS automatically reinitiates skip-sequential processing when the sequence is broken.

You can use skip-sequential processing if you change the key, RBA, or RRN in the RIDFLD option of the EXEC CICS READNEXT or EXEC CICS READPREV command to point to the next record you want. You can even do this on the first EXEC CICS READNEXT or EXEC CICS READPREV command after an EXEC CICS STARTBR or EXEC CICS RESETBR command.

Note: The RIDFLD option on the EXEC CICS READNEXT or EXEC CICS READPREV command must be in the same form (key, RBA or RRN) as that used in the EXEC CICS STARTBR command or last EXEC CICS RESETBR command. If you use generic keys on a forward browse, the new RIDFLD must also be a generic key, although it need not be of the same length.

Including the KEYLENGTH option on the EXEC CICS READNEXT command has the same effect as an EXEC CICS RESETBR command, because the key length has been changed. To continue browsing from this new point, remove the KEYLENGTH option from subsequent EXEC CICS READNEXT commands.

Note also that if a "key equal to" search is specified on an EXEC CICS STARTBR command or on an EXEC CICS RESETBR command, an EXEC CICS READNEXT command using skip-sequential processing may give a NOTFND condition. It is not possible to obtain the last record on the file by specifying a complete key of X'FF's during skip sequential processing. You must use an EXEC CICS RESETBR command to specify the key in this form.

Updating records

To update a record, you must first retrieve it using an EXEC CICS READ command with the UPDATE option. The record is identified in exactly the same way as for a direct read. In an emulated VSAM KSDS or ESDS, the record may (as with a direct read) be accessed by way of an FCT entry that refers either to the base, or to a path defined to it.

After modification by the application program, the record is written back to the file using the EXEC CICS REWRITE command. The EXEC CICS REWRITE command does not identify the record being rewritten, because in any one transaction CICS allows only a single update to a given file to be in progress at any time.

A record retrieved as part of a browse operation cannot be updated during the browse. The application program must end the browse, read the desired record with an EXEC CICS READ UPDATE command, and perform the update. Failure to end the browse before issuing the EXEC CICS READ UPDATE command may cause a deadlock.

The record to be updated may (as in the case of a direct read) be read into an area of storage supplied by the application program or into storage set by CICS. If the record is read into CICS SET storage, it should normally be copied into application storage and rewritten from that storage. For an EXEC CICS READ UPDATE command, CICS SET storage remains valid until the next EXEC CICS REWRITE, EXEC CICS UNLOCK, EXEC CICS DELETE without RIDFLD, or EXEC CICS SYNCPOINT command, whichever is encountered first.

If you want to release the string held by an EXEC CICS READ UPDATE command without rewriting or deleting the record, use the EXEC CICS UNLOCK command. This releases any CICS storage acquired for the EXEC CICS READ command and releases emulated VSAM resources held by the EXEC CICS READ command.

For an emulated VSAM KSDS, the primary key embedded in the record should not be altered when the record is modified. Similarly, if the update is being made by way of a path, the embedded alternate key used to identify the record should not be altered either, although other alternate keys may be altered. This is because if the embedded key in the record were altered, the record's location in the file would be changed and therefore the locking afforded by the UPDATE request would no longer apply. When rewriting a record, the *original* record read for update would be discarded and a new record (with a new key) would be written. No exception conditions would be raised.

If the FCT entry allows variable-length records, the length of the record may be changed. The length of records in an ESDS, an RRDS, and a fixed-length KSDS must not be changed on update.

Specifying record length

For a file defined to CICS as containing fixed-length records, the length of record being rewritten *must equal the length defined to emulated VSAM*, which may be obtained by issuing the iSeries display file definition (DSPFD) command or by looking at the related FCT entry. For variable-length records, you must specify the length with both the EXEC CICS READ and the EXEC CICS REWRITE commands. The length must not be greater than the maximum defined for the file at file definition.

Deleting records

You can not delete records directly from an ESDS. However, if you access the file through an alternate path, CICS/400 sees the file as a KSDS and allows records to be deleted from it. This is different from mainframe CICS, where VSAM prevents the deletion of records from an underlying ESDS. To prevent the deletion of records from an arrival-sequenced physical file object in CICS/400, you should put object authority on the file so as not to allow deletes. Then, if deletion is attempted against an alternate path, CICS/400 will return a NOTAUTH condition.

You can delete a record in an emulated VSAM KSDS or RRDS by first retrieving it for update and then issuing an EXEC CICS DELETE command. As in the case of the EXEC CICS REWRITE command, the record to be deleted must not be identified within the EXEC CICS DELETE command; it is, by default, the record most recently read for update. When an EXEC CICS DELETE command follows an EXEC CICS READ UPDATE command, it must not include the RIDFLD option. If the RIDFLD option is included, an INVREQ condition is returned to the application program.

You can also delete a record in a single operation, again using the EXEC CICS DELETE command. In this case, you identify the record to be deleted as part of the command. You do this by specifying the RIDFLD option.

If a full key is used either on an EXEC CICS READ UPDATE command before an EXEC CICS DELETE command, or with the EXEC CICS DELETE command, a single record with that key is deleted. So, if the file is being accessed by way of a logical view that allows nonunique alternate keys, only the first record with that key is deleted. After the deletion, the DUPKEY condition occurs if records still exist with the same alternate key.

If you want to release the string held by an EXEC CICS READ UPDATE command without rewriting or deleting the record, use the EXEC CICS UNLOCK command. This releases any CICS storage acquired for the EXEC CICS READ command and releases emulated VSAM resources held by the EXEC CICS READ command.

Deleting groups of records (generic delete)

You can use a generic key with the EXEC CICS DELETE command. Then, instead of deleting a single record, all the records in the file whose keys match the generic key are deleted with the single command. However, this cannot be used if the KEYLENGTH value is equal to the length of the whole key (even if duplicate keys are allowed). The number of records deleted is returned to the application program if the NUMREC option is included with the command. If access is by way of a logical view, the records deleted are all those whose alternate keys match the generic key.

Adding records

Add new records to a file with the EXEC CICS WRITE command. They must always be written from an area provided by the application program.

Adding to a KSDS

When adding a record to an emulated VSAM KSDS, the base key of the record identifies the position in the file where the record is to be inserted. Although the key is part of the record, CICS also requires the application program to specify the key separately using the RIDFLD option on the EXEC CICS WRITE command. If

the key specified in the RIDFLD option differs from the one embedded in the record, the embedded key takes precedence. You are not recommended to make use of this fact because the record locking protocol may be compromised.

A record added to a KSDS by way of a logical view is also inserted into the file in the logical position determined by the base key. The physical position is determined by OS/400. However, the command must also include the alternate index key as the record identifier.

Adding to an ESDS

A record added to an ESDS is always added to the end of the file, if the file was created with the OS/400 CRTPF command with the reuse deleted records (REUSEDLT) option set to *NO. You cannot insert a record in an ESDS between existing records. After the operation is completed, the relative byte address in the file where the record was placed is returned to the application program in the field specified in the RIDFLD option.

When adding a record to an ESDS by way of a logical view the record is also placed at the end of the file. The command must include the alternate key in the same way as for a KSDS path.

Adding to an RRDS

To add a record to an RRDS, include the relative record number as a record identifier on the EXEC CICS WRITE command. The record is then stored in the file in the position corresponding to the RRN.

Specifying record length

When writing to a fixed-length emulated VSAM file, the record length must match the value specified at the time the file was created. In this case you need not include the length with the command, although you may do so to check whether the length agrees with that originally defined to emulated VSAM. If the file is defined as containing variable-length records, the command must always include the length of the record.

Review of file control command options

Some of the file control command options you may specify are:

- RIDFLD
- INTO or SET
- FROM
- LENGTH

Use of the LENGTH option varies, depending on how you use the other options.

The RIDFLD option

Whatever you do to a record (read, add, delete (except when you have read the record for update first), or start a browse), you identify the record by the RIDFLD option. Further, during a browse using EXEC CICS READNEXT or EXEC CICS READPREV commands, you must include the RIDFLD option to give CICS a way to return the identifier of each record retrieved.

The RIDFLD option identifies a field containing the record identification appropriate to the access method and the type of file being accessed.

The RIDFLD option by itself isn't always enough to identify a specific record in the file. So, when retrieving records from an emulated VSAM KSDS, or an emulated VSAM KSDS or ESDS by way of an alternate index path, or when setting a starting position for a browse in this type of file, you can have one or both of the further options GTEQ and GENERIC with your command.

With EXEC CICS READNEXT or EXEC CICS READPREV commands, the application program would not usually set the RIDFLD field. After each command, CICS updates this field with the actual identifier of the record retrieved. (You can alter the RIDFLD value to set a new position from which to continue the browse.)

The INTO and SET options

With the EXEC CICS READ, EXEC CICS READNEXT, or EXEC CICS READPREV commands, the record is retrieved and put in main storage according to your INTO and SET options.

The INTO option specifies the main storage area into which the record is to be put.

For fixed-length records, you need not include the LENGTH option. If you do, the length specified must exactly match the defined length; otherwise, you get the LENGERR condition.

For variable-length records, always specify (in the LENGTH option) the longest record your application program accepts (which must correspond with the value defined emulated VSAM as the maximum record size when the file was created); otherwise, you get the LENGERR condition. LENGERR occurs if the record exceeds this maximum length, and the record is then truncated to that length. After the record retrieval, if you include the LENGTH option, the data area specified in it is set to the actual record length (before any truncation occurs).

The SET option specifies a pointer to the address of the buffer in main storage acquired by CICS to hold the record. When using the SET option, you need not include the LENGTH option. If you do include it, the data area specified is set to the actual record length after the record has been retrieved.

The FROM option

When you add records (using the EXEC CICS WRITE command), or update records (using the EXEC CICS REWRITE command), specify the record to be written with the FROM option.

The FROM option specifies the main storage area that contains the record to be written. In general, this area is part of the storage owned by your application program. With the EXEC CICS REWRITE command, the FROM area is usually (but not necessarily) the same as the corresponding INTO area on the EXEC CICS READ UPDATE command. The length of the record can be changed when rewriting to a emulated VSAM KSDS with variable-length records.

Always include the LENGTH option when writing to a file with variable-length records. If the value specified exceeds the maximum allowed in the definition, LENGERR is raised when the command is executed. LENGERR is also raised if the LENGTH option is omitted when accessing a file with variable-length records.

When writing to a file with fixed-length records, CICS uses the length specified in the definition as the length of the record to be written, so you need not have the

LENGTH option. If you do, its value is checked against the defined value and you get a LENGERR condition if the values don't match.

Avoiding transaction deadlocks

Design your applications so as to avoid transaction deadlocks. A deadlock occurs if each of two transactions (for example, A and B) needs exclusive use of some resource (for example, a particular record in a file) that the other already holds. Transaction A waits for the resource to become available. However, if transaction B isn't in a position to release it because it, in turn, is waiting on some resource held by A, both are deadlocked and the only way of breaking the deadlock is to cancel one of the transactions, thus releasing its resources.

A transaction may have to wait for a resource for several reasons while executing file control commands:

- Any record that is being modified is held in exclusive control by the access method for the duration of the request.
- If a transaction has modified a record in a recoverable file, CICS locks that record to the transaction even after the request that performed the change has completed. The transaction can continue to access and modify the same record; other transactions must wait until the transaction releases the lock, either by terminating or by issuing a syncpoint request. For more information, see "Syncpointing" on page 106.

Whether a deadlock actually occurs depends on the relative timing of the acquisition and release of the resources by different concurrent transactions. Application programs may continue to be used for some time before meeting circumstances that cause a deadlock; so it's important to recognize and allow for the possibility of deadlock early in the application program design stages.

Here are examples of different types of deadlock:

- Two transactions running concurrently are modifying records within a single recoverable file, through the same FCT entry, as follows:

```
Trans.1:  READ UPDATE  rec.1
          REWRITE     rec.1
Trans.2:  DELETE      rec.2
Trans.1:  WRITE       rec.2
Trans.2:  READ UPDATE  rec.1
          REWRITE     rec.1
```

Transaction 1 has acquired the record lock for record 1. Transaction 2 has similarly acquired the record lock for record 2. The transactions are deadlocked because each wants to acquire the lock held by the other.

- Two transactions running concurrently are modifying two recoverable files as follows:

```
Trans.1:  READ UPDATE  file 1 rec.1
          REWRITE     file 1 rec.1
Trans.2:  READ UPDATE  file 2 rec.2
          REWRITE     file 2 rec.2
Trans.1:  READ UPDATE  file 2 rec.2
          REWRITE     file 2 rec.2
Trans.2:  READ UPDATE  file 1 rec.1
          REWRITE     file 1 rec.1
```

Here the record locks have been acquired on different files as well as different records; however, the deadlock is similar to the first example.

When an application cannot get exclusive control of a record (for example, during a deadlock situation), the OS/400 database facilities wait for the record for a

maximum time as specified in the **maximum record wait time** (WAITRCD) defined for the file. If this time is exceeded, the OS/400 database facility issues a time-out exception and CICS returns the IOERR condition to your application program, qualified by additional OS/400 information in the EIBRCODE field.

You can avoid deadlocks by following these rules:

- All applications that update (modify) multiple resources must do so in the same order. For instance, if a transaction is updating more than one record in a file, it can do so in ascending key order. A transaction that is accessing more than one file should always do so in the same predefined sequence of files.
- An application that issues an EXEC CICS READ UPDATE command must follow it with an EXEC CICS REWRITE, EXEC CICS DELETE without RIDFLD, or EXEC CICS UNLOCK command to release the position before doing anything else to the file, and in any case as soon as possible.
- An application must end all browses on a file by means of EXEC CICS ENDBR commands (thereby releasing the position) before issuing an EXEC CICS READ UPDATE, EXEC CICS WRITE, or EXEC CICS DELETE with RIDFLD command, to the file.

KEYLENGTH option for remote files

In general, file control commands need the RIDFLD and KEYLENGTH options. The KEYLENGTH option can be specified explicitly in the command, or determined implicitly from the FCT.

For remote files for which the SYSID option has been specified, the KEYLENGTH option must be specified if the RIDFLD option is passing a key to file control. If the remote file is being browsed, the KEYLENGTH option is not required for the EXEC CICS READNEXT or EXEC CICS READPREV command.

Record identification

You have three ways to identify records in emulated VSAM files:

- Key
- Relative byte address (RBA)
- Relative record number (RRN)

Identifying records by key

Generally, if you use a key, you can specify either a complete key or a generic (partial) key. The exception to this rule is when you write a record to a KSDS, in which case you *must* specify the complete key in the RIDFLD option of the command.

When you use a generic key, you must specify its length in the KEYLENGTH option and you must specify the GENERIC option on the command. A generic key cannot have a key length equal to the full key length. You must define it to be shorter than the complete key.

You can also specify the GTEQ option on certain commands, for both complete and generic keys. The command then positions at, or applies to, the record with the next higher key if a matching key cannot be found. When accessing a file by way of an alternate index path, the record identified is the one with the next higher alternate key when a matching record cannot be found.

Even when using generic keys, always use a storage area for the record identification field that is equal in length to the length of the complete key. During a browse operation, after retrieving a record, CICS copies into the record identification area the actual identifier of the record retrieved. CICS returns a complete key to your application, even when you specified a generic key on the command. For example, a generic browse through an emulated VSAM KSDS returns the complete key to your application on each EXEC CICS READNEXT and EXEC CICS READPREV command.

Relative byte address (RBA) and relative record number (RRN)

You can use the RBA and RRN options on most commands. The RBA option can only be used on ESDS emulated VSAM files, whereas the RRN option can only be used on an RRDS file. In effect, they define the format of the record identification field (RIDFLD). Unless you specify either the RBA or the RRN, the RIDFLD option should hold a key to be used for accessing an emulated VSAM KSDS (or an emulated VSAM KSDS or ESDS by way of an alternate index).

RBA

RBA specifies that the record identification field contains the relative byte address of the record to be accessed. A relative byte address is used to access an emulated VSAM ESDS. All file control commands that refer to an ESDS base, and specify the RIDFLD option, must also specify the RBA option.

RRN

RRN specifies that the record identification field contains the relative record number of the record to be accessed. The first record in the file is number one. All file control commands that refer to an RRDS, and specify the RIDFLD option, must also specify the RRN option. The RRN option cannot be used on KSDS or ESDS emulated VSAM files.

CICS locking of emulated VSAM records in recoverable files

Whenever records within a recoverable file are modified, record locks are issued by the OS/400 database facility on behalf of the I/O operation performed by CICS.

Record locks of CICS recoverable files, along with data integrity, are managed by the OS/400 commitment control facility, even for files shared between two CICS systems or between CICS and other AS/400 jobs. When a CICS shell begins, the OS/400 commitment control environment is started by CICS with a lock level (LCKLVL) of *CHG. Record locks are issued whenever records in a recoverable file are modified. The locks are held for the transaction doing the change until it issues a syncpoint request or terminates (a syncpoint is automatically performed). For emulated VSAM recoverable file processing, note the following:

- Whenever an emulated VSAM record is obtained for modification or deletion, the record is locked by OS/400. This permits only one transaction at a time to modify or delete the record. Other transactions have to wait until the first transaction has reached a syncpoint, or the first transaction unlocks the record.
- For the EXEC CICS READ UPDATE and EXEC CICS REWRITE-related commands, the record lock is acquired as soon as the EXEC CICS READ UPDATE command has been issued.

If the record read with the EXEC CICS READ UPDATE command has not been modified or deleted, you can release the record lock with the EXEC CICS UNLOCK command.

For an EXEC CICS DELETE command that has not been preceded by an EXEC CICS READ UPDATE command, the record lock is acquired at the time the emulated VSAM command is run. The same is true for an EXEC CICS WRITE command.

For an EXEC CICS DELETE GENERIC command, each record deleted acquires a separate lock for the transaction issuing the request.

Note: If a transaction has either added or modified a record within a recoverable file but has not yet reached a syncpoint, that record can be read by another transaction if the UPDATE option of the EXEC CICS READ command was not specified.

Part 4. Data communication

Chapter 11. Introduction to data communication	133
Chapter 12. Introduction to basic mapping support (BMS)	135
How BMS affects programming	135
BMS maps	136
BMS map definition	136
Creating BMS map sets	137
Cataloging BMS map sets	137
BMS commands	138
Level of BMS	139
Base and towers architecture	139
Summary of support for CICS/400 BMS	139
Chapter 13. CICS/400 basic mapping support (BMS)	141
Information display systems	141
IBM 3270 Information Display System	141
IBM 5250 Information Display System	141
Input operations	141
Sending data	142
Modified data tags	142
Attention identifiers	142
Output operations	143
Display field concepts	143
Attribute character	144
Screen layout design	146
Screen sizes	147
Defining BMS maps	147
Defining a map set	147
Defining maps within a map set	147
Data fields	148
Maps without fields	148
Defining fields within a BMS map	148
Terminating a map set definition	148
Creating BMS maps	148
Symbolic description map	148
Physical map	149
Map set suffixing	149
Writing programs to use BMS services	151
Copying symbolic description maps	152
Data structures	152
Input map data structures	153
Input field suffixes	153
Output map data structures	153
Attribute constants	154
Incorrect data	155
Sending data to a display device	155
Composite displays	156
Refreshing and modifying displays	156
Getting storage for a data structure	156
Alternative data structures	158
Device control options	158
Cursor positioning	159
Normal cursor positioning	159
Initial display position	159
Symbolic cursor positioning	159
Accessing data outside the program	159
Receiving data from a display	160
Receiving data into an alternative data structure	160
Uppercase translation	161
Mapping data from another data area	161
Responding to terminal input	162
Exception conditions	162
The EIBAID field	162
The EXEC CICS HANDLE AID command	163
Text processing	164
Display characters in text	164
Control characters in text	165
Character attribute control (3270 devices only)	165
Unsupported attributes	166
Graphic data fields	166
Printed output	167
Using the hardware print key	167
Using asynchronous page build transaction	167
Printer formatting considerations	168
Blank lines and 3270 printers	168
Setting the printer page width	168
Form feed characters	168
Chapter 14. Terminal control	169
Terminal-oriented task identification	170
Logical unit communication protocol	170
Send/receive mode	170
Send/receive protocol (INVITE option)	171
Chaining the input data	171
Chaining the output data	171
Response protocol	172
Preventing interruptions (bracket protocol)	172
Handling attention identifiers (EXEC CICS HANDLE AID)	173
OS/400 display data streams	174
Terminal control and DBCS	174
Chapter 15. Intercommunication considerations	175
Design considerations	175
Transaction routing	175
Function shipping	176
Distributed program link (DPL)	176
Using the distributed program link function	177
Examples of distributed program link	178
Programming considerations for distributed program link	182
Issuing multiple distributed program links from the same client task	182
Sharing resources between the client program and server program	183
Mixing DPL and function shipping to the same CICS system	183
Mixing DPL and DTP to the same CICS system	183

Restricting a program to the distributed program link subset	183
Determining how a program was invoked	184
Exception conditions for EXEC CICS LINK command	184
Asynchronous processing	185
Distributed transaction processing (DTP)	186
Common Programming Interface Communications (CPI Communications)	186

Chapter 11. Introduction to data communication

You have two basic ways of communicating with the terminals and logical units in the network:

1. **Basic mapping support** provides commands and options that can be used to format data in a standard manner. BMS converts data streams provided by the application program to conform to the requirements of the devices. Conversely, data received from a device is converted by BMS to a standard form. However, not all devices supported by CICS can be used with BMS and, for those that cannot, terminal control must be used. In CICS/400 BMS, device support is provided only for 3270 and 5250 terminals, and for SCS printers. BMS is described in Chapter 12, "Introduction to basic mapping support (BMS)," on page 135 and Chapter 13, "CICS/400 basic mapping support (BMS)," on page 141..
2. **Terminal control** is the basic method for communicating with devices. BMS extends the facilities of terminal control to simplify further the handling of data streams. BMS uses terminal control facilities when invoked by an application program. Terminal control provides commands and options that can be specified in various combinations according to the requirements of the devices. However, application programs written in this way are dependent on the data formatting requirements of these devices, and a detailed knowledge of the devices is required. Terminal control is described in Chapter 14, "Terminal control," on page 169.

Intercommunication is another area of communication that you may be concerned with. It is described in Chapter 15, "Intercommunication considerations," on page 175.. The principal facilities in this area that you might use are:

- **Transaction routing** enables a terminal in one CICS system to run a transaction in another CICS system. See "Transaction routing" on page 175.
- **Function shipping** allows you to access resources in a remote system. See "Function shipping" on page 176.
- **Distributed program link (DPL)** enables an application program running on one CICS system to link to another application program running in a remote CICS system. See "Distributed program link (DPL)" on page 176.
- **Asynchronous processing** enables a CICS transaction to initiate a transaction in a remote system and to pass data to it. See "Asynchronous processing" on page 185..
- **Distributed transaction processing (DTP)** enables a CICS transaction to communicate with a transaction running in another system. See "Distributed transaction processing (DTP)" on page 186.

Chapter 12. Introduction to basic mapping support (BMS)

Basic mapping support (BMS) is an interface between CICS application programs and terminal devices, including printers.

A CICS application program can use BMS or terminal control commands to perform input and output. BMS has most of the facilities you need, and is easier to use than terminal control. However, you might sometimes need to use terminal control.

BMS lets you separate the tasks of display design and CICS application programming. It interprets generalized *device-independent* application program output commands, and generates *device-dependent* data streams for specific output devices. It also transforms incoming data streams to a form acceptable to application programs. BMS learns about the format of the data stream for the terminal from the terminal control table terminal entry (the TCTTE) for the task, not from the application program.

You can use the same BMS input or output commands in your application program to talk to different kinds of devices. A single BMS command in your program applies equally to various devices because BMS interprets commands differently for different device types.

BMS commands are quite simple, because all the low-level formatting information is held separately, in **maps**. This makes your application programs easier to write and less affected by changes to the system or its devices. You can change the layout of the information on the terminal device just by changing the map source and recreating the maps.

For the OS/400 programmer who is new to the concepts of the CICS family, BMS can be most easily described as similar to OS/400 data description specifications that would be used to describe a display image or printer file. See the DDS information in the Database and file systems topic in the iSeries Information Center for DDS reference information and the *ADTS for AS/400: Screen Design Aid* manual for further details.

BMS definitions are held in members of a physical source file just like the source for any other high-level language program on the iSeries. There is a version of this physical source file called QMAPSRC which you can use as a sample for your BMS maps.

How BMS affects programming

Different versions of a display map can exploit the features of the IBM 3270 or IBM 5250 terminals. By having the screen data in **fields** (that is, defining data as having **field** format), you can address predefined fields in a display symbolically by name from within your application, without knowing the actual screen positions of those fields.

Changing field data to and from its displayable form is called **mapping**.

Although the same fields must appear in all versions of a display image, you can move them around in the different versions. This is useful when you're

programming for a mixture of display screen sizes. See “Map set suffixing” on page 149 for information about applications that use multiple terminal types.

You can also display data in **text** format. This presents data as a series of lines on a display screen or printer. To format text data, BMS breaks it into strings that are, as nearly as possible, the same length as each line of the display device. BMS assumes that text is a sequence of words separated by blanks, and breaks the text into lines at an appropriate point.

BMS maps

Maps tell BMS how to format field data; they are not needed for text data. Every BMS mapping command names a map that contains formatting instructions. Each map has two forms, **physical** and **symbolic**.

Maps must belong to a map set. You usually group related maps together into one map set. You define a map set by coding a series of BMS macros. The first of these macros (DFHMSD) defines the map set itself; the second (DFHMDI) defines the first or subsequent maps. The third macro (DFHMDF) defines fields within a map.

The **physical map** tells BMS where the fields are to be placed on the terminal device, and what their attributes are.

A **symbolic description map** is a source language data structure that BMS runtime support uses to resolve source program references to fields in the map. Symbolic description maps are described in more detail in Chapter 13, “CICS/400 basic mapping support (BMS),” on page 141.

The CICS CL command CRTICSMAP generates both types of maps in one run. A standard OS/400 prompt screen is produced showing all available parameters and their defaults if you enter just the command name and press the PF4 key. Optional help text is also available if you press the PF1 key.

You could define a single map set to suit every particular terminal attached to a CICS system. However, you might want (or need) to format the same data differently for different devices. For example, the same transaction might be initiated from displays of various screen sizes.

BMS map definition

You define map sets, maps, and fields within maps with the following macros:

DFHMSD

Define a map set.

DFHMDI

Define a map.

DFHMDF

Define a field.

Use of these macros is explained in “Defining BMS maps” on page 147.

The macros define the size, shape, position (row and column), potential content, and characteristics (such as color, protected or unprotected, and bright or dark). You can optionally use them to define initial data for one or more fields to be displayed at a terminal. You should design the layout and content of a display before attempting to code the macros.

Always start a map definition with the DFHMSD macro, specifying TYPE=MAP, as follows:

```
DFHMSD TYPE=MAP
```

DFHMSD is always followed by a DFHMDI macro for the first (or only) map in the map set:

```
DFHMSD TYPE=MAP  
DFHMDI ...
```

After DFHMDI come the DFHMDF macros (if any) to define the individual fields within the map:

```
DFHMSD TYPE=MAP  
DFHMDI ...  
DFHMDF ...  
DFHMDF ...
```

If there is more than one map in your map set, repeat the sequence of DFHMDI and DFHMDF macros for each subsequent map in the set. End the map set definition with a DFHMSD macro with the TYPE=FINAL operand.

So for two maps, the sequence of macros may be as follows:

```
DFHMSD TYPE=MAP  
DFHMDI ...  
DFHMDF ...  
DFHMDF ...  
DFHMDI ...  
DFHMDF ...  
DFHMDF ...  
DFHMDF ...  
DFHMDF ...  
DFHMSD TYPE=FINAL
```

Specify attributes of map sets, maps, and fields by using operands in the appropriate macros. Some operands can be specified on both the DFHMDF and DFHMDI macros, and some can be specified on both the DFHMSD and DFHMDI macros. In such cases, specifying the operand on the DFHMDI (map definition) macro defines the default value for all fields within the map. For example, if you want to define a map with most fields blue, you can specify COLOR=BLUE on the DFHMDI macro; and only specify the color operand for the individual DFHMDF macros which need to be different. Similarly, operands on DFHMSD macro can define defaults for all maps within the map set and all fields within those maps.

Some facilities of 3270 devices, such as color, are not provided by all terminal models. Attempts to use a facility that the terminal does not provide are ignored. This means that different 3270 terminals do not necessarily need different maps. Some of the 3270 extended attributes are not supported on 5250 devices. Details are given in the descriptions of the operands in Appendix D, "BMS macro summary," on page 553.

Creating BMS map sets

The create CICS map (CRTCICSMAP) CL command is used to generate screens from BMS source files. For an explanation of the syntax and parameters available with this command, see page 301.

Cataloging BMS map sets

You can use the same set of DFHMSD, DFHMDI, and DFHMDF macros to define both the physical maps and symbolic description maps of a map set. When you

use the CRTCICSMAP CL command, physical and symbolic maps are created in the library. Symbolic maps are created either as COBOL copybooks in file QLBSRC or C header files in file H, depending on the value of the LANG parameter on the DFHMSD macro. You can select a different target file using the LMAPSRC and LMAPMBR parameters. The physical map is created as a user space object in the library. You can copy the symbolic map structure into any application program that refers to the map set before compiling a program.

Note: You must define a physical map as a processing program table (PPT) entry with CICSMAP(*YES) using CICS resource definition commands. The name of the map generated by CRTCICSMAP is the name that belongs in the PPT.

BMS commands

BMS performs input and output operations in response to commands in your application program. These commands are like other CICS commands, but they always name the map containing the mapping information. Application program statements, other than EXEC CICS commands, can refer to fields in a map by name. With BMS commands, not only can you read and change the contents of fields; you can also set or modify their attributes (for example, color or protected). The attributes are provided in the symbolic description maps for applications to set through normal application programming statements.

The BMS application programming interface provides the following functions for application programs.

- **Mapping**

The EXEC CICS SEND MAP command merges the data in the physical map (derived from the source map you defined) with data values set in the application data structure by the application program, and generates an appropriate device-dependent data stream to display this merged data at a terminal.

The EXEC CICS RECEIVE MAP command takes a device-dependent data stream from a terminal, and interprets it based on the specified physical map. When the positions (row and column) of a data-stream field and a map field match, BMS updates the corresponding part of the application data structure with the data-stream data.

- **Text processing**

The EXEC CICS SEND TEXT command provides very basic text support. BMS can split text into lines and pages that fit on the target display device. It honors new line characters embedded in the text and builds the display image such that words are not split across lines.

- **Device control**

BMS provides a “high-level” way of specifying device controls (such as: ERASE, ERASEAUP, and ALARM). You can specify them together with EXEC CICS SEND MAP or EXEC CICS SEND TEXT commands, or independently using EXEC CICS SEND CONTROL.

The Basic Mapping Support API commands are designed to support 3270 architecture. When CICS/400 applications communicate with DBCS-capable 5250 devices, the data streams are converted between 3270 and 5250 formats using iSeries system facilities.

Level of BMS

In some other CICS environments there are three pregenerated versions of BMS:

1. Minimum function
2. Standard function
3. Full function

In CICS/400, **minimum function BMS** plus the EXEC CICS SEND TEXT command is supported. Support is provided for 3270-2, 3270-5, 5250, and ASCII terminal devices, and for SCS printers. The other levels are listed here so you can understand their function. The levels of BMS have the following meanings:

- **MINIMUM**

This level is a small, fast, basic function mapping program. It has a substantially shorter path length than standard or full BMS and handles the most frequently used BMS requests.

In CICS/400, BMS supports 3270 display devices and printers, 5250 display devices and printers, and SCS printers. It supports extended attributes and large screens, but not partitions, outboard formats, or MSR control. It supports the EXEC CICS SEND MAP, EXEC CICS RECEIVE MAP, EXEC CICS SEND TEXT (subset), and EXEC CICS SEND CONTROL commands only, with no cumulative mapping, terminal operator paging, routing, or message switching.

- **STANDARD**

This level supports the EXEC CICS SEND MAP, EXEC CICS RECEIVE MAP, EXEC CICS SEND TEXT, EXEC CICS SEND CONTROL, EXEC CICS SEND PARTNSET, and EXEC CICS RECEIVE PARTN commands. It does not support cumulative mapping, terminal operator paging, or message switching.

- **FULL**

This level generates a full-function BMS, with full device support. You need this level of support if you need to use the BMS paging transaction CSPG and the message switching transaction CMSG.

Base and towers architecture

The architecture of the CICS family API has been defined as a base-and-towers specification, in the *CICS Family: API Structure* manual. In this book, the BMS levels of minimum, standard, and full disappear. CICS/400 BMS conforms to this architecture specification as follows:

BMS Application Programming Base

Supported in full

BMS Map Definition Base

Supported with minor deviations

All defined BMS subsets

None supported

Summary of support for CICS/400 BMS

The following is a list of devices and functions supported for BMS within CICS/400:

- Basic 3270 and 5250 display devices
- SCS printers
- Default and alternate screen sizes
- Extended attributes
- Formfeed control

- Command-level requests
- EXEC CICS SEND MAP
- EXEC CICS SEND CONTROL
- EXEC CICS RECEIVE MAP and EXEC CICS RECEIVE MAP FROM
- Subset of EXEC CICS SEND TEXT
- Map set suffixing
- Block data
- Automatic setting of write control character (WCC) line width
- ERASE, ERASEAUP, FORMFEED, CURSOR, and WCC options

Chapter 13. CICS/400 basic mapping support (BMS)

In CICS/400, BMS supports the IBM 3270 and IBM 5250 range of display devices and printers. See “Level of BMS” on page 139 for a definition of the level of BMS supported in CICS/400. This chapter introduces:

- The IBM 3270 and IBM 5250 display devices
- The principles of display layout design
- The way that you specify display layouts to CICS/400
- The commands and options BMS has for communicating with a display that has a predefined layout

Information display systems

The following describes the two types of information display systems supported by CICS/400:

IBM 3270 Information Display System

The 3270 data stream conveys both displayable data characters and nondisplayable control characters between the host processor and a terminal. To use BMS commands, you do not have to understand the format of the data stream. But you need to know what the data stream allows you to do. Refer to the *3270 Data Stream Programmer's Reference* manual for more information about the 3270 data stream, and the features available on 3270 terminals.

IBM 5250 Information Display System

The 5250 data stream, not unlike the 3270 data stream, provides displayable and nondisplayable control characteristics between the host processor and a terminal. CICS/400 provides the ability for the 5250 device to emulate a 3270 device. The 5250 keys are translated into 3270 function keys. The COBOL program refers to the 3270-type attributes and 3270 function keys. At run time, BMS translates the 3270-type data to 5250 data. The 5250 attention identifier (AID) keys are converted to 3270 AID keys and stored in the EIBAID field. The 5250 page-up, page-down, and HOME keys are not currently supported.

Input operations

The operations you perform at a 3270 or 5250 terminal need not always result in data being sent to the host processor. For example, your user can press the alphanumeric keys indefinitely without sending data. However, certain actions (such as pressing ENTER) always cause your terminal to send a data stream, even if you haven't provided any data.

Apart from the alphanumeric keys, other keys that you can press without sending data include:

- Repeat-action keys
- Forward and backward tabbing keys
- New line tabbing key
- Horizontal cursor positioning keys
- Vertical cursor positioning keys
- Backspace key

- Erase input (ERASE INPUT) key
- Erase end-of-field (ERASE EOF) key
- Insert mode (INS MODE) key
- Delete (DEL) key

These keys and special features of individual terminal models make it easier for you to enter data.

Sending data

When you have typed data on to a display and want to send it to the host processor, you can:

- Press the ENTER key
- Press a program function (PF) key

Although the display sends modified data when you press the PF keys, the PF keys are not normally used for this. Generally, the programmer has assigned a specific meaning to the key itself.

If a field is initialized by an output map or contains data from any other source, data that is entered as input overwrites only the equivalent length of existing data; any surplus existing data remains in the field and could cause unexpected interpretation of the new data.

Modified data tags

If you want to send data without having to enter it explicitly, you can set the modified data tag (MDT) for the required field. The MDT can be set by:

- An output command
- Entering data in the field

Note: Be aware that setting the MDT is device-dependent. It is possible that not all models support this function.

Attention identifiers

An attention identifier (AID) character is always sent to the host processor whenever an input operation is performed. This indicates the cause of the input operation.

CICS ensures that an application program receives the input data intended for it. The AID allows the application program to react differently, depending on the input operation. The effect of different combinations of data and AIDs depends entirely on the design of the application program.

If you want to get the attention of the host processor without sending data, you can generate an AID by:

- Pressing the CLEAR key, but remember that this erases any displayed map and data
- On a 3270, pressing a program access (PA) key; on a 5250, sending a record backspace AID by pressing the HOME key once or twice, depending on the current cursor position. See note 2 for more information about the record backspace AID

In some cases, the keys on the 5250 and 3270 keyboards that cause an AID to be sent are different. The interpretation of 5250 AIDs by BMS is shown in Table 7 on page 143.

Table 7. Correspondence between 5250 and 3270 AIDs

5250 AID	Interpreted as
PF1 - PF24	PF1 - PF24
ENTER	ENTER
CLEAR	CLEAR
Record backspace	PA1 (See notes)
Help	PF1

Notes:

1. PA2 and PA3 are not emulated on 5250 devices.
2. The record backspace AID is generated when the HOME key is pressed, provided that the cursor is in the home position. For 5250 terminals, the home position on a blank CICS screen is (1,2). Otherwise, the home position is at the start of the first unprotected field on the screen. If the cursor is not in the home position, pressing the HOME key moves the cursor to the home position but does not generate an AID. The HOME key must be pressed a second time to generate the AID.

Output operations

A terminal can receive data from an application program, or send data back to the program. Some of the data can be displayed; the rest consists of device controls. Through data streams containing device controls built by BMS, you can, for example:

- Sound the audible alarm (if the terminal has one)
- Reset the modified data tag (MDT) of each field
- Print the contents of a display screen
- Erase all unprotected fields
- Position the cursor

The way you use these features is up to you. However, they can improve the usability of your application program.

Display field concepts

Using BMS maps, a screen of data can be divided up into fields. Application programs interact with these fields by using the symbolic description maps associated with the BMS maps.

A field starts with an attribute character, continues with data characters, and ends at the next attribute character. A field may contain a single character only or it can span several lines; the last character on a line is logically followed by the first character on the next line.

If the screen width is the same as the map width, BMS allows a field to “wrap around” from the end of one line to the start of the next. Because of its dependence on resource definitions, an application design should not rely on this function.

Normally a display image is divided into several fields by the program, but it is possible to have an unformatted input (no SBA characters), or an AID key with no user input data. An unformatted data stream can occur if a CLEAR key was

previously pressed to erase the display image. Pressing the CLEAR key clears not only the visible data, but also unformats the display screen.

Note: Set buffer address (SBA) is a data stream control field that is used in both outbound and inbound data streams. An SBA is indicated by X'11'. Outbound data streams contain SBAs to describe the screen positions where user data is to be placed. SBAs in the inbound data stream describe to BMS what fields were entered by the user.

When an attention key is pressed, such as CLEAR, PA1, PA2, or PA3, only the AID key and cursor position is transmitted. No user-entered data or MDT fields are transmitted. Both above situations cause a MAPFAIL condition to be raised.

An application programmer can use the EXEC CICS HANDLE CONDITION command to detect a MAPFAIL condition. The EXEC CICS HANDLE CONDITION command is described in Chapter 6, "Dealing with exception conditions," on page 87,, and the EXEC CICS HANDLE AID command is described in "The EXEC CICS HANDLE AID command" on page 163.

Attribute character

The attribute character is always the first character of a field. It occupies a character position on the display screen but appears as a blank.

Attribute bytes can convey the following field characteristics:

- **Unprotected**

You can enter any keyboard character into an unprotected field.

- **Numeric-only**

A numeric-only field is unprotected and only the digits 0 through 9 and the special characters period, dash, and DUP may be entered. If the keyboard numeric lock feature is installed on the 3270 and the operator attempts to enter any other characters, the keyboard is locked. If the keyboard numeric lock feature is not installed, any data can be entered in the field. On a data entry keyboard, a numeric-only field causes a numeric shift to occur.

- **Protected**

Data cannot be entered in a protected field. If the operator attempts to enter data, the keyboard is locked. A 1-byte protect field (stopper field) is usually defined following an input field. This protects the operator from entering more characters than the input field allows.

- **Autoskip**

An autoskip field is a protected field that automatically skips the cursor to the next unprotected field. Keyword fields and stopper fields following fixed-length data fields are normally defined with autoskip attribute characters.

Note: The unprotected, numeric-only, protected, and autoskip characteristics of the attribute character are mutually exclusive. Only one may be selected for each field.

- **Normal intensity**

A normal-intensity field displays the data at the normal operating intensity.

- **Bright intensity**

A bright-intensity field displays the data at a brighter than normal intensity. This feature is often used to highlight keywords, errors, or operator messages.

- **Nondisplay**

A nondisplay field does not display the data on the screen for operator viewing. Nor can you print the data contained in a nondisplay field. Nondisplay fields might be used to enter security data when the screen is visible to others. This attribute characteristic should be used with care, because the operator loses the ability to verify the data entered in a nondisplay field. This field might also be used to store messages on the display screen. The messages can be displayed later by changing the attribute character to bright or normal intensity.

Note: The normal, bright, and nondisplay characteristics of the attribute character are mutually exclusive. Only one may be selected for each field.

- **Base color**

The IBM 3279 Model 2A or 3A display device (3270 terminal with extended attributes) produces a **base color** image by using the PROTECT and INTENSIFY attributes of the 3270 standard data stream to select four colors: white, red, blue, and green. A switch on the display control panel permits the operator to select default color, causing the display to behave as a monochrome 3270 display, with white representing INTENSIFY. The protect bit retains its protect function when conveying color information. You can change the field colors of input, intensified input, protected, and intensified-protected fields on some devices.

The IBM 5292 (limited color) displays white when representing INTENSIFY. Specific settings on other models are device-dependent.

- **Extended color**

The IBM 3279 Model 2B or 3B, and IBM 5292 (full color) use **extended color** attributes in an extended data stream to determine the colors of display elements. The data stream can specify the colors of multicharacter fields. Seven colors can be selected: blue, red, pink, green, turquoise, yellow, and neutral. Selecting **neutral** sets the field to the default color for the specific device (usually white for terminals and black for printers).

An IBM 3279 Model 2B or 3B acts as a Model 2A or 3A until it detects an extended color attribute byte in the data stream. It displays the image in default color or base color, according to the setting of the switch on the control panel.

If an extended color attribute is received, the display treats the whole image as an extended color image. Fields that have no color attribute adopt the default colors (green for normal intensity, white for bright). If the color control switch has been set to base color, the part of the image that has already been displayed changes from base color to default color. Such a change, which could disturb an operator, can be avoided by applying an extended color attribute to the first field in any image that uses extended color.

The device interprets extended color attributes to determine the colors of fields in an image.

- **Extended highlighting**

Extended highlighting can be applied to characters, or character fields, in a display that uses the extended data stream. It can take one of three forms: BLINK, REVERSE, or UNDERSCORE.

- **Modified data tag (MDT)**

The modified data tag is turned on whenever fields are modified by the operator. When the operator presses the ENTER key or a PF key, only fields that have been modified by the operator or selected by the cursor select are transmitted to the processor. The program may send fields to the terminals with the modified data tag already on to guarantee that the field is returned with the next inbound transmission.

Note: Causing data to be sent back is not necessarily an efficient way to pass data from program to program.

- **Insert-cursor indicator**

The insert-cursor indicator is not a field attribute. Instead, it places the cursor under the first data character of the field whenever the map is sent with the ERASE option. If the insert-cursor indicator is specified for more than one field, the cursor is placed under the first data character of the last field specified. If no insert-cursor indicator is specified, the cursor is placed at position zero (row one, column one) on the display screen.

Note: If symbolic cursor is explicitly specified, this overrides the insert-cursor indicator option.

- **Background transparency**

Background transparency determines whether the background of an alphanumeric field is transparent or opaque; that is, whether an underlying (graphic) presentation space is visible between the characters.

This feature is not supported by the 5250 architecture. However, you may define this feature in maps because BMS will prevent the attribute being placed in output data streams to a 5250 device.

- **Field outlining**

Field outlining allows lines to be included above, below, to the left, and to the right of a field. You can use these lines in any combination to construct boxes around fields or groups of fields.

- **SO/SI creation**

SO/SI creation indicates that the field may contain a mixture of single-byte and double-byte character set (DBCS) data. The DBCS subfields within an EBCDIC field are delimited by SO (shift out) and SI (shift in) characters. SO and SI both occupy a single screen position (normally displayed as a blank). They can be included in any non-DBCS field on output if they are correctly paired. The terminal user can transmit them inbound if they are already present in the field, but may only add them to an EBCDIC field if the field has the SOSI attribute.

Not all devices support all the attributes. BMS ensures that attributes not supported by the device (as specified in the terminal definition) are ignored when building the data stream.

Screen layout design

The features of the 3270 system allow screen layouts to be designed for operator convenience and efficiency. The success of an online system depends on its ease of use, screen clarity, and terminal operator acceptance.

Refer to the following manual for information about screen layout design and user dialog flows:

- *SAA CUA Basic Interface Design Guide, SC26-4583*

The following features of some 3270 and 5250 displays make it easier for the layout designer to meet the requirement of ease of use:

- Color
- Field highlighting
- Ease of correction
- Numeric shift for numeric data

- Validation
- Field delimiters or stoppers (to control the length of data entered)

The maximum number of fields that may be sent to a 5250 display device is 256. You should bear this limit in mind when designing screens for use on 5250 devices.

Screen sizes

Screen sizes supported are 24 X 80 and 27 X 132.

The system administrator uses the CEDA transaction (see the *CICS for iSeries Administration and Operations Guide*) or native OS/400 CL commands (see the *CICS for iSeries Administration and Operations Guide*) to:

- Specify the alternate screen sizes, by means of the ALTSCN parameter on the ADDCICSTCT or CHGCICSTCT CL command.
- Specify whether the default or alternate screen size is to be used by the transaction, by means of the SCRNSZE parameter on the ADDCICSPCT or CHGCICSPCT CL command.

Defining BMS maps

This section describes the three macros DFHMSD, DFHMDI, and DFHMDF. They are used to define BMS map sets, maps, and fields. It shows how to use the macros to define a simple map set, and how to create this map set for use by application programs.

You must define all maps (including a single map) as part of a map set. You always start your map set definition with a DFHMSD TYPE=MAP or TYPE=DSECT macro, and must always end it with a DFHMSD TYPE=FINAL macro. Labels start in column 1, macros start in column 10, operands start in column 15, and continuation characters are placed in column 72, with the continued operands in column 16. Comments may be included in the source by coding an asterisk (*) in column 1, followed by the comment text.

Defining a map set

You use the DFHMSD macro to define a set of maps (a **map set**). The macro consists of operands that define characteristics of the map or maps in the map set. Some of the operands of DFHMSD macros establish defaults for subsequent DFHMDI and DFHMDF macros. The map set source files can be stored before assembly.

For information about the syntax of the DFHMSD macro, see page 555. You should also see “Getting storage for a data structure” on page 156 for information about storing maps, and “Creating BMS maps” on page 148 for information about suffixing map set names.

Defining maps within a map set

Each map in a map set is defined using the DFHMDI macro. This macro is similar in form to DFHMSD and specifies defaults for fields within the map. It allows you to override some of the options inherited from DFHMSD, and to specify some new ones. For information about the full syntax of the DFHMDI macro, see page 562.

A map set definition must contain at least one map definition. Where you have more than one map, you code their definitions one after another, the end of one being marked by the next DFHMDI macro or by a DFHMSD TYPE=FINAL macro.

All maps in a map set are loaded whenever any one of them is used. If all the maps in a map set are used during a single invocation of the program, the single load of all maps is more efficient than loading each map as it is required. You should ensure that you use unique names for maps within a map set, or within multiple map sets that are copied into an application program.

Another reason for loading several maps at the same time is that more than one of them can appear on the screen at one time. This is because a map definition can specify where a map is to be placed on the screen. When BMS sends a map to a display, it does not erase the existing contents of the display unless you code the ERASE option. Instead, it uses your program data, plus constant map data, to overlay part of the display screen. If you design your maps so that they occupy different parts of a screen, you can display them at the same time. Alternatively, you can design some maps in a map set so that they overlay one another and allow you to erase parts of the contents of the screen without affecting the rest of the screen.

Data fields

A map usually consists of one or more data fields. Each field contains display data, and has a set of associated attributes that are initialized by coding operands in a DFHMDF macro. All field definition macros following a map definition macro belong to that map. The end of one field definition is indicated by the beginning of another, by the next DFHMDI macro, or by a DFHMSD TYPE=FINAL macro.

Maps without fields

You can define maps that have no fields. You do this to reserve part of the screen for use by another program. By defining such a null map, you ensure that BMS has no effect on data that appears in the reserved part of the screen.

There are other considerations when coordinating the use of a screen between BMS and other programs; see "Accessing data outside the program" on page 159.

Defining fields within a BMS map

The DFHMDF macro is used to specify initial attributes to be given to fields within a map. For the syntax of the DFHMDF macro, see page 566.

Terminating a map set definition

The macro DFHMSD TYPE=FINAL terminates a map set definition. For the full syntax of the DFHMSD macro, see page 555. If you specify the name of the map set, it must match that specified in the DFHMSD macro.

Creating BMS maps

You create a BMS map definition, generating a symbolic description map (sometimes referred to as a logical map) and a physical map.

Symbolic description map

A symbolic-description map-set definition is created as either a COBOL copybook or C header file containing the record layout or structure, depending on the value of the LANG option on the DFHMSD macro. It is stored as a member of a source file defined by the LMAPSRC and LMAPMBR parameters of the CRTICSMAP CL

command. The member name is usually the same as the name of the map set, but it need not be. The symbolic description map must be copied into the application program before it can be used.

Physical map

A physical map-set definition is created as *USRSPC in the library specified.

When you create the physical map set, you should consider whether to add a suffix to its name (specified with the SUFFIX operand on the DFHMSD macro). The reason for suffixing a map set is that you might want to produce alternative versions of it (each with a different suffix) for different terminal models.

Map set suffixing

If you want to execute the same transaction from more than one type of terminal, you might need to use BMS map set suffixing. If you are prepared to use the same map to format data for all your terminals, you need not read the rest of this section. If however, you want to organize output data according to the terminal in use, making best use of its features, you ought to consider suffixing map sets.

To avoid problems at the creation stage you should:

- use the SUFFIX or TERM operand on your DFHMSD macros (you can safely use the same name for your map set and your maps)
- ensure that your physical maps are created with the correct suffixes
- ensure that the PPT entry for the physical map uses the correctly suffixed name

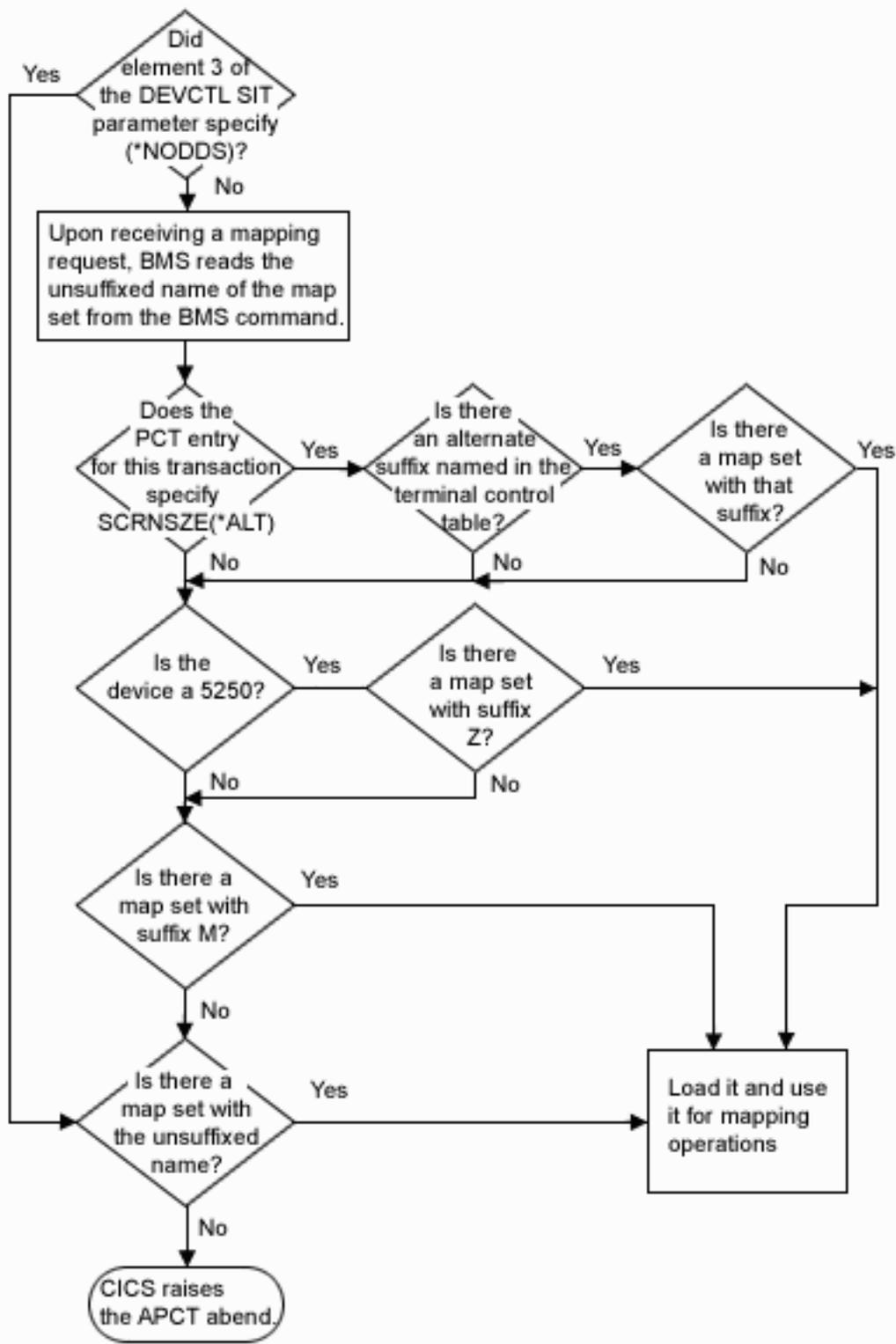


Figure 29. BMS map set suffixing logic

If you have displays with screens of different sizes, you might want to arrange display fields differently for each size of screen, ensuring that each display appears “balanced”. For different versions of the same map, fields must be in the same

order in each map definition macro, but the screen order can be different for each version. You add a different suffix to each version of the same map. You can also use map set suffixing to send maps to another terminal in a different language. For example, you may want to create the map with a suffix that denotes German, and then send a map to a terminal or signed-on user in Germany.

When a mapping operation is requested by a BMS command, CICS adds a suffix to the map set name specified in the command, and attempts to load a map set with that suffixed name.

The search for a suitable map set is carried out as follows:

1. If the terminal is a 5250, CICS looks for a map set with the specified name suffixed with Z.
2. If this test fails or if the device is a 3270, CICS looks for a map set with the specified name suffixed with M.
3. If this test fails, CICS looks for a map set with the specified name and no suffix.

Although a map set created specifically for use with a 5250 terminal is suffixed with Z, CICS allows for the fact that the 5250 may be in use only as a 3270 emulator and therefore a map set suffixed with M or an unsuffixed map set, would be valid.

Display devices can be of two screen sizes: 80 characters by 24 lines, or 132 by 27. For example, a 3278 Model 5 display device has a screen that is 132 characters wide, and 27 lines deep. If your terminal has different characteristics, you may use a suffix of your own choice, using the SUFFIX operand. Map set suffixing can provide separate physical map layouts with the same symbolic description map definition. If you do not need to distinguish between maps for the two types, you need produce only one version, and should give it an unsuffixed name.

You can code SUFFIX, instead of TERM, on DFHMSD if you need to create a special version of a map. By specifying SCRNSZE(*ALT) when you define the PCT entry for the transaction that uses the map, you tell BMS to try to load a special version of the map. This is the version of the physical map whose suffix is specified by the ALTSUFFIX operand of the TCT entry for the terminal. Table entries such as these are usually defined by a system administrator.

If all your map sets are unsuffixed, you get better performance if NODDS is specified in the device dependent suffixing element (element 3) of the DEVCTL parameter of the ADDCICSSIT command. However, if your system has been initialized with the default DDS option, you will get better performance if all your map sets are suffixed. Figure 29 on page 150 shows why this is so.

Writing programs to use BMS services

The layout of a BMS input or output display is defined by one or more maps. These can define display data fields that can be addressed by name from the application program. This means that the attributes (that is, color, highlighting, and so on) and contents of such fields can be changed dynamically.

Application programs use the BMS EXEC CICS SEND MAP and EXEC CICS RECEIVE MAP commands to send and receive display data. The remainder of this

chapter shows the syntax of these and other commands, and demonstrates their use. It also explains how to produce a printed copy of a screen image, and to output data to a printer.

Copying symbolic description maps

The section “Defining BMS maps” on page 147 describes how to define the symbolic version of a map set. The created version of a map set (the symbolic storage definition) is an application data structure that must be copied into any application program that refers to fields in its maps.

The following example shows you how to copy these structures for each programming language. In this example, `mapsetname1`, `mapsetname2`, and `mapsetname3` are the names of members in the source library that contain BMS symbolic map definitions. These member names are the same as the names used for the symbolic description maps, as described in “Creating BMS maps” on page 148..

- A COBOL/400 program should contain a COBOL/400 COPY statement for each symbolic map definition. Generally, you should code the COPY statements in the working-storage section of a program:

```
WORKING-STORAGE SECTION.  
COPY mapsetname1.  
COPY mapsetname2.  
COPY mapsetname3.  
:  
:
```

If the maps are located in the linkage section, storage must be allocated first if an EXEC CICS SEND MAP FROM command is issued before an EXEC CICS RECEIVE MAP command. BMS only preallocates storage for the SET option at receive time, not send time.

- A ILE C program must contain a #include statement for each symbolic storage definition:

```
#include mapsetname1;  
#include mapsetname2;  
#include mapsetname3;  
:  
:
```

Data structures

The symbolic map data structures that result from executing map and field definition macros contain extended versions of the fields, each one consisting of subfields. Each subfield can be referred to by the name assigned to the field, plus a single-letter suffix. Each kind of subfield has a different suffix.

Furthermore, the entire input or output data structure can be addressed by its suffixed name. The suffixed name of an input map is its original name extended by the suffix “I”. The corresponding suffix for the output map is “O”.

In this example, the 12-byte prefix is generated for compatibility with the terminal input-output area (TIOA) prefix option.

```

01 B169MAPI
02 FILLER PIC X(12).
02 TRANSL BINARY PIC S9(4).
02 TRANSF PICTURE X.
02 FILLER REDEFINES TRANSF.
03 TRANSA PICTURE X.
02 TRANSI PIC X(4).
02 NUMBL BINARY PIC S9(4).
02 NUMBF PICTURE X.
02 FILLER REDEFINES NUMBF.
03 NUMBA PICTURE X.
02 NUMBI PIC 9999.
01 B169MAPO REDEFINES B169MAPI.
02 FILLER PIC X(12).
02 FILLER PICTURE X(3).
02 TRANSO PIC X(4).
02 FILLER PICTURE X(3).
02 NUMBO PIC X(4).

```

Figure 30. Some suffixes and subfields

Input map data structures

The suffixes used to address subfields, and the contents of those subfields, in input maps are:

- F** A flag byte. This is normally set to X'00'. If the field has been modified but no data is sent (that is, the field is cleared), the flag byte is set to X'80'.
- I** Input data read from the display. It is set to X'00' if no data is entered for that field.

If the previous map sent specified FSET for this field, the data previously sent is returned even if the operator did not key into this field.
- L** A halfword binary length value. This defines the number of characters that are returned into the data field before the application program refers to it.

Input field suffixes

Having read data, a program can process it by issuing ordinary application programming commands that address fields by name.

Consider a field, called INPUT, in an input map. A program can test that either its length field INPUTL contains a value greater than 0 (data has been entered) or that its flag byte INPUTF indicates that the field has been cleared. If one of these conditions is true, it can, for example, move the first INPUTL characters from INPUTI to another data area.

The suffix on the data structure for the entire map enables you to manipulate the entire data structure. For example, you can write simple commands to copy the entire structure into another data area.

Output map data structures

The suffixes used to address subfields, and the contents of those subfields, in output maps are:

- A** An attribute byte defining the characteristics of the field (for example, protected or unprotected).
- C** An attribute byte specifying the color of the field.

- H** An attribute byte defining the extended highlighting to be used for the field.
- M** An attribute byte defining that SO/SI creation is to be used.
- O** Output data to be sent to the display. The program usually stores data in such a field before sending the map. If the contents of the field begin with a null character (X'00') the entire field is ignored, and the contents of the display field are taken from the physical map. If you want to send a blank field, you must store blanks (X'40') in the symbolic map data structure. Being nonnull, this overrides the contents of the physical map.
- P** An attribute byte defining the programmed symbol set to be used within a field in a display. CICS/400 supports only PS=8.
- U** An attribute byte defining the outline to be used.
- V** An attribute byte defining the kind of validation to be performed on data typed into a display field.

If MODE=INOUT is specified, the "fieldnameA" subfield is defined in the input map data structure. (In COBOL/400, compiler errors occur if a MOVE statement modifying an attribute byte is qualified to refer to the output map.)

Subfields with suffixes C, H, M, P, and U are generated if specified in the DSATTS operand of the DFHMDI and DFHMSD macros in the BMS map source.

As with input data fields, a program can address individual subfields in an output field, verifying or changing their contents. For example, an application program can check a calculated data value, for example, BALANCE. If the value is found to be negative, the color attribute constant BALANCEC in a field called BALANCE can be set to produce red characters when displayed. The data value in the field occupies subfield BALANCEO.

You can also manipulate the entire output data structure using its suffixed name. For example, you can copy data into it from another area. You are recommended to write commands to set the entire data structure to nulls (X'00') before using its corresponding physical map in an output operation. By doing this, you ensure that fields and attributes in the output display inherit the default contents of the physical map, not whatever happens to be in the symbolic data structure. Figure 31 shows how you might do this.

```

COBOL/400      MOVE LOW-VALUES TO MAPO
ILE C          memset(mapo,X'00' sizeof(mapo));
```

Figure 31. Setting output map data structure to nulls

Attribute constants

Subfield suffixing allows an application program to change the data within a data structure. However, the bit patterns representing particular attributes are difficult to remember, so CICS provides a list of named standard attribute bytes. You can code these names in a program instead of their hexadecimal equivalents. To use them, you must copy the list into your program, using the name DFHBMSCA, or you may set up your own copybook with names that are meaningful to you. For information about the constants and their meanings, see Appendix B, "BMS-related constants," on page 545.

Using attribute constants and subfield suffixing, a program can modify field attributes using simple commands. Figure 32 gives an example of how you could (1) put data into an output data field, (2) set the color attribute of the output data field, and (3) set the highlighting attribute of the output data field:

```
COBOL/400      MOVE CUSTNO TO ACCOUNTO.... (1)
                MOVE DFHBLUE TO ACCOUNTC... (2)
                MOVE DFHBLINK TO ACCOUNTH.. (3)
ILE C          accounto=CUSTNO;.... (1)
                accountc=DFHBLUE;... (2)
                accounth=DFHBLINK;.. (3)
```

Figure 32. Modifying map field attributes

Additional installation-defined named attribute constants can be created in DFHBMSCA in the source library.

Incorrect data

BMS does not check the validity of attribute and data values in the symbolic data structure. However BMS does ensure that attributes are not sent to terminals that do not support them. Incorrect data may be transmitted to the terminal. Some terminals can detect this incorrect data and send error information to CICS. This error information is handled by CICS code and can result in an abnormal termination of the transaction with an ATNI abend code.

Sending data to a display device

You use the EXEC CICS SEND MAP command to send mapped data to a display device.

You are recommended to use this command in preference to the EXEC CICS SEND TEXT command when sending data to DBCS capable devices, particularly when the device is a 5250.

If you use DBCS data, you can use the EXEC CICS SEND MAP command to send maps to 3270 displays without any problem. However, when the map is sent to a 5250 display, the right most character is lost. You should consider reserving the right most position in fields defined as PS=8 as a *redundant* character.

You can send three kinds of data using the EXEC CICS SEND MAP command, depending on the options you specify, as follows:

1. **Constant display data** (with attributes) such as headings, footings, prompt fields, and comments
2. **Variable display data** (with attributes) such as user data or warning messages
3. **Device control data** such as instructions to clear the screen or sound an alarm before displaying data

For the full syntax of the EXEC CICS SEND MAP command, see page 436.

The MAP option names the map that is used to format the data, and the MAPSET option names the map set where the map belongs.

If the MAPSET option is omitted in an EXEC CICS SEND MAP command, the name in the MAP option is taken as the map set name, and the map set is assumed to contain a single map. If the map set has more than one map, the first is used, regardless of individual map names.

In its simplest form, the EXEC CICS SEND MAP command is used as follows:

- The application program assigns values to variables named in the symbolic description map.
- The program issues an EXEC CICS SEND MAP command. This uses the application data in the application data structure to replace default data and attributes in the physical map, and sends the modified map to the display.

For example, if the first map in a map set called DISPLAY is an output map of the same name, the map can be displayed using the command:

```
EXEC CICS SEND MAP('DISPLAY')
```

Another map, called ERROR, in the same map set can be displayed by:

```
EXEC CICS SEND MAP('ERROR') MAPSET('DISPLAY')
```

Note: The omission of the MAPSET option in an EXEC CICS SEND MAP command is not recommended, except when the map set contains only one map. By default, BMS displays application data or attribute data from the application data structure rather than default data from the physical map. To override this for a given field, your program must set the corresponding subfield in the data structure to hexadecimal zeros (X'00') before issuing an EXEC CICS SEND MAP command.

Composite displays

If your program sends a succession of maps to a display device, the final form of the display depends on both the design of the maps, and the form of the EXEC CICS SEND MAP command. For example, if the final map fills the screen, or the EXEC CICS SEND MAP command includes the ERASE option (see “Device control options” on page 158), it obliterates all previous output. However, if you design your maps to occupy different parts of the screen, or to overlay each other only partially (see “Defining maps within a map set” on page 147), you can combine them to produce the final display.

Refreshing and modifying displays

You use the MAPONLY option of the EXEC CICS SEND MAP command to build a display using data from the physical map, without inserting user data. This can be useful when sending a menu to a display device, because no data is sent with the map, and input data fields regain their default data values (perhaps blank).

You use the DATAONLY option to modify the variable data in a display image that has already been created by a previous EXEC CICS SEND MAP command. BMS transmits variable data but no physical map data.

No data is sent for fields that you have cleared to nulls (X'00'). You can use EXEC CICS SEND MAP DATAONLY to ensure that only changed fields are sent. The value X'00' is a valid field attribute only on 3270 devices. (It means UNPROT, NORM, or NO MDT.) If you wish to change a field's attribute to UNPROT, NORM, or NO MDT on a 5250 device, you should use the graphic character equivalent, which is X'40'. All the graphic character equivalents for BMS attributes are defined in Appendix B, “BMS-related constants,” on page 545.

Getting storage for a data structure

You have now seen how to map data from one or more data structures. Depending on how you define your map sets, a program might have to issue commands to acquire main storage for the data structures it uses. It does this by issuing EXEC CICS GETMAIN commands. You can usually avoid having to code EXEC CICS GETMAIN commands by coding STORAGE=AUTO on the DFHMSD macro.

It has been assumed so far in this chapter that every output map has its own data structure. However, you might decide that this uses too much storage. To save storage, you can specify that different maps are to use the same storage area. You do this by coding `BASE=name` (or nothing at all), instead of `STORAGE=AUTO`, on the `DFHMSD` macro. This section describes what happens when you code each operand, and how it affects application programs. For the full syntax of the `BASE` operand, see “`DFHMSD`” on page 555.

Remember that, however you acquire storage, you should clear its contents (to `X'00'`) before issuing an `EXEC CICS SEND MAP` command. If you do not do this, existing data in storage can modify the output display unpredictably. If you use an `EXEC CICS GETMAIN` command to acquire storage, you can clear the storage by coding the `INITIMG` option.

COBOL/400:

STORAGE=AUTO

The data structure must be copied into the working-storage section. CICS acquires storage automatically for every map; you do not have to code an `EXEC CICS GETMAIN` command.

BASE=name The map set must be copied into the linkage section. You must code an `EXEC CICS GETMAIN` command to acquire enough main storage to contain the largest map in the set.

Nothing specified

If the map set is copied into the working-storage section, you don't have to code an `EXEC CICS GETMAIN` command, but you should place the largest map first in the set.

If the map set is copied into the linkage section, you must code an `EXEC CICS GETMAIN` command to get storage for it.

Note: When you use an `EXEC CICS GETMAIN` command to get main storage for a COBOL/400 map, you must ensure that you establish addressability for the map. For further information about the `EXEC CICS GETMAIN` command, see page 367.

ILE C:

STORAGE=AUTO

specifies that the symbolic description maps are to be declared as having the `AUTOMATIC` storage class. If `STORAGE=AUTO` is not specified, they are declared as pointers. You cannot specify both `BASE=name` and `STORAGE=AUTO` for the same map set. If `STORAGE=AUTO` is specified and `TIOAPFX` is not, `TIOAPFX=YES` is assumed.

BASE=name You must code an `EXEC CICS GETMAIN` command that gets at least enough main storage to contain the largest symbolic map in the map sets sharing this base.

The name specified in the `BASE` operand is used as the name of the pointer variable on which the symbolic description map is based. If you omit this operand, the default name (`BMSMAPBR`) is used for the pointer variable. You must establish addressability for the based structures.

Nothing specified

You must code an `EXEC CICS GETMAIN` command that sets the

pointer BMSMAPBR to the address of the acquired data area. The EXEC CICS GETMAIN command must get at least enough storage to contain the largest map in the sets.

Alternative data structures

The examples so far have shown EXEC CICS SEND MAP commands that contain literal map names. If the map name referred to by your program is to be a variable, you need to code additional options, FROM and LENGTH, on the EXEC CICS SEND MAP command. Also, you may want to use your own data area rather than the data structures from the symbolic description map, even when you use a literal map name.

FROM enables you to display data stored in a data area other than the data structure for the symbolic description map. In the command syntax summary, "data-area" represents the name of the alternative data area.

FROM and MAPONLY are mutually exclusive.

LENGTH specifies the length of the data string stored in the FROM data area. You must specify the LENGTH option if the data to be mapped is shorter than the data area expected by the map.

Device control options

In addition to transmitting application data to a display, BMS can relay device control commands. An application program uses options on the EXEC CICS SEND command to specify which controls are to be activated. Alternatively, it can use the BMS EXEC CICS SEND CONTROL command, which transmits device control commands without also sending application data.

For example, the following EXEC CICS SEND MAP command erases the screen before data is displayed.

```
EXEC CICS SEND MAP('ERROR') MAPSET('DISPLAY')
          ERASE
```

You can code one or more of the following device control options in an EXEC CICS SEND MAP command:

- ALARM** Sound audible alarm on displaying data.
- CURSOR** Specify position of cursor after output. The cursor position is a halfword binary value, representing the absolute screen address of the cursor. However, you need not always specify a value. For more information, see "Cursor positioning" on page 159. If a value is specified for the cursor option, it must be positive, because a negative value may cause unpredictable results.
- ERASE** Erase screen and place cursor in top left-hand corner of screen before output, unless the cursor is positioned specifically, either in the map definition or by the CURSOR option.

The first EXEC CICS SEND MAP command of any CICS application program should specify ERASE. This ensures that the size of the screen is set to default or alternate, according to the SCRNSZE parameter defined in the PCT for this transaction.
- ERASEAUP** Erase all unprotected fields before output.

FORMFEED	Send a form feed character as the first character in the device-dependent data stream.
FREEKB	Unlock the keyboard for data input.
FRSET	Reset all modified data tags (to “not modified” state) before output.
PRINT	Start printing (when terminal is a printer).

Cursor positioning

You can control the positioning of the display cursor in three different ways, as described below.

Normal cursor positioning

You can specify a 2-byte cursor position on the BMS EXEC CICS SEND commands. This enables you to specify the absolute value of the cursor position on the display screen after the EXEC CICS SEND command has been performed. Note that the first location on the display screen is address zero.

You specify the address on CURSOR option, as in the following example:

```
CURSOR(44)
```

Initial display position

If you omit the CURSOR option on an EXEC CICS SEND MAP command with the ERASE option, BMS searches the map for a field with the IC attribute. (You would have given it this attribute by coding ATTRB=IC on the DFHMDF macro for the field.) If there is more than one field with the IC attribute, BMS places the cursor at the beginning of the last one. If there is no such field, BMS places the cursor at screen address zero.

If you omit the CURSOR option from an EXEC CICS SEND CONTROL command, or from an EXEC CICS SEND MAP command without the ERASE option, cursor position remains unchanged.

Symbolic cursor positioning

You can use symbolic cursor positioning instead of coding an explicit value on the CURSOR option of the EXEC CICS SEND MAP command.

To do this:

1. Specify MODE=INOUT in the DFHMSD macro.
2. Set the length of the field (where the cursor is to be positioned) to -1.
3. Execute the EXEC CICS SEND command, specifying CURSOR without an argument.

CICS then places the cursor under the first data byte in the field on the output screen. If the length of more than one field is set to -1, the cursor is placed at the beginning of the first of those fields.

If you use symbolic cursor positioning with the EXEC CICS SEND CONTROL command, the cursor is always positioned at position zero of the panel.

Accessing data outside the program

Sometimes your program needs access to information held by CICS. The EXEC CICS ASSIGN command allows it such access.

Some EXEC CICS ASSIGN command options apply exclusively to BMS; for information about these options, see page 327. However, the only EXEC CICS ASSIGN command options you can use under BMS are those concerned with the position and size of maps. These are particularly useful with null maps. (Null maps have already been described under “Maps without fields” on page 148.) The EXEC CICS ASSIGN command options you can use are:

MAPLINE Requests the number of the line, on a display, that contains the origin of the most recently sent map.

MAPCOLUMN Requests the number of the column, on a display, that contains the origin of the most recently sent map.

MAPWIDTH Returns the width of the most recently sent map.

MAPHEIGHT Returns the height of the most recently sent map.

Receiving data from a display

You use the EXEC CICS RECEIVE MAP command to receive data from a display. The data from the display is mapped into a data area in an application program. This command is not sensitive to DBCS data and so works in the same way for DBCS data as for SBCS data.

For the full syntax of the EXEC CICS RECEIVE MAP command, see page 415. The MAP option names the map that is used to convert the data to its unformatted form, and the MAPSET option names the map set where the map belongs. The effect of omitting the MAPSET option is the same as explained for an EXEC CICS SEND MAP command on 155.

For example, in its simplest form the RECEIVE MAP command is coded as:

```
EXEC CICS RECEIVE MAP('DISPLAY')
```

This command tells BMS to map the input data into a symbolic map data structure called DISPLAY. The example assumes that the name of the map set is also DISPLAY.

Another map, MENU, in the same map set can be read by:

```
EXEC CICS RECEIVE MAP('MENU') MAPSET('DISPLAY')
```

This command tells BMS to map the input data into a symbolic map data structure called MENU.

After an EXEC CICS RECEIVE MAP command, your program can determine the inbound cursor position by inspecting the halfword binary value stored in EIBCPOSN. In doing this, the application program becomes dependent on the physical layout of the screen, although BMS separates the screen layout from the application for other interfaces.

Receiving data into an alternative data structure

The sample EXEC CICS RECEIVE MAP commands shown above use a literal for the name of the map or map set. You can also use a variable for these names, but you must use one of the INTO or SET options.

If you code the INTO option, display data is mapped into the named data area rather than into the data structure for the symbolic description.

If you code the SET option, BMS acquires a data area for you, maps the display data into it, and stores the address of the data area in the named pointer reference.

The rules for getting main storage for an input operation are the same as for output. For more information, see “Getting storage for a data structure” on page 156..

BMS sets the receiving area to nulls (X'00') before performing the receive operation, so you should save any data in this area before performing a receive operation. Furthermore, if you depend on BMS to set a data area to nulls for you during a receive operation, you should be aware that, if the MAPFAIL condition occurs, BMS does not set the input map to nulls.

If an operator types into a BMS input map, but does not fill one of the fields, BMS justifies the input data and pads the empty part of the field according to predefined rules. These depend on what you specify with the JUSTIFY operand of the DFHMDF macro. For general-usage programming interface information about the JUSTIFY operand, see “DFHMDF” on page 566.

The MAPFAIL condition can occur unexpectedly after an EXEC CICS RECEIVE MAP command. For example, it occurs if the terminal operator presses a PA key when CICS is waiting to perform an EXEC CICS RECEIVE MAP command. The PA1, PA2, PA3, or CLEAR keys will produce MAPFAIL conditions. PF keys or the enter key produce the MAPFAIL condition only if no user data was entered into a field, with MDT off, on the area of the screen defined for the received map. Therefore, you should always consider using the RESP option and inspecting the returned code, or coding an EXEC CICS HANDLE CONDITION command for the MAPFAIL condition.

Uppercase translation

By default, the data to be mapped is assumed to come from a terminal. The terminal control table entry for the terminal can specify that all input data is to be translated to uppercase by specifying UCTRN(*YES) in the TCT definition for the terminal. You can override this for any individual EXEC CICS RECEIVE command by specifying ASIS. Note, however, that ASIS has no effect on the first EXEC CICS RECEIVE MAP command of a transaction. (This means that ASIS is irrelevant to pseudoconversational transactions that issue only one EXEC CICS RECEIVE MAP command.)

Mapping data from another data area

Sometimes you need to perform an input mapping operation in two stages; accepting and storing the input data in one stage, and mapping it in the second. For example, your program might receive (but not map) data using a terminal control EXEC CICS RECEIVE command. It would then have to map the data from the storage area or pass the data to a subsequent program to map the data.

This might be done to avoid the overhead incurred by BMS mapping, if the received data does not need to be processed by the first program.

You use the FROM and LENGTH options of the EXEC CICS RECEIVE MAP command to specify that data is to be mapped from a data area instead of from a terminal. The FROM option names the data area; the LENGTH option indicates the number of bytes of data to be mapped. If the data is produced by a terminal

control EXEC CICS RECEIVE command, the LENGTH value of the EXEC CICS RECEIVE MAP command must match that returned by the original EXEC CICS RECEIVE command.

The LENGTH option is used when multiple maps in a map set have been sent to the display device. When an EXEC CICS RECEIVE INTO command is executed, this can only be done once against the input data stream. Otherwise, the next EXEC CICS RECEIVE INTO causes a read request to be issued to the device. A technique of the EXEC CICS RECEIVE MAP FROM command allows the programmer to issue multiple receive requests to determine on which map of the map set the user entered data.

Before issuing an EXEC CICS RECEIVE MAP command, the program should inspect EIBAID for the program function keys.

You cannot issue the EXEC CICS RECEIVE MAP command in a task not associated with a terminal, because BMS needs to refer to terminal information to analyze the data stream.

For general programming information about the terminal control EXEC CICS RECEIVE command, see page 412.

Note: The data obtained from an EXEC CICS RECEIVE BUFFER command cannot be mapped, because the data does not contain set buffer address (SBA) orders and a MAPFAIL condition is raised.

Responding to terminal input

Some operator actions cause an AID to be sent to CICS. Each such action generates a different AID. The AID is a single character value, which an application program can test by inspecting the contents of the EIBAID field and comparing it to the values supplied in the DFHAID copybook. This can be used as a mechanism for controlling program flow. The EXEC CICS HANDLE AID command controls conditional branching caused by AIDs. If any of the RESP, RESP2, and NOHANDLE options have been specified, the HANDLE AID function is suspended for that command.

Exception conditions

CICS exception conditions have already been introduced under Chapter 6, "Dealing with exception conditions," on page 87. Exception conditions can occur when you are using BMS commands. For general-usage programming interface information about the BMS commands, and the default system action they invoke, see Chapter 32, "Application programming commands - reference," on page 323.

An exception condition is not necessarily an error condition. Sometimes you might even want to treat an exception condition as part of the normal course of events.

If a RESP, RESP2, or NOHANDLE option is specified on the BMS command, then any HANDLE CONDITION function is suspended for that command.

The EIBAID field

A program can examine the value of the EIBAID field in the EIB to find out which attention key has been pressed. The 3270 and 5250 terminal transmits an AID character stored in field EIBAID. The program can compare the contents of EIBAID with the constants supplied in the CICS copybook DFHAID. Using EIBAID is

particularly suited to a structured programming environment. For general-usage programming interface information about DFHAID, see Appendix B, "BMS-related constants," on page 545.

The EXEC CICS HANDLE AID command

For COBOL/400 programs only, instead of examining the contents of EIBAID, you can use the EXEC CICS HANDLE AID command to pass control to a specified label when CICS receives an AID from a display device; control is passed after the input operation is completed. In the absence of an EXEC CICS HANDLE AID command for an AID, control returns to the application program at the point immediately following the input request.

You can suspend the EXEC CICS HANDLE AID command using the EXEC CICS PUSH HANDLE and EXEC CICS POP HANDLE commands as described in Chapter 6, "Dealing with exception conditions," on page 87. Note that the RESP option (invokes NOHANDLE) suspends the HANDLE AID function in the same way as it does with the HANDLE CONDITION function, and is better suited to a structured programming environment.

An EXEC CICS HANDLE AID command takes precedence over an EXEC CICS HANDLE CONDITION command, unless the exception condition stops receipt of the AID. If an AID is received during an input operation that an EXEC CICS HANDLE AID command is active for, control passes to the label specified in the EXEC CICS HANDLE AID command, regardless of any exception conditions that occur.

The EXEC CICS HANDLE AID command options that can be specified under BMS are:

ANYKEY

Any PF key, or the CLEAR key, but not ENTER; in addition, for a 3270 device, any PA key and, for a 5250 device, record backspace are included. See note 2 on page 143 for more information about record backspace

CLEAR

The key of that name

ENTER

The key of that name

PA1, PA2, or PA3 (3270), and record backspace (5250)

Any of the program access keys for a 3270 device, or record backspace for a 5250 device. See note 2 on page 143 for more information about record backspace.

PF1 through PF24

Any of the program function keys.

An EXEC CICS HANDLE AID command for a specified AID remains active until the task is terminated or until another EXEC CICS HANDLE AID command is issued for that AID. (If no label is specified in the new request, the existing EXEC CICS HANDLE AID command is suspended.)

A EXEC CICS HANDLE AID command is valid only for the program that issues it. Each new program in a task starts without any active HANDLE AID settings. When control returns to a program from a program at a lower logical level, the EXEC CICS HANDLE AID commands that were active in the higher-level program before control was transferred from it are reactivated, and any EXEC CICS HANDLE AID commands activated in the lower-level program are deactivated.

If an AID covered by the general option ANYKEY is received and there is no active EXEC CICS HANDLE AID command for the specified AID but there is an active EXEC CICS HANDLE AID ANYKEY command, control passes to the label specified in this command. A HANDLE AID command for an AID overrides the HANDLE AID ANYKEY command in relation to that AID.

The following example shows an EXEC CICS HANDLE AID command that specifies one label (LAB1) for the PA1 key AID, a second label (LAB2) for the PA2 and PA3 key AIDs, all of the PF key AIDs except PF10, and the CLEAR key AID:

```
EXEC CICS HANDLE AID
      PA1(LAB1)
      ANYKEY(LAB2)
      PF10
END-EXEC.
```

You cannot code more than 16 options in a single EXEC CICS HANDLE AID command.

Text processing

You use the EXEC CICS SEND TEXT command to send text to a terminal. You cannot send DBCS data to a 5250 device using this command nor are you recommended to use it to send DBCS data to 3270 devices. You should use the EXEC CICS SEND MAP command instead. However, you can use the EXEC CICS SEND TEXT command to send SBCS data to DBCS-capable devices. For information about the syntax of the EXEC CICS SEND TEXT command, see “SEND TEXT” on page 439.

The data area containing the text to be sent is specified in the FROM option. The LENGTH option specifies the length of data sent from this area. To help control the format of the display, the text may contain embedded new line characters (X'15') and embedded blanks (X'40'). Character attribute controls may be embedded for 3270 devices but will be removed for 5250 devices. Character attribute controls are discussed in “Character attribute control (3270 devices only)” on page 165.

DBCS data in the text must be preceded by an SA order sequence of X'2843F8'. If SBCS data follows the DBCS data, the DBCS data must be delimited with another SA order to reset the DBCS indication. This can be done either with the SA order sequence of X'284300' or X'280000'.

Display characters in text

When formatting the text, BMS splits it into lines of length less than or equal to the terminal page width. The terminal page width is assumed to be 80 bytes, unless the transaction uses the alternate screen size of 27x132, in which case the terminal page width is 132 bytes. Terminal screen width is specified using the ALTSCN parameter of the TCT resource definition. See the *CICS for iSeries Administration and Operations Guide* for further information about defining TCT entries.

BMS pads the ends of lines with blanks rather than splitting words. BMS starts each line with a single blank corresponding to the attribute byte. On a 3270 display device, the attribute byte is set to unprotected, autoskip, and normal intensity.

If a line of text ends with a character, however, and the next character is a blank, BMS processes the data as if it were a sentence, that is, the blank will be removed and the next character positioned in the first column of the next line.

Where a line of text ends with a blank and the next character is also a blank, BMS honors all blanks to process the data as if it were in table format.

When DBCS data is being formatted, BMS looks for the DBCS blank character (X'4040') not the SBCS blank (X'40').

BMS prevents the first byte of a DBCS character being placed in the last column position of a row. A single null value is placed in this position.

Control characters in text

BMS accepts either of two control characters in the text.

The newline character which has the SBCS value of X'15', or the DBCS value of X'0015', forces BMS to skip to the next line on the display. When used in a DBCS field, the newline character must be coded in its two-byte DBCS representation. BMS removes the newline character from the output data stream.

The null character which has the SBCS value of X'00', or the DBCS value of X'0000', is treated as a display character. Unlike the blank character, it is not used by BMS to determine the splitting of data between lines.

When used in a DBCS field, the null character must be coded in its two-byte DBCS representation. BMS retains null characters in the output data stream.

No other control characters are accepted by BMS. Their use would result in an abend with code ABMY.

If the FROM data area contains more text than fits on a single page, BMS creates more than one page. These overwrite each other on a display terminal.

The other options of the EXEC CICS SEND TEXT command have the same effect as the corresponding options of the EXEC CICS SEND MAP command, as described in "Sending data to a display device" on page 155.

Character attribute control (3270 devices only)

When data is destined for a device that supports the 3270 extended data stream, you can include SA (set attribute) orders in the data area specified in the FROM option. These orders enable you to apply extended attributes to characters or words in the data stream. The text may contain SA (Set Attribute) sequences for the following types:

- Reset All Attribute Types (type value of x'00')
- Extended Highlighting (type value of x'41')
- Foreground Color (type value of x'42')
- Character Set (type value of x'43')
- Transparency (type value of x'46')

You should refer to the *3270 Data Stream Programmer's Reference* manual for details of coding SA sequences. BMS will accept any of the above values as an attribute type value, but does not validate the setting of that attribute. For example, if you code the SA order X'2842F1', BMS recognizes the 28 as an SA order, and then

validates the 42 as the attribute type value for COLOR. Although in this example the F1 which follows is valid and represents BLUE, BMS accepts any value in this position. You are responsible for ensuring this value is correct.

Orders for extended attributes not supported by a terminal are removed from the data stream by BMS. If a sequence of orders is less than three characters long, or contains an invalid attribute type, the transaction is terminated abnormally with abend code ABMX.

Attributes remain effective until overridden by subsequent orders. Attributes are reset to their default values by a subsequent EXEC CICS SEND TEXT command.

As described in Appendix B, “BMS-related constants,” on page 545, copybook DFHBMSCA contains a selection of predefined constants that you can use in your programs.

Unsupported attributes

The Field Outlining attribute is supported only in the 3270 data stream architecture as a field based attribute, not a character attribute. Because the EXEC CICS SEND TEXT command supports only character attributes, field outlining cannot be used.

The Background Color attribute is supported in the 3270 data stream architecture as both a field-based and a character attribute. However, since CICS has never supported it in either capacity, it is regarded as invalid by the EXEC CICS SEND TEXT command.

The SOSI attribute is also supported in the 3270 data stream architecture as both a field-based and a character attribute. However, CICS supports SOSI only as a field-based attribute and it is therefore regarded as invalid by the EXEC CICS SEND TEXT command.

Graphic data fields

Graphic data fields are fields that contain only DBCS characters. These fields are also referred to as “DBCS-only” fields, and in the mainframe environment they are referred to as “pure DBCS” fields.

In the 3270 architecture, data in graphic fields is not enclosed by SO and SI characters. The data is defined as graphic by using the PS (Programmed Symbols) extended attribute with a type value of X'F8'.

However, in the 5250 architecture, unless the new *ideographic* fields support of the 5494 controller is in effect, graphic data fields must be enclosed in SO and SI characters.

This usually means that the iSeries data conversion routines remove one DBCS character from graphic fields to make space for an SO and an SI character. The rightmost character is removed from the fields.

If a graphic field wraps around from the bottom line of the display to the top line, several DBCS characters may be removed from the field. Although BMS prevents fields wrapping in this way, this might occur as a result of a terminal control EXEC CICS SEND command when the application is building the data stream.

The loss of the single character in each field could be overcome by placing a DBCS blank character at the rightmost position in each graphic field.

Printed output

Very often you may want printed output (hardcopy) in addition, or instead of, the screen images produced by a transaction. You can use the hardware print key to print the contents of your display screen, or you can use the asynchronous page build transaction of CICS to produce hardcopy on a printer.

Using the hardware print key

Some display terminal models have a hardware print key. Pressing this key initiates a print process involving only the terminal, its controller, and a printer attached to the controller. This allows you to print the contents of your display screen. Neither the host processor, nor CICS, nor your application program can control this process.

Using asynchronous page build transaction

The method of printing described so far has the advantage of being simple to implement. However, it does not provide the flexibility often needed in commercial applications. In particular, it does not allow your program to combine mapped output data to produce an entire printed page.

Before looking at how this might be done, first consider how a program uses a printer.

For an SCS printer, the print data is formatted into an OS/400 object known as a print file, which is associated with a particular printer. This print file is named in the TCT entry for the printer. Any type of EXEC CICS SEND command issued by a program running against this TCT entry causes CICS to format data into a print file.

For BMS commands, CICS emulates 3270 printer buffering. Data is formatted into the print file when either the PRINT option is specified on an EXEC CICS SEND MAP, EXEC CICS SEND TEXT, or EXEC CICS SEND CONTROL command, or the PRINT option is defined on the DFHMSD or DFHMDI macro for the map. The data is erased only if BMS receives an EXEC CICS SEND MAP or EXEC CICS SEND CONTROL command that specifies the ERASE option.

Two ways of printing a page built from multiple maps are:

1. Using the interval control EXEC CICS START command.

You use the EXEC CICS START command to initiate a secondary CICS task. This is a print task if the TERMID option of the command names a printer as its **principal facility**. Your initial transaction can pass data to the print task by specifying the FROM and LENGTH options of the EXEC CICS START command. If the primary transaction has already created a series of output data structures in the FROM area, the secondary transaction can map the data into the printer buffer, then initiate printing using a BMS EXEC CICS SEND command with the PRINT option.

2. Using a transient data queue with a trigger level.

You can send symbolic map data structures to a transient data queue using the EXEC CICS WRITEQ command. CICS can be made to initiate a print transaction when a specific number of records have been written to the queue. The name of the transaction to be initiated, the identifier of the printer that is to be its principal facility, and the trigger level it is started at, are defined in the destination control table (DCT).

Note, however, that output from several instances of your transaction may be interleaved on the transient data queue. This can be avoided if all the data to be printed by an instance of your transaction is stored in a single transient data queue item, or if the transient data queue record contains the name of an alternative resource (for example, a temporary storage queue) on which the records to be printed are stored. Alternatively, each instance of your transaction can get exclusive control of the transient data queue by using the EXEC CICS ENQ and EXEC CICS DEQ commands.

Printer formatting considerations

You should note the following when sending data to a printer.

Blank lines and 3270 printers

Every line in a map for a 3270 printer must contain field data (blanks if necessary), because the 3270 does not print empty lines (that is, lines of null characters).

Setting the printer page width

The 3270 printer prints data using a line width specified in the write control character (WCC). This line width must be set to 40, 64, or 80 columns, or the printer platen width. The WCC line width is set by BMS from the TCT page width. Unexpected results occur if the TCT page width is not 40, 64, or 80 columns, or is not the printer platen width. For printers, the line width is one byte less than the width of the print file associated with the TCT resource definition for the printer.

Form feed characters

You can code an option called FORMFEED on the EXEC CICS SEND MAP and EXEC CICS SEND CONTROL commands. This generates a form feed character (X'0C') at the start of the data stream. If you code this option for a terminal that doesn't support form feed, CICS simply ignores the request.

The form feed character occupies screen position 1 (the top left-hand corner) on a 3270 display or printer. It can be overwritten by other data sent to the terminal, but then the form feed does not occur.

The FORMFEED option on 3270 displays is particularly useful if the screen is to be printed using the hardware local-copy key. Its use ensures that the screen image is printed on a new page.

Be careful when using the FORMFEED option on an EXEC CICS SEND CONTROL command. The EXEC CICS SEND CONTROL command always generates a complete blank page. Thus an EXEC CICS SEND CONTROL FORMFEED skips to a new page and also sends this as a blank page. However, as described earlier, 3270 printers sometimes suppress null lines so that a blank page is printed as a single line.

Chapter 14. Terminal control

This chapter introduces the CICS terminal control program, which allows user-written application programs and terminals and logical units to communicate using terminal control commands.

Terminal control is used to control communication with logical units. A logical unit can represent a terminal device, such as a display or printer device, or it can represent another system (for example, another iSeries).

CICS/400 terminal control commands support the following types of SNA logical units:

- **LU 1**—communication with application programs or devices. In CICS/400, this LU is for 3270-type SNA character string (SCS) printers.
- **LU 2**—communication with 3270-type display devices.
- **LU 3**—communication with 3270-type data stream printers.
- **LU 6.2**—Advanced Program-to-Program Communication (APPC) for CICS-to-CICS sessions and communication with other systems such as the System/36, and other communication interfaces such as CPI-C and APPC on any platform, and OS/400 ICF.

Devices of the 5250-type are supported through data stream translation. For an explanation of this process, see “OS/400 display data streams” on page 174.

Terminal control uses the standard OS/400 data management function to communicate with logical units. It uses the following types of device files:

- **Printer file (*PRTF)**—used for LU type 1 SCS printers.
- **Display file (*DSPF)**—used for LU type 2.
- **Intersystem communication function file (*ICFF)**—used for LU type 6.2, APPC.

Note: For an explanation of the naming convention used above, see “Conventions and terminology used in this book” on page xiii.

Terminal control handles data translation, synchronization of input and output operations, and the session control needed to read from or write to a terminal. This frees the application from controlling terminals.

You can use terminal control to communicate with a remote system by means of distributed transaction processing (DTP), which is described in the *CICS for iSeries Intercommunication* book.

Commands and options that apply specifically to logical units are described in “Logical unit communication protocol” on page 170.

You can use the following terminal control commands (provided they apply to your terminal or logical unit):

EXEC CICS RECEIVE	Read data from a terminal or logical unit
EXEC CICS SEND	Write data to a terminal or logical unit
EXEC CICS CONVERSE	Converse with a terminal or logical unit

EXEC CICS ISSUE SIGNAL Send an asynchronous interrupt

When using EXEC CICS RECEIVE to receive data from a terminal, you should be aware that the data stream returned may contain SBA data if the screen was in a formatted state or was a 5250. In this case, you should consider using an EXEC CICS RECEIVE MAP command instead.

Terminal-oriented task identification

When CICS receives input from a logical unit to which no task is attached, it has to determine which transaction is to be initiated. The CICS/400 application shell performs this transaction identification and task attachment. The methods by which the user can specify the transaction to be initiated and the sequence in which CICS tests these specifications are covered in the STRCICSUSR CL command in the *CICS for iSeries Administration and Operations Guide*.

When the task is attached, the application program for the transaction is connected to the logical unit for the duration of the task. The application shell directs terminal control to make this connection. The application program then communicates with the logical unit through terminal control commands.

Logical unit communication protocol

An application program communicates with an SNA logical unit by using terminal control commands. However, communication with logical units is governed by the conventions (protocols) that apply to each type of logical unit. This section describes the additional commands and options provided by CICS to enable application programs to comply with these protocols.

The following table summarizes the EXEC CICS command options that CICS/400 provides for these protocols. It shows the CICS/400 support for these options by logical unit type.

Table 8. CICS/400 LU protocol options

Option	LU 1	LU 2	LU 3	LU 6.2
INVITE	Passed	Yes	Yes	Yes
CNOTCOMPL	Passed	N/A	N/A	N/A
DEFRESP	Passed	Passed	Passed	N/A
LAST	Passed	Passed	Passed	Yes

Note: **Yes**—Option is supported. **Passed**—Option is ignored for logical units owned by the local CICS/400 system, but is passed to remote systems during transaction routing. **N/A**—Option is not applicable to this logical unit.

Send/receive mode

For SNA logical units, only one of the two ends of the session can be in send mode at any one time, that is, one is in send mode, the other is in receive mode. An application program in send mode can issue any commands for the logical unit. On the other hand, one in receive mode, can issue only receive requests until the mode is changed back to send. The EIB indicator EIBRECV informs the application program that it is in receive mode and that it must perform the above operations.

For displays, the transaction would normally be in send mode, if the INVITE option is not used, and can ignore the EIBRECV indicator. Displays work with a subset of the full protocols.

Send/receive protocol (INVITE option)

The INVITE option of an EXEC CICS SEND command informs the session partner that it is now in send mode and that it should send a reply. At the same time it places the transaction in receive mode. The transaction should now issue an EXEC CICS RECEIVE command as its next operation.

Chaining the input data

You can present the data to the program in assembled chains. The unit of data from a logical unit is the request unit (RU). One or more RUs can be grouped together and treated as a chain. Chain assembly is done by OS/400, depending on logical unit type. When chain assembly is done, instead of an input request being satisfied by one RU at a time until the chain is complete, the whole chain is assembled and is sent to the CICS application program satisfying just one request. This ensures that the integrity of the whole chain is known before it is presented to the application program.

The last RU in a chain (even if it is the only RU in the chain) raises an end-of-chain (EOC) condition. For logical units that do not send chained data (for example, the 3270 logical unit), the EOC condition occurs for every receive request. For logical units that send chained data, the EOC condition usually occurs for every receive request, but it may not, depending on the length of the data. By using RESP options, you can detect the EOC condition and pass control to the next instruction in the program, where a decision can be made on subsequent processing. See “How to use the RESP and RESP2 options” on page 87 for further information about using RESP.

An EXEC CICS HANDLE CONDITION EOC command gives control to a user-written routine, which can do any additional processing required when the complete chain has been received. You can use the EXEC CICS IGNORE CONDITION command to ignore the EOC condition in cases where it is raised on every EXEC CICS RECEIVE command.

Chaining the output data

As in the case of input data, output data is sent as request units (RUs). If the length of the data to be sent exceeds the RU size, the data is broken into several RUs and they are sent as a chain. During transmission from CICS to the logical unit, the RUs are marked FIC (first-in-chain), MIC (middle-in-chain), or EOC (end-of-chain) to denote their position in the chain. An RU that is the only one in a chain is marked OIC (only-in-chain).

For some logical unit types, the application program can control the chaining of outbound data. You can chain the outbound data by including the CNOTCOMPL (chain-not-complete) option on EXEC CICS SEND commands. This indicates the continuation of the chain. Excluding the CNOTCOMPL option on the last RU completes the chain. In general, the CNOTCOMPL option should not be used. Once an output request with CNOTCOMPL specified has been made, subsequent output requests may not use the LAST option until the beginning of the next chain (that is, the first output request following an output request in which CNOTCOMPL is omitted).

Response protocol

The two types of response that CICS can request for outbound data for an application program are:

- Definite response
- Exception response

“Definite response” means a response, even if an error does not occur. The “exception response” means a response only if an error occurs.

If exception response protocol is used, an exception response may not be received and handled immediately after it arises.

CICS/400 does not allow specification of which response is required; OS/400 controls the type of response used. Remote terminal-owning systems, used with transaction routing, may allow the system programmer to specify which response type is required.

The use of definite response protocol has some performance disadvantages, but may be necessary for some application programs. The DEFRESP option on the EXEC CICS SEND command provides a more flexible method of specifying the protocol to be used. One example of the use of this option is to request a definite response for every tenth output command, with exception response being the general rule.

Because a definite response can be requested only on the last element in the chain, the DEFRESP and CNOTCOMPL options are mutually exclusive.

Preventing interruptions (bracket protocol)

Bracket protocol prevents the interruption of a transaction between CICS and a logical unit. Generally, a bracket can be initiated by CICS or by the logical unit, and ended only by CICS unless it is for an APPC (LU6.2) logical unit, in which case the logical unit can end it. A bracket can also delimit conversation between CICS and the logical unit or merely the transmission of a series of data chains in one direction.

Bracket protocol is used when CICS communicates with specific logical units. The use of brackets is not usually apparent to the application program.

Only on the last output request of a task to a logical unit does the bracket protocol become apparent to the application program. On the last output request to a logical unit, the application program may specify the LAST option on the EXEC CICS SEND command. The last output request is defined as either the last EXEC CICS SEND command specified for a task without chain control; or as the output request that transmits the FIC or OIC marker of the last chain of a transaction with chain control.

The LAST option causes CICS to transmit an end-bracket indicator with the final output message to the logical unit. This indicator notifies the logical unit that the current transaction is ending. If the LAST option is not specified, CICS waits until the task detaches before sending the end-bracket indicator. Because an end-bracket indicator is transmitted only with the first RU of a chain, the LAST option is ignored for a transaction with chain control unless FIC or OIC is also specified.

Including an EXEC CICS FREE command after an EXEC CICS SEND command with the LAST (LU6.2 only) option may be useful if the transaction does not terminate immediately after issuing the EXEC CICS SEND command. This allows another transaction to be initiated from the LU or from CICS.

Handling attention identifiers (EXEC CICS HANDLE AID)

Note: This information applies to COBOL programs only.

The RESP, RESP2, and NOHANDLE options on EXEC CICS commands suspend the use of the EXEC CICS HANDLE AID command. In the absence of an EXEC CICS HANDLE AID command, control returns to the application program at the point immediately following the input command. You can suspend the EXEC CICS HANDLE AID command using the EXEC CICS PUSH HANDLE and EXEC CICS POP HANDLE commands.

An EXEC CICS HANDLE AID command takes precedence over an EXEC CICS HANDLE CONDITION command. If an AID is received during an input operation for which an EXEC CICS HANDLE AID command is active, control passes to the label specified in that command regardless of any conditions that may have occurred (but which did not stop receipt of the AID).

The EXEC CICS HANDLE AID command for a given AID applies only to the program in which it is specified, remaining active until the program is ended, or until another EXEC CICS HANDLE AID command for the same AID is met, in which case the new command overrides the previous one.

When control returns to a program from a program at a lower logical level, the EXEC CICS HANDLE AID commands that were active in the higher-level program before control was transferred from it are reactivated, and those in the lower-level program are deactivated.

If no EXEC CICS HANDLE AID command is active for any PA key, PF key, or the CLEAR key, but one is active for ANYKEY, control is passed to the label specified for ANYKEY. An EXEC CICS HANDLE AID command for an AID overrides the EXEC CICS HANDLE AID ANYKEY command for that AID.

If a task is initiated from a terminal by use of an AID, the first EXEC CICS RECEIVE command in the task does not read from the terminal but only copies the input buffer (even if the length of the data is zero) so that control may be passed by means of an EXEC CICS HANDLE AID command for that AID.

An EXEC CICS RECEIVE MAP command with the FROM option does not cause an EXEC CICS HANDLE AID command to be invoked because no terminal input is involved. For information about this command, see page 371.

Note: If you use the NOHANDLE option (or RESP or RESP2 options, which invoke NOHANDLE), it suspends the EXEC CICS HANDLE AID function. If you want to change the program processing, depending on the attention key pressed, compare the contents of EIBAID with the fields in the standard attention identifier list (DFHAID), and then transfer control to the routine needed to perform the function you want.

OS/400 display data streams

Terminal control handles data stream translation for display devices and printers. This enables CICS application programs, using only 3270 data streams, to work with all types of display devices and printers attached to the OS/400:

- ASCII display devices and printers
- 3270-type display devices and printers
- 5250-type display devices and printers

Display devices and printers attached to OS/400 are normally seen as 5250-type devices by programs running on OS/400. ASCII and 3270-type display devices and printers must have their data streams translated to and from 5250 data streams. OS/400 does this translation. It also maps the ASCII and 3270 keyboards into logical 5250 keyboards.

CICS/400 application programs, using terminal control commands, see display devices and printers as 3270-type devices (LUTYPE2/LUTYPE3). Application programs send and receive 3270 data streams with terminal control commands. Terminal control translates these data streams into the 5250 data streams that OS/400 uses.

CICS uses the OS/400 supplied data stream translation functions. OS/400 3270 Device Emulation uses these functions also. This gives CICS/400 and OS/400 3270 DE the same keyboard mapping. See the *3270 Device Emulation Support* manual for information about keyboard differences.

Terminal control and DBCS

The terminal control API commands EXEC CICS SEND and EXEC CICS RECEIVE can be used against DBCS capable display devices. However, when using these commands, the application is responsible for building and interpreting the device data streams.

Data streams must be in 3270 data stream format, even when the device associated with the transaction is a 5250. In other words, for an EXEC CICS SEND command the application must build a 3270 data stream, and for an EXEC CICS RECEIVE command the application must interpret a 3270 data stream, whether the actual device is a 3270 or a 5250.

When the device is a 5250, the data stream is translated and there may be some loss of data in graphic fields. You are advised to avoid the use of character attributes when sending data to 5250 devices. Refer to the *3270 Device Emulation Support* manual for more information about DBCS data stream translation for 3270 emulation.

Chapter 15. Intercommunication considerations

This chapter provides only a summary of what you need to consider when you write applications that communicate with other CICS systems. For further information, see the *CICS Family: Interproduct Communication* book and the *CICS for iSeries Intercommunication* book.

You can run application programs in a CICS intercommunication environment using one or more of the following:

- **Transaction routing**—enables a terminal in one CICS system to run a transaction in another CICS system.
- **Function shipping**—enables your application program to access resources in another CICS system.
- **Distributed program link (DPL)**—enables an application program running in one CICS system to link to another application program running in a remote CICS system.
- **Asynchronous processing**—enables a CICS transaction to start another transaction in a remote system and optionally pass data to it.
- **Distributed transaction processing (DTP)**—enables a CICS transaction to communicate with a transaction running in another system.

There is one interface available for DTP: command-level EXEC CICS. The Systems Application Architecture* (SAA) interface for DTP called CPI Communications (Common Programming Interface Communications) is not directly supported by CICS/400.

Design considerations

If your application program uses more than one of these facilities, you obviously need to bear in mind the design considerations for each one. Also, if your program uses more than one intersystem session for distributed transaction processing, it must control each session according to the rules for that type of session.

Transaction routing

Transactions that can be invoked from a terminal owned by another CICS system, or that can acquire a terminal owned by another CICS system during transaction initiation, must be able to run in a transaction routing environment.

Generally, you can design and code such a transaction just like one used in a local environment. However, there are a few restrictions related to basic mapping support (BMS), pseudoconversational transactions, and the terminal on which your transaction is to run. All programs, tables, and maps that are used by a transaction *must* reside on the system that owns the transaction. (You can duplicate them in as many systems as you need.)

Some CICS transactions are related to one another, for example, through common access to the CWA or through shared storage acquired using an EXEC CICS GETMAIN command. When this is true, the system programmer must ensure that these transactions are routed to the same CICS system.

When a request to process a transaction is transmitted from one CICS system to another, transaction identifiers can be translated from local names to remote names. However, a transaction identifier specified in an EXEC CICS RETURN command is not translated when it is transmitted from the transaction-owning system to the terminal-owning system.

Function shipping

You code a program to access resources in a remote system in much the same way as if they were on the local system. You can use:

- **File control** commands to access files on remote systems
- **Temporary storage** commands to access data from temporary storage queues on remote systems
- **Transient data** commands to access transient data queues on remote systems

Three additional exception conditions can occur with remote resources. They occur if the remote system is not available (SYSIDERR), if a request is invalid (ISCINVREQ), or if the mirror transaction ² abends (ATNI for ISC connections).

Distributed program link (DPL)

The distributed program link function in CICS/400 adds to the distributed transaction functions available in CICS by enabling a CICS program (the client program) to call another CICS program (the server program) in a remote CICS system. There are several reasons why you might want to design your application to use distributed program link. Some of these are:

- To separate the end-user interface (for example, BMS screen handling) from the application business logic, such as accessing and processing data, to enable parts of the applications to be ported from host to workstation more readily
- To obtain performance benefits from running programs closer to the resources they access, and thus reduce the need for repeated function shipping requests
- To offer a simple alternative, in many cases, to writing distributed transaction processing (DTP) applications

You can specify that the program to which an application is linking is remote by specifying the remote system name either on the EXEC CICS LINK command or by using the SYSID parameter of the ADDCICSPPT or CHGCICSPPT CL commands.

The basic flow in distributed program link is described in the *CICS Family: Interproduct Communication* book. The following terms, illustrated in Figure 33 on page 177, are used in the discussion of distributed program link:

Client system The CICS system running an application program that issues a link to a program in another CICS system

Server system The CICS system to which a client system ships a link request

Client program

The application program that issues a remote link request

2. A CICS-supplied transaction that recreates a request that is function shipped from one system to another, issues the request on the second system, and passes the acquired data back to the first system.

Server program

The application program specified on the link request, and which is executed in the server system

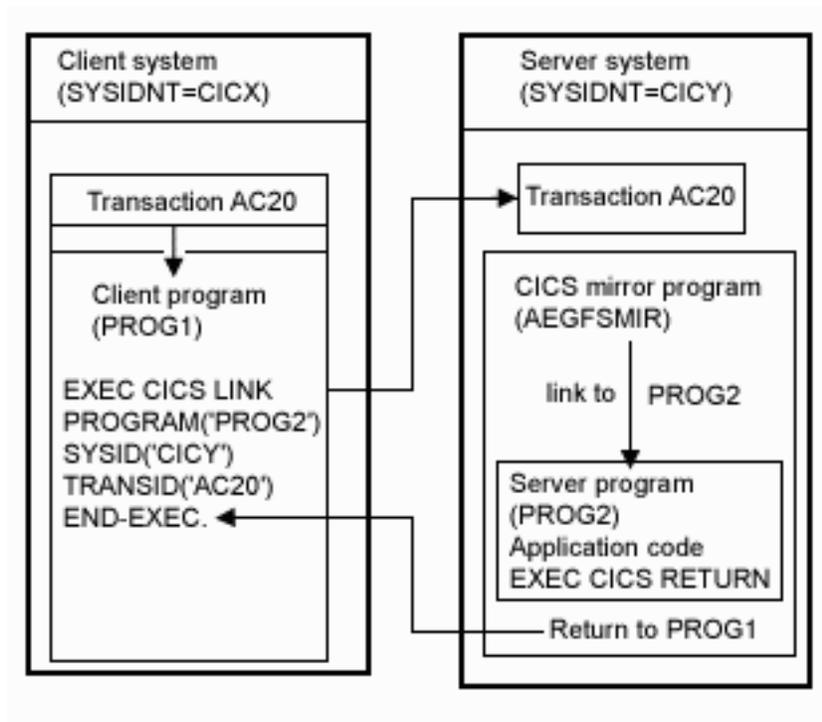


Figure 33. Illustration of distributed program link

Using the distributed program link function

The distributed program link function provides a number of options. You can specify:

- The name of the remote system (the server system).
- The name of the server program, if it is known by a different name in the server system.
- That you want to run the linked program locally, but restrict it to the distributed program link subset of the application programming interface (API) for testing purposes. (Server programs cannot use the entire CICS API when executed remotely; the restrictions are listed in Figure 38 on page 185).
- That the server system will take a syncpoint independently from the client.
- The name of the transaction you want the program to run under in the server system.
- The data length of the COMMAREA being passed.

A server program can itself issue a distributed program link and act as a client program with respect to the program it links to.

The options shown in Table 9 on page 178, and the parameters shown in Table 10 on page 178, are used on the EXEC CICS LINK command and on the PPT respectively, in support of the distributed program link facility.

Note: The full syntax of the EXEC CICS LINK command is given page 380.

Table 9. Options on EXEC CICS LINK command supporting DPL

Option	Description
DATALENGTH	Specifies the length of the contiguous area of storage (from the start of the COMMAREA) that the application is sending to a server program.
SYSID	Specifies the name of the connection to the server system to which you want the client system to ship the program link request. Note: SYSID specified on the EXEC CICS LINK command overrides the remote SYSID specified in the PPT entry.
SYNCONRETURN	Specifies that you want the server system to take a syncpoint on successful completion of the server program. Note: This option is unique to the EXEC CICS LINK command and cannot be specified on the PPT.
TRANSID	Specifies the name of the transaction that the server system is to attach for execution of the server program. Note: TRANSID specified on the EXEC CICS LINK command overrides any remote TRANSID specified on the PPT.

Table 10. ADDCICSPPT and CHGCICSPPT CL command parameters supporting DPL

Keyword	Description
SYSID	Specifies the name of the connection to the server system (SYSID) to which you want the client system to ship the program link request.
RMTPGMID	Specifies the name by which the program is known in the server system (if different from the local name).
APISET	Specifies whether the program is restricted to the distributed program link subset of the CICS API. Note: This option is unique to the PPT and cannot be specified on the EXEC CICS LINK command.
TRANSID	Specifies the name of the transaction that the server system is to attach for execution of the server program.

Examples of distributed program link

A COBOL example of a distributed program link command is shown in Figure 34. The numbers down the right-hand side of the example refer to the numbered sections, following the figure, which give information about each option.

EXEC CICS LINK PROGRAM('DPLPROG')	1
COMMAREA(DPLPROG-DATA-AREA)	2
LENGTH(24000)	2
DATALENGTH(100)	2
SYSID('CICR')	3
TRANSID('AC20')	4
SYNCONRETURN	5
END-EXEC.	

Figure 34. COBOL example of a distributed program link

1 The program name of the server program

A program may have different names in the client and server systems. The

name you specify on the EXEC CICS LINK command depends on whether or not you specify the SYSID option, and also on whether the PPT entry in the client system specifies the RMTPGMID parameter.

If the program link command specifies a remote system on the SYSID option, CICS ships the link request to the server system without reference to the PPT entry in the client system. In this case, the program name specified on the link command must be the name by which the program is known in the server system.

If you do not specify the SYSID option on an EXEC CICS LINK command, however, the local name of the program is used. CICS looks up the PPT entry in the local (client) system. If the program is remote, CICS ships the request to the system specified in the SYSID parameter using the name by which the program is known in the server system. This is the RMTPGMID value if one was specified; otherwise the local name is used.

If the system name (SYSID) on the EXEC CICS LINK command is the same name as the client system, CICS initially processes the request as though it is a local link. CICS checks the local PPT entry and, if it specifies a remote system name, the client system ships the request to the system specified in the SYSID parameter. In this case, the remote system name on the PPT entry overrides the SYSID name on the program link command. If SYSID is not specified in the PPT entry, or it names the local system name, CICS invokes the program in the client system (a local link).

2 The communication data area (COMMAREA)

To improve performance, you can specify the DATALENGTH option on the EXEC CICS LINK command. This allows you to specify the amount of COMMAREA data you want the client system to pass to the server program. Typically, you use this option when a large COMMAREA is required to hold data that the server program is to return to the client program, but only a small amount of data needs to be sent to the server program by the client program, as in the example.

3 The remote system ID (SYSID)

The SYSID option on the EXEC CICS LINK command enables you to specify a 4-character name for the server system to which you want the client system to ship a program link request. This is the name of the terminal control system table (TCS[®]) entry installed in the client system defining the connection with the server system. (CICS uses the connection name in a table look-up to obtain the netname of the server system.) The name of the server system you specify on the SYSID parameter can be the name of the client system in which case the program is run locally.

If the server system is unable to load or run the requested program (DPLPROG in our example), CICS returns the PGMIDERR condition to the client program in response to the link request. Note that EIBRESP2 values are not returned over the link for a distributed program link request where the error is detected in the server system. For errors detected in the client system, EIBRESP2 values *are* returned.

4 The remote transaction (TRANSID) to be attached

The TRANSID option is available on both the EXEC CICS LINK command and also on the ADDCICSPPT and the CHGCICSPPT CL commands. This enables you to tell the server system what transaction identifier to use when it attaches the mirror task under which the server program runs. If you specify the TRANSID option, you must define the transaction in the server system, and associate it with the supplied mirror program AEGFSMIR. This option allows

you to specify your own attributes on the transaction definition for the purpose of performance and fine tuning. For example, you could vary the task priority and transaction class attributes.

You are recommended to specify the transaction identifier of the client program as the transaction identifier for the server program. This enables any statistics and monitoring data you collect to be correlated correctly under the same transaction.

The transaction identifier used on a distributed program link request is passed to the server program as follows:

- If you specify your own transaction identifier for the distributed program link request, this is passed to the server program in the EIBTRNID field of the EIB.
- If you do not specify a transaction identifier for the DPL request, CICS copies the transaction identifier under which the client program is running, into the EIBTRNID field of the server program.

5 The synchronization option for the server program

When you specify the SYNCONRETURN option, it means that the resources on the server are committed in a separate logical unit of work immediately before returning control to the client; that is, an implicit syncpoint is issued for the server just before the server returns control to the client. Figure 35 on page 181 shows an example of using distributed program link with the SYNCONRETURN option. The SYNCONRETURN option is intended for use when the client program is not updating any recoverable resources, for example, when performing screen handling. However, if the client does have recoverable resources, they are not committed at this point. They will be committed when the client itself reaches a syncpoint or in the implicit syncpoint at client task end. You must ensure that the client and server programs are designed correctly for this purpose, and that you are not risking data integrity. For example, if your client program has shipped data to the server that results in the server updating a database owned by the server system, you only specify an independent syncpoint if it is safe to do so, and when there is no dependency on what happens in the client program. This option has no effect if the server program runs locally in the client system unless APISET(*DPLSUBSET) is specified. In this case, the syncpoint rules governing a local link apply.

Without the SYNCONRETURN option, the client commits the logical unit of work for both the client and the server resources, with either explicit commands or the implicit syncpoint at task end. Thus, in this case, the server resources are committed at the same time as the client resources are committed. Figure 36 on page 182 shows an example of using distributed program link without the SYNCONRETURN option.

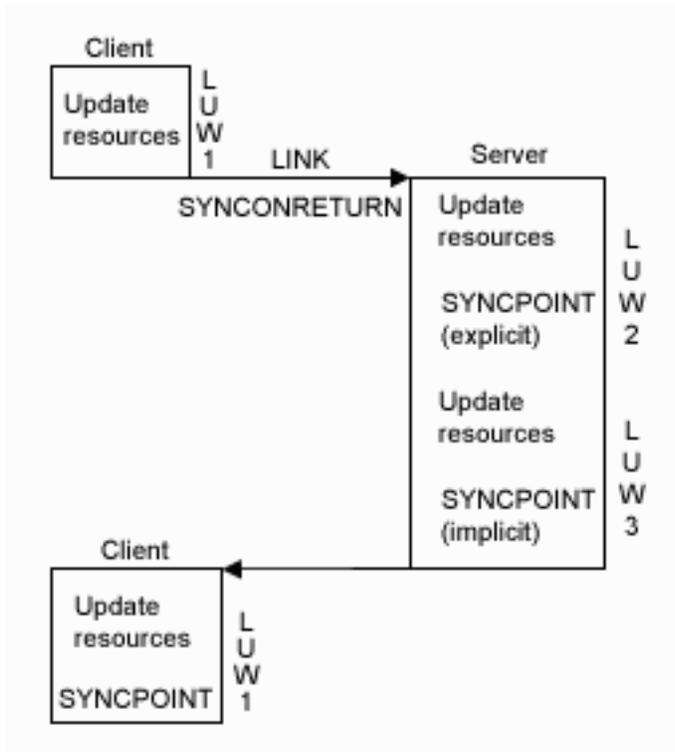


Figure 35. Using distributed program link with the SYNCONRETURN option

Note: This includes three logical units of work: one for the client and two for the server. The client resources are committed separately from the server.

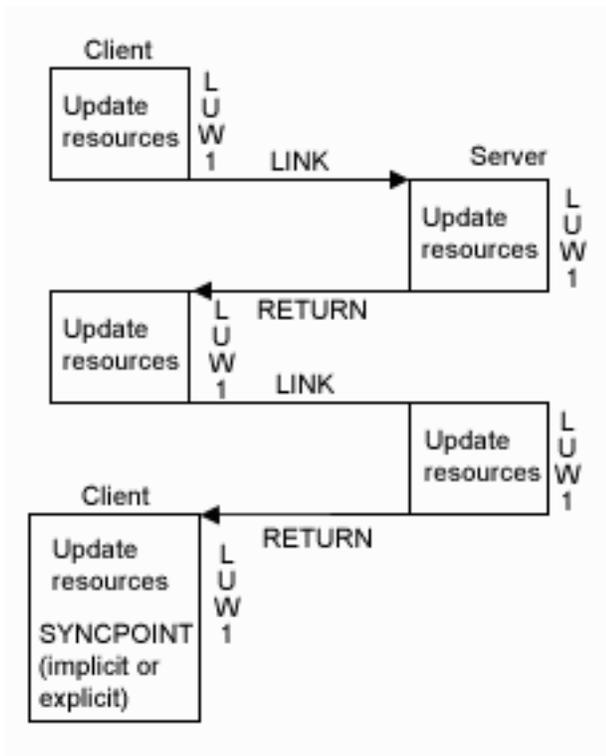


Figure 36. Using distributed program link without the SYNCONRETURN option

Note: The implicit or explicit syncpoint causes all client and server resources to be committed. There is only one logical unit of work because the client is responsible for determining when both the client and server resources are committed.

You need to consider the case when the client has an EXEC CICS HANDLE ABEND command. When the client is handling abends in the server, then the client gets control when the server abends. This is also true when the SYNCONRETURN option has been specified on the EXEC CICS LINK command. In this case, it is recommended that the client issue an abend after doing the minimum of cleanup. This causes both the client logical unit of work and the server logical unit of work to be backed out.

Programming considerations for distributed program link

There are some factors you should consider when writing application programs that use distributed program link.

Issuing multiple distributed program links from the same client task

A client task cannot request distributed program links to a single CICS server system using more than one transaction code in a single client unit of work unless the SYNCONRETURN option is specified. It can issue multiple distributed program links to one CICS server system with the same or the default transaction code.

Sharing resources between the client program and server program

The server program does not have access to the lifetime storage of tasks on the client, for example, the TWA. Nor does it necessarily have access to the resources that the client program is using, for example, files, unless the file requests are being function shipped.

Mixing DPL and function shipping to the same CICS system

Great care should be taken when mixing function shipping and DPL to the same CICS system, from the same client task. These are some considerations:

- A client task cannot function ship requests and then use distributed program link with the SYNCONRETURN option in the same client logical unit of work. The distributed program link would fail with an INVREQ response. In this case EIBRESP2 is set to 14.
- A client task cannot function ship requests and then use distributed program link with the TRANSID option in the same client logical unit of work. The distributed program link would fail with an INVREQ response. In this case, EIBRESP2 is set to 15.
- Any function-shipped requests that follow a DPL request with the SYNCONRETURN option run in a separate logical unit of work from the server logical unit of work.
- Any function-shipped requests running that follow a DPL request with the TRANSID option to the same server system will run under the transaction code specified on the TRANSID option, instead of under the default mirror transaction code. The function-shipped requests will be committed as part of the overall client logical unit of work when the client commits.
- Any function-shipped requests running before or after a DPL request without the SYNCONRETURN or TRANSID options are committed as part of the overall client logical unit of work when the client commits.

See the book, the *CICS Family: API Structure* book and the *CICS for iSeries Intercommunication* book for more information about function shipping.

Mixing DPL and DTP to the same CICS system

Care should be taken when using both DPL and DTP in the same application, particularly using DTP in the server program. For example, if you have not used the SYNCONRETURN option, you must avoid taking a syncpoint in the DTP partner which requires the DPL server program to syncpoint.

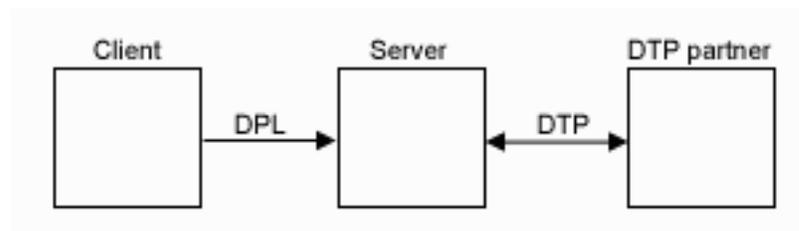


Figure 37. Example of mixing DPL and DTP

Restricting a program to the distributed program link subset

Note: For a description of the naming convention used in this section, see “Conventions and terminology used in this book” on page xiii.

When a program executes as the result of a distributed program link, it is restricted to a subset of the full CICS API called the distributed program link subset. The commands that are prohibited in a server program are summarized in Figure 38 on page 185.

You can specify, in the program resource definition only, that you want to restrict a program invoked by a local EXEC CICS LINK command to this subset with the APISET(*DPLSUBSET) option. The use of any prohibited commands can then be detected before an application program is used in a distributed environment. The APISET(*DPLSUBSET) option should be used for early testing purposes only, and should never be used in production.

When the server program is running locally the following considerations apply:

- If APISET(*DPLSUBSET) is specified on the server program then the SYNCONRETURN option causes an implicit syncpoint to be taken in the local server program, prior to returning control to the client program. In this case, because the server program is running locally, both the client and server resources are committed. However, you should note that SYNCONRETURN is intended for use when the client has no recoverable resources.
- If APISET(*FULLAPI) is specified on the server program then the SYNCONRETURN option is ignored.
- The TRANSID and DATALENGTH options are ignored when processing the local link, but the format of the arguments is checked, for example, the TRANSID argument cannot be all blank.

Determining how a program was invoked

The 2-byte values returned on the STARTCODE option of the EXEC CICS ASSIGN command are extended in support of the distributed program link function enabling the server program to find out that it is restricted to the distributed program link subset. See "ASSIGN" on page 327 for details.

Exception conditions for EXEC CICS LINK command

There are some exception conditions on the EXEC CICS LINK command that relate specifically to DPL. See page 380 for a description of these exception conditions.

Exception conditions returned to the client program: Condition codes returned to a client program describe such events as "remote system not known" or "failure to commit" in the server program. There are different reasons, identified by EIBRESP2 values, for raising the INVREQ and LENGERR conditions on an EXEC CICS LINK command. The ROLLEDBACK, SYSIDERR, and TERMERR conditions may also be raised. See page 380 for details.

The PGMIDERR condition is raised on the EXEC CICS HANDLE ABEND PROGRAM, EXEC CICS LOAD, EXEC CICS RELEASE, and EXEC CICS XCTL commands if the local program definition specifies that the program is remote. This exception is qualified by an EIBRESP2 value of 9.

Exception conditions returned to the server program: The INVREQ condition covers the use of prohibited API commands. INVREQ is returned, qualified by an EIBRESP2 value of 200, to a server program if it issues one of the prohibited commands summarized in Figure 38 on page 185. If the server program does not handle the INVREQ condition, the default action is to abend the mirror transaction under which the server program is running with abend code ADPL.

- ADDRESS
 - TCTUA
- ASSIGN
 - BTRANS COLOR EXTDS FACILITY FCI HILIGHT
 - KATAKANA MAPCOLUMN MAPHEIGHT MAPLINE
 - MAPWIDTH OPCLASS OUTLINE PS QNAME SCRNHT SCRNWD
 - SIGDATA SOSI TCTUALENG TERMCODE UNATTEND VALIDATION
- CONNECT PROCESS
- CONVERSE
- EXTRACT ATTRIBUTES
- EXTRACT PROCESS
- FREE
- HANDLE AID
- ISSUE ABEND
- ISSUE CONFIRMATION
- ISSUE ERASEAUP
- ISSUE ERROR
- ISSUE SIGNAL
- LINK
 - INPUTMSG INPUTMSGLEN
- RECEIVE
- RECEIVE MAP
- RETURN
 - INPUTMSG INPUTMSGLEN
- SEND
- SEND CONTROL
- SEND MAP
- SEND TEXT
- SYNCPOINT (but can be issued in the server program if SYNCONRETURN is specified on the EXEC CICS LINK command)
- WAIT CONVID
- XCTL
 - INPUTMSG INPUTMSGLEN

Figure 38. API commands prohibited in programs invoked by distributed program link

Note: Where only certain options are prohibited on the command, they are shown. All the APPC commands listed are prohibited only when they refer to the principal facility. One of these, the EXEC CICS CONNECT PROCESS command, causes an error even if it refers to the principal facility in a non-DPL environment. It is included here because, if an EXEC CICS CONNECT PROCESS command refers to its principal facility in a server program, the exception condition raised indicates a DPL error.

Asynchronous processing

The response from a remotely initiated transaction is not necessarily returned to the task that initiated the transaction, which is why the processing is referred to as **asynchronous**. Asynchronous processing is useful when you do not need or want to tie up local resources while having a remote request processed. For example,

with online inquiry on remote databases, terminal operators can continue entering inquiries without having to wait for an answer to the first one.

You can start a transaction on a remote system using an EXEC CICS START command just like a local transaction. You can use the EXEC CICS RETRIEVE command to retrieve data that has been stored for a task as a result of a remotely issued EXEC CICS START, EXEC CICS CANCEL, EXEC CICS SEND, or EXEC CICS RECEIVE command, as if it were a local transaction.

Distributed transaction processing (DTP)

The main advantage of DTP is that it allows the two transactions to have exclusive control of a session and to “converse”. DTP is particularly useful when you need remote resources to be processed remotely or if you need to transfer data between systems. It also allows you to design flexible and efficient applications.

DTP can be used with a variety of partners, including both CICS and non-CICS platforms, as long as they support APPC. For further information, see the *CICS Family: Interproduct Communication* book and the *CICS for iSeries Intercommunication* book.

Common Programming Interface Communications (CPI Communications)

Systems Application Architecture (SAA) CPI Communications provides an alternative API to existing CICS APPC support. CPI Communications can be used in SAA languages (outside of CICS/400 API support).

CPI Communications defines an API that can be used in APPC networks that include multiple system platforms, where the consistency of a common API is seen to be of benefit.

The CPI Communications interface can converse with applications on any system that provides an APPC API. This includes applications on CICS platforms. You may use EXEC CICS APPC API commands on one end of a conversation and CPI Communications commands on the other.

CPI Communications is not directly supported by CICS/400. CPI Communications API usage is managed by native OS/400 communication facilities. For information on restrictions, see the *CICS for iSeries Intercommunication* book.

Part 5. CICS management functions

Chapter 16. Control region	189
Chapter 17. Application shell	191
Chapter 18. Interval control	193
Timer-related tasks	194
Expiration times	194
Request identifiers.	196
Chapter 19. Task control	197
Chapter 20. Program control	199
Defining and using CICS tables	199
Application program logical levels	200
Link to another program expecting return.	200
Passing data to other programs	201
COMMAREA	201
INPUTMSG	203
Using the INPUTMSG option on the EXEC	
CICS RETURN command	205
Other ways of passing data.	205
Examples of passing data	205
Chapter 21. Access to system information.	211
System programming commands	211
EXEC interface block (EIB)	211
Chapter 22. Storage control	213
Chapter 23. Transient data control	215
Intrapartition destinations	215
Extrapartition destinations	215
Indirect destinations	216
Automatic transaction initiation (ATI)	216
Chapter 24. Temporary storage control	219
Temporary storage queues	219
Temporary storage commands.	220
Typical uses of temporary storage control	220
Chapter 25. Printer spooling.	223
When are printer spooling files closed?.	223

Chapter 16. Control region

The control region (CR) facility provides for the control, scheduling, and work management mechanisms necessary to coordinate all the shared resources of a CICS environment:

- Shared system and user storage areas are initialized, managed, and released.
- Runtime resource definitions are built and validated.
- Resource status is initialized or reset depending on startup options.
- Cleanup of shared resources from previous control region execution is provided when appropriate.

The application programmer interfaces with the application shell facility in CICS/400. No user programs actually run in the control region; only CICS service modules do. A brief description is provided here because the control region is the controlling mechanism that allows the user shells to run and gain access to the necessary control blocks.

Control region initialization builds the shared resource environment common to all CICS jobs running under the same CTRLGN (or SYSID).

Initialization consists of those tasks that are performed once per system, and which affect all users:

- Making shared system code available
- Acquiring shared storage areas
- Loading CICS resource tables
- Building shared control blocks (for example, Common Work Area)
- Establishing entry points for CICS

When control region initialization is complete, CICS users are allowed to access CICS transactions through use of the application shell facility. The shell must be initiated with the same CTRLGN as the control region under which it is to run.

During the **processing** loop, the control region waits for shell service requests, shell status updates, or timer-driven processes.

- Shell service requests include resource status changes, extrapartition transient data requests, completion of ISC Logical Units of Work (LUW), and control region status changes.
- Timer events include expiration of interval control start request times, and periodic housekeeping requests.

Shutdown of the control region can be caused by:

- The master terminal operator entering the CEMT PERFORM SHUTDOWN command
- Operator-initiated ENDSBS command, for the control region currently running under a separate subsystem
- Operator-initiated ENDCICS CL command
- Critical internal errors

The OS/400 system operator is notified that shutdown has begun. No new work may enter the system. Shell startup requests are rejected. The batch and user shells are notified by user queue messages of the control region shutdown and type of shutdown. If a non-immediate shutdown is requested, the control region waits until all the batch and user shells have completed. If an immediate shutdown is requested, the control region sends immediate shutdown messages to all shell jobs.

Note: The STRCICS CL command used to bring up a CICS/400 control region, and the ENDCICS CL command used to end an active control region are described in more detail in the *CICS for iSeries Administration and Operations Guide*. Both commands are normally initiated by the system administrator.

Chapter 17. Application shell

The application shell provides the task scheduling work management mechanisms needed to build and refresh the application programming environment for CICS transactions. The CICS application shell provides CICS programs with an interface to resources managed by host language calls.

CICS application programs are executed only in an application shell. There are three types of shell:

- **Interactive** shells are started when a user causes a STRCICSUSR CL command to be entered.
- **IC batch** shells are started and stopped under the control of the interval control facilities running in the control region.
- **ISC inbound** shells are started either by a “start prestart job” (STRPJ) command issued at control region startup, or by the inbound OS/400 EVOKE processing.

Transactions are scheduled from four sources:

- Terminal input (including the STRCICSUSR CL command)
- Automatic transaction initiation (ATI) by transient data queue trigger levels
- Interval control starts
- Inbound intersystem communication (ISC) requests

CICS terminal users either explicitly or implicitly (by user profile or OS/400 menu selection) cause a STRCICSUSR CL command to be entered. Automatic transaction initialization and interval control starts are processed in a similar manner. They are essentially timer-driven transactions.

An interactive or terminal shell is started by the following OS/400 control language (CL) command, where *sysid* refers to the control region this shell is associated with:

```
STRCICSUSR CTLRGN(sysid)
```

For a description of other optional parameters that can be used, and a more detailed explanation of this command, see the *CICS for iSeries Administration and Operations Guide*.

CICS-managed batch shells are started with an internal CICS transaction that completes the initialization of the specific batch shell function. CICS-managed batch shells are terminated when their function is complete, when ISC links are disabled, or by CICS shutdown.

User application initialization builds the background environment common to all CICS sessions. This includes:

- Verifying that the CICS control region is up and available
- Invoking shell resource initialization routines
- Establishing a protected environment to insulate CICS from application errors
- Optionally starting an initial transaction

Initialization also consists of those tasks which must be performed individually for each user the first time they run a CICS transaction:

- Opening shared files and queues
- Opening private files and queues (if any)
- Acquiring storage areas unique to each user
- Building private control blocks (for example: task control area (TCA), terminal control table terminal entries (TCTTE), and EXEC interface block (EIB))

When initialization is complete, a CICS user is allowed to enter transactions.

After initialization, the shell coordinates the repetitive process of event (ATI, IC START, ISC, or terminal input) scheduling and task termination with syncpoint and rollback processing.

Note: The shell can receive work from a data queue, display file, or ICF file.

The runtime processing of the shell can be described in the following steps:

1. Check for interval control activity.
2. Call terminal control to accept input by means of data queue records.
3. If a CICS data queue message is returned, call the internal CICS function.
4. If a terminal input output area (TIOA) is returned:
 - a. Determine a transaction ID
 - b. Ensure that the transaction is valid and available
 - c. Initialize the EIB
 - d. Link to the application program
 - e. Perform a default syncpoint commit or rollback at task termination
 - f. Perform a scan of task termination deferred work elements (DWE) for work ready to process
5. Check whether CICS is in shutdown mode. If not, processing is repeated until shutdown is indicated.

Termination can be caused by one of the following:

- Terminal input (CESF transaction)
- If a “single shot transaction” was specified (an initial transaction was entered for the TRANID parameter on the STRCICSUSR CL command)
- Completion of the ATI or IC START work (nonterminal shells)
- ISC link shutdown
- CICS control region shutdown requests

Termination provides for a controlled reversal of the initialization process. CICS resources are freed. Work is allowed to complete unless an immediate shutdown is indicated. The user is returned to the point from which the STRCICSUSR CL command was issued.

If termination is caused by a CICS shutdown, the user is informed of the involuntary shutdown. Control is returned to OS/400 facilities.

Nonterminal users are terminated by internal data queue commands or ISC conversation link termination.

Chapter 18. Interval control

The CICS interval control program, in conjunction with a time-of-day machine instruction (MI) interface maintained by CICS, provides functions that can be performed at a specific time; such functions are known as **time-controlled** functions. The primary task of the CICS interval control facility is the handling, synchronization, and initiation of tasks requested by user application programs and CICS internal service modules. Other functions include obtaining and formatting time requests for the user.

Using interval control commands, you can:

- Start a task at a specified time or after a specified interval and pass data to it (EXEC CICS START command).
- Retrieve data passed on an EXEC CICS START command (EXEC CICS RETRIEVE command).
- Delay the processing of a task (EXEC CICS DELAY command).
- Request notification when a specified time has expired (EXEC CICS POST command).
- Wait for an event to occur (EXEC CICS WAIT EVENT command).
- Cancel the effect of previous interval control commands (EXEC CICS CANCEL command).
- Request the current date and time of day (EXEC CICS ASKTIME command).
- Select the format of date and time (EXEC CICS FORMATTIME command).

The above CICS commands may be divided into four groups:

1. Retrieving and formatting time

The EXEC CICS ASKTIME and EXEC CICS FORMATTIME commands merely request or format timer values. The time is obtained from MI interfaces (EXEC CICS ASKTIME command), or supplied by an application program (EXEC CICS FORMATTIME command), and formatted according to the options requested.

2. Timer-related event control management

The EXEC CICS DELAY, EXEC CICS POST, and EXEC CICS WAIT EVENT commands cause processing to be suspended or restarted after a time interval.

3. Timer-related task initialization

The EXEC CICS START command is associated with starting a new CICS task. A specified task may be started, either at a given terminal (that may be remote), or without a terminal, at a specified time (immediately or in the future). Optionally data may be passed to that task. The data can then be accessed by the started task by means of the EXEC CICS RETRIEVE command.

4. Cancellation of a timer-related event or task

The EXEC CICS CANCEL command nullifies a previous (but still unfulfilled) interval control request. This previous request may have been an EXEC CICS DELAY, EXEC CICS POST, or EXEC CICS START command. The appropriate entry is removed from the list of pending events, so the result is as if the corresponding request had never been issued. EXEC CICS CANCEL can also be used to nullify an unexpired EXEC CICS START command on a remote system.

For more details about specific CICS commands, see Chapter 32, "Application programming commands - reference," on page 323.

Timer-related tasks

Interval control includes timer-event driven tasks that initiate actions scheduled by CICS commands. This facility examines the interval control element (ICE) storage chain contained in the CICS shared system storage for expired events, and initiates the appropriate action. After the ICE has expired, its entry is removed from the chain.

A timer-related task can be in any of three states:

- Unexpired—the expiration time for the task or event is still in the future
- Expired but suspended due to a CICS resource being unavailable
- Expired and awaiting execution

When a task expires, interval control attempts to schedule the task for execution if all the CICS resources that it requires are available. If any of the resources are not available, the entry is said to be suspended waiting for a resource to become available. When it is determined that all resources are available, the entry is then said to be enabled awaiting execution.

Expiration times

The time at which a time-controlled function is to be started is known as the **expiration time**. You can specify expiration times absolutely, as a time of day (using the TIME option), or as an interval that is to elapse before the function is to be performed (using the INTERVAL option). For the EXEC CICS DELAY, EXEC CICS POST, and EXEC CICS START commands, both INTERVAL and TIME options are available. For the EXEC CICS DELAY command, you can use the FOR and UNTIL options; and for the EXEC CICS START command, you can use the AFTER and AT[®] options. See Chapter 32, “Application programming commands - reference,” on page 323 for information about these commands.

You use an **interval** to tell CICS to start a transaction in a specified number of hours, minutes, and seconds from the current time. A nonzero INTERVAL value always indicates a time in the future—the current time plus the interval you specify. The hours may be 0–99, but the minutes and seconds must not be greater than 59. For example, to start a task in 40 hours and 10 minutes, you would code:

```
EXEC CICS START INTERVAL(401000)
```

To use the AFTER option for the same example, you would code:

```
EXEC CICS START AFTER HOURS(40) MINUTES(10)
or
EXEC CICS START AFTER MINUTES(2410)
```

You use an **absolute time** to tell CICS to start a transaction at a specific time, again using hhmmss. For example, to start a transaction at 3:30 in the afternoon, you would code:

```
EXEC CICS START TIME(153000)
```

An absolute time is always relative to the midnight before the current time and may therefore be earlier than the current time. TIME may be either in the future or the past relative to the time at which the command is executed.

To use the AT option for the same example, you would code:

```
EXEC CICS START AT HOURS(15) MINUTES(30)  
or  
EXEC CICS START AT MINUTES(930)
```

CICS uses the following rules:

- If you specify a task to start at any time within the previous six hours, it will start immediately, unless the start time is before midnight (past) of the day on which you specify it. For example:

```
EXEC CICS START TIME(123000)
```

This command, issued at 05:00 or 07:00 on Monday expires at 12:30 on the same day.

```
EXEC CICS START TIME(020000)
```

This command, issued at 05:00 or 07:00 on Monday expires immediately because the specified time is within the preceding six hours.

```
EXEC CICS START TIME(003000)
```

This command, issued at 05:00 on Monday expires immediately because the specified time is within the preceding six hours. However, if it is issued at 07:00 on Monday, it expires at 00:30 on Tuesday, because the specified time is not within the preceding six hours.

```
EXEC CICS START TIME(230000)
```

This command, issued at 02:00 on Monday expires immediately because the specified time is within the preceding six hours.

- If you specify a time with an hours component that is greater than 23, you are specifying a time on a day following the current one. For example, a time of 250000 means 1 a.m. on the day following the current one, and 490000 means 1 a.m. on the day after that.

If you do not specify an expiration time or interval option on the EXEC CICS DELAY, EXEC CICS POST, or EXEC CICS START command, CICS responds using the default of INTERVAL(0), which means immediately.

Because each end of an intersystem link may be in a different time zone, you should use the INTERVAL form of expiration time when the transaction to be started is in a remote system.

If the system fails, the times you have specified are retained as interval control elements (ICEs) in recoverable temporary storage.

Request identifiers

As a means of identifying the request and any data associated with it, a unique request identifier is assigned by CICS to each EXEC CICS DELAY, EXEC CICS POST, and EXEC CICS START command. You can specify your own request identifier by means of the REQID option but it must be unique. If you do not, CICS assigns (for EXEC CICS POST and EXEC CICS START commands only) a unique request identifier and places it in field EIBREQID in the EXEC interface block (EIB). You must specify a request identifier if you want the request to be canceled at a later time by an EXEC CICS CANCEL command.

Chapter 19. Task control

The CICS task control program provides functions that synchronize task activity, or that control the use of resources. Within CICS/400, enqueueing and dequeuing of resources occurs in the CICS/400 user shells. The OS/400 system administrator should distribute shell priorities so that a user shell does not enqueue a resource, and then not have a priority high enough to allow it to complete processing.

OS/400 processes tasks according to priorities assigned by the CICS/400 system administrator. Control is given to the highest priority task that is ready to be processed, and is returned to the operating system when no further work can be done by CICS or by your application programs.

Task control supports the EXEC CICS ENQ and EXEC CICS DEQ commands. In CICS/400 the EXEC CICS SUSPEND command is acceptable as a current command, but is treated as a null operation. These commands are described in more detail in “DEQ” on page 350 and “ENQ” on page 355.

You can schedule the use of a resource by a task (ENQ and DEQ). This is often useful in protecting a resource from concurrent use by more than one task; that is, making that resource serially reusable. Each task that is to use the resource issues an enqueue (EXEC CICS ENQ) command. The first task to enqueue has the use of the resource immediately. Subsequent EXEC CICS ENQ commands for the resource, issued by other tasks, result in those tasks being suspended until the resource is available (assuming that an EXEC CICS HANDLE CONDITION ENQBUSY command has not been issued). Using the ENQ/DEQ mechanism, there is no way to guarantee that two or more tasks issuing EXEC CICS ENQ and EXEC CICS DEQ commands will issue these commands in a given sequence relative to each other. By inspecting the contents of the EIBRESP field, you can check whether the EXEC CICS ENQ command was successful or not. For more information, see “How to use the RESP and RESP2 options” on page 87.

Each task using a resource should issue a dequeue (EXEC CICS DEQ) command when finished, to release the resource for use by other tasks. The enqueue is also released automatically at the end of the task, or of the logical unit of work (LUW) if task is not specified on the EXEC CICS ENQ command.

Chapter 20. Program control

The CICS program control program governs the flow of control between application programs in a CICS system. The name of the application referred to in a program control command must have been defined as a program to CICS. You can use program control commands to:

- Link one of your application programs to another program in the same region, anticipating subsequent return to the requesting program (EXEC CICS LINK command). The COMMAREA and INPUTMSG options of this command allow data to be passed to the requested application program.
- Link one of your application programs to another program in a separate CICS region, anticipating subsequent return to the requesting program (EXEC CICS LINK command). The COMMAREA option of this command allows data to be passed to the requested application program. This is referred to as distributed program link (DPL). You cannot use the INPUTMSG and INPUTMSGLEN options of the EXEC CICS LINK command when using DPL. See page 380 for more details about this restriction. For more information about DPL, see Chapter 15, "Intercommunication considerations," on page 175.
- Transfer control from one of your application programs to another, with no return to the requesting program (EXEC CICS XCTL command). The COMMAREA and INPUTMSG options of this command allow data to be passed to the requested application program. You cannot use the INPUTMSG and INPUTMSGLEN options of the EXEC CICS XCTL command when using DPL. See 424 for more details about this restriction.
- Return control from one of your application programs to another, or to CICS (EXEC CICS RETURN command). The COMMAREA and INPUTMSG options of this command allow data to be passed to a newly initiated transaction. You cannot use the INPUTMSG and INPUTMSGLEN options of the EXEC CICS RETURN command when using DPL. See 424 for more details about this restriction.

You can use the RESP option to deal with abnormal terminations.

Defining and using CICS tables

The equivalent of a CICS table on CICS/400 is a *USRSPC object. The *USRSPC objects are identified to CICS/400 by PPT entries with CICSMAP(*YES) specified. For example:

```
ADDICSPPT LIB(QCICSSAMP) GROUP(USPC) PG MID(USERTBL)
          CICSMAP(*YES) PGM OBJ(TESTER/USERTABLE)
```

If you are using CICS tables, program control commands allow you to:

- Provide first-byte addressability to a *USRSPC object defined to CICS/400 as a map set (LOAD).

Note: EXEC CICS LOAD cannot be used in CICS/400 to load an application program.

- Remove first-byte addressability to a *USRSPC object defined to CICS/400 as a map (EXEC CICS RELEASE command) that has been previously loaded.

To load a *USRSPC object, use the following LOAD commands:

```
EXEC CICS LOAD PROGRAM(USERTBL)
          SET(1s-header-record)
:
```

If the contents of a *USRSPC are changed by a CICS program, the *USRSPC will be changed for all users accessing the *USRSPC and the changes are permanent across all user shells and control region starts. (This is not true for other CICS platforms.) You can avoid this in the following ways:

- Put the *USRSPC in a library that is not accessed by CICS or give it a name that is not used by CICS. Use the CRTDUPOBJ command using the name in the PPT either before the control region is started or before each user shell is started.
- Use the OS/400 security facilities to make the *USRSPC object read only. See the *iSeries Security Reference* for details.

Application program logical levels

Application programs running under CICS are executed at various logical levels. The first program to receive control within a task is at the highest logical level. When an application program is linked to another, expecting an eventual return of control, the linked-to program is considered to reside at the next lower logical level. When control is simply transferred from one application program to another, without expecting return of control, the two programs are considered to reside at the same logical level.

Link to another program expecting return

The EXEC CICS LINK command is used to pass control from an application program at one logical level to an application program at the next lower logical level. If the linked-to program is not already in main storage, it is loaded. When the EXEC CICS RETURN command is processed in the linked-to program, control is returned to the program that initiated the link, at the next sequential process instruction.

The linked-to program operates independently of the program that issues the EXEC CICS LINK command with regard to handling exception conditions, attention identifiers, and abends. For example, the effects of EXEC CICS HANDLE commands in the linking program are not inherited by the linked-to program, but the original EXEC CICS HANDLE commands are restored on return to the linking program. You can use the EXEC CICS HANDLE ABEND command to deal with abnormal terminations in other link levels. See “HANDLE ABEND” on page 369 for information about this command. Figure 39 shows the concept of logical levels.

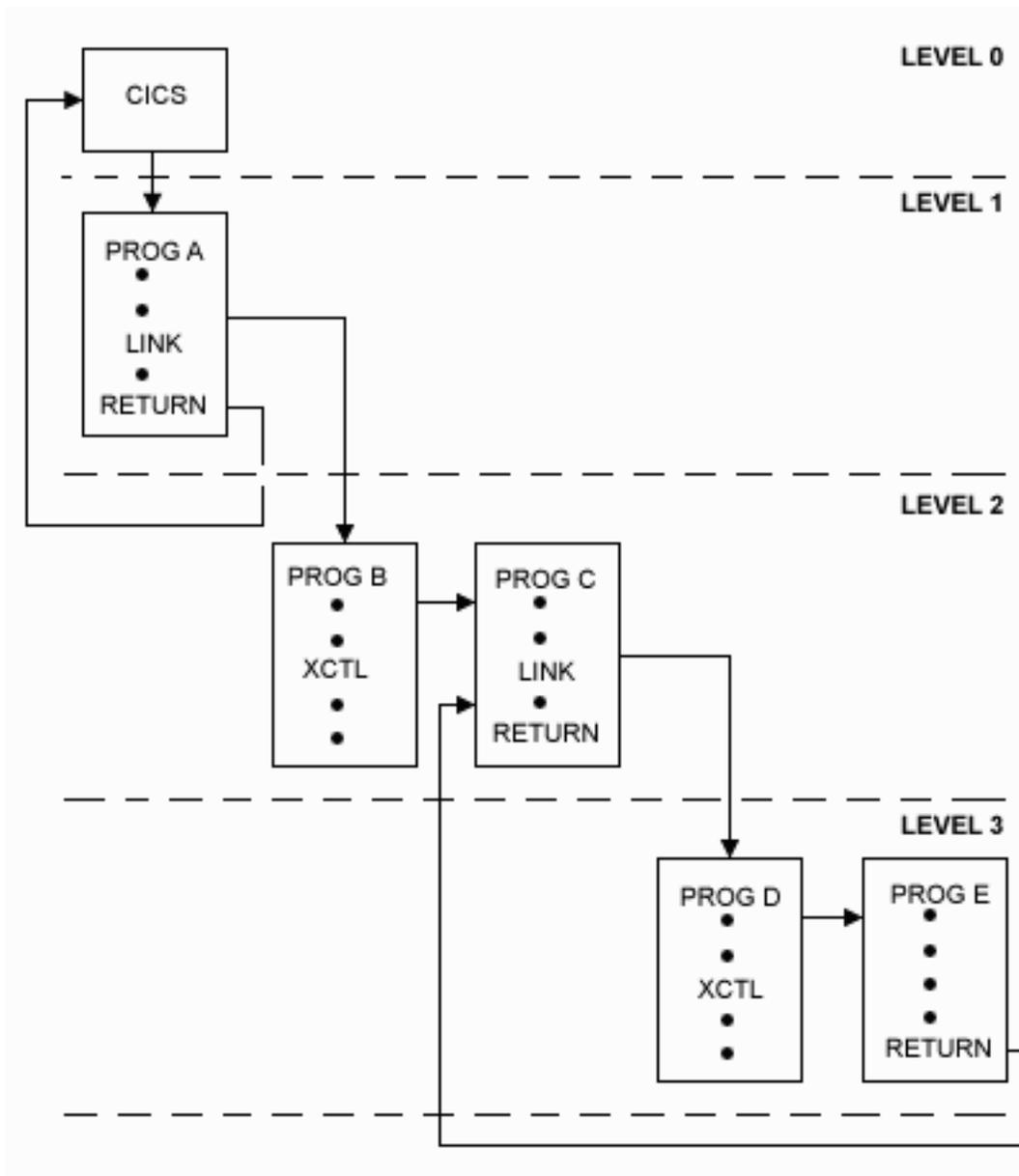


Figure 39. Application program logical levels

Passing data to other programs

You can pass data to another program when control is passed to that other program using a program control command.

COMMAREA

The COMMAREA option of the EXEC CICS LINK and EXEC CICS XCTL commands specifies the name of a data area (known as a **communication area**) in which data is passed to the program being invoked.

In a similar manner, the COMMAREA option of the EXEC CICS RETURN command specifies the name of a communication area in which data is passed to

the transaction identified in the TRANSID option. (The TRANSID option specifies a transaction that is initiated when the next input is received from the terminal associated with the task.)

The invoked program receives the data as a parameter. The program must contain a definition of a data area to allow access to the passed data.

In a receiving COBOL program, you must give the data area the name DFHCOMMAREA. If a COBOL program passes a COMMAREA as part of an EXEC CICS LINK, an EXEC CICS XCTL, or an EXEC CICS RETURN command, either the working-storage or the linkage section can contain the data area. A program receiving a COMMAREA should specify the data in the linkage section. This applies when the program is either of the following:

- The receiving program during an EXEC CICS LINK or an EXEC CICS XCTL command where a COMMAREA is passed
- The initial program where the EXEC CICS RETURN command of a previously called task specified a COMMAREA and TRANSID

In a ILE C program that is receiving a COMMAREA, the COMMAREA must be defined as a structure based on a pointer. The program then must issue the EXEC CICS ADDRESS COMMAREA command to gain addressability to the passed data.

The receiving data area need not be of the same length as the original communication area; if access is required only to the first part of the data, the new data area can be shorter. However, it must not be longer than the length of the communication area being passed, because if it is, your transaction may inadvertently attempt to read data outside the area that has been passed. This may cause your transaction to have unpredictable results. It may also overwrite data outside the area, which could cause an abend.

To avoid this happening, your program should check whether the length of any communication area that has been passed to it is as expected by accessing the EIBCALEN field in the EIB of the task. If no communication area has been passed, the value of EIBCALEN is zero; otherwise, EIBCALEN always contains the value specified in the LENGTH option of the EXEC CICS LINK, EXEC CICS XCTL, or EXEC CICS RETURN command, regardless of the size of the data area in the invoked program. You should ensure that the value in EIBCALEN matches the value expected by your program, and make sure that your transaction is accessing data within that area.

You may also add an identifier to COMMAREA as an additional check on the data that is being passed. This identifier is sent with the sending transaction and checked for by the receiving transaction.

When a communication area is passed using an EXEC CICS LINK command, the invoked program is passed a pointer to the communication area itself. Any changes made to the contents of the data area in the invoked program are available to the invoking program, when control returns to it; to access any such changes, the program names the data area specified in the original COMMAREA option.

When a communication area is passed using an EXEC CICS XCTL command, a copy of that area is made unless the area to be passed has the same address and length as the area that was passed to the program issuing the command. For example in Figure 39 on page 201, if program A issues an EXEC CICS LINK command to program B, which in turn issues an EXEC CICS XCTL command to

program C, and if B passes to C the same communication area that A passed to B, program C will be passed addressability to the communication area that belongs to A (not a copy of it), and any changes made by C will be available to A when control returns to it.

When a lower-level program, which is a linked-to program, issues the EXEC CICS RETURN command, control passes back one logical level higher than the program returning control. If the task is associated with a terminal, the TRANSID option can be used at the lower level to specify the transaction identifier for the next transaction to be associated with that terminal. The transaction identifier comes into play only after the highest logical level has relinquished control to CICS using the EXEC CICS RETURN command and input is received from the terminal. Any input entered from the terminal, apart from an attention key, is interpreted wholly as data. You may use the TRANSID option without COMMAREA when returning from any link level, but be aware that it might be overridden on a later EXEC CICS RETURN command. If an EXEC CICS RETURN command fails at the top level because of an invalid COMMAREA, the TRANSID becomes null. Also, you can only specify COMMAREA or IMMEDIATE at the highest level, otherwise you will get an INVREQ with RESP2=2.

In addition, the COMMAREA option can be used to pass data to the new task that is to be started.

The invoked program can determine which type of command invoked it by accessing field EIBFN in the EIB. This field must be tested before any CICS commands are issued. If the program was invoked by an EXEC CICS LINK or EXEC CICS XCTL command, the appropriate code is found in the EIBFN field; if it was invoked by an EXEC CICS RETURN command, no CICS commands have been issued in the task, and the field contains zeros. Figures 41 through 44 show some examples of passing data.

INPUTMSG

The INPUTMSG option of the EXEC CICS LINK, EXEC CICS XCTL, and EXEC CICS RETURN commands is another way of specifying the name of a data area to be passed to the program being invoked. For information about the use of these commands see Chapter 32, "Application programming commands - reference," on page 323. In this case the invoked program gets the data by processing an EXEC CICS RECEIVE command. This option enables you to invoke ("front-end") application programs that were written to be invoked directly from a terminal, and which contain EXEC CICS RECEIVE commands to obtain initial terminal input.

If a linked-to program issues an EXEC CICS RECEIVE command to obtain initial input from a terminal, but the initial RECEIVE request has already been issued by a higher-level program, there is no data for the program to receive. In this case, the application waits on input from the terminal. You can ensure that the original terminal input continues to be available to a linked-to program by invoking it with the INPUTMSG option.

When an application program invokes another program, specifying INPUTMSG on the EXEC CICS LINK (or EXEC CICS XCTL or EXEC CICS RETURN) command, the data specified on the INPUTMSG continues to be available even if the linked-to program itself does not issue an EXEC CICS RECEIVE command, but instead invokes yet another application program. See Figure 40 on page 204 for an illustration of INPUTMSG.

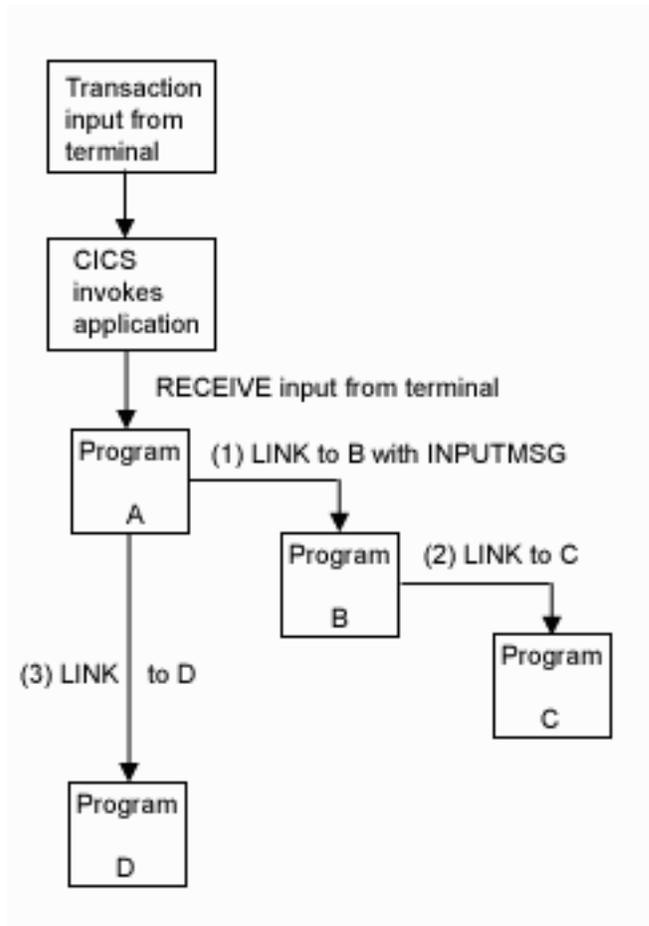


Figure 40. Use of INPUTMSG in a linked-to chain

Notes:

1. In this example, the “real” first EXEC CICS RECEIVE command is issued by program A. By linking to program B with the INPUTMSG option, it ensures that the next program to issue an EXEC CICS RECEIVE request can also receive the terminal input. This can be either program B or program C.
2. If program A simply wants to pass on the unmodified terminal input that it received, it can name the same data area for the INPUTMSG option that it used for the EXEC CICS RECEIVE command. For example:

```

EXEC CICS RECEIVE
      INTO(TERMINAL-INPUT)
:
:
EXEC CICS LINK
      PROGRAM(PROGRAMB)
      INPUTMSG(TERMINAL-INPUT)
:
:

```

3. As soon as one program in a linked-to chain issues an EXEC CICS RECEIVE command, the INPUTMSG data ceases to be available to any subsequent EXEC CICS RECEIVE command. In other words, in the example shown, if B issues an EXEC CICS RECEIVE request before linking to C, the INPUTMSG data area is not available for C.

4. This method of communicating data from one program to another can be used for any kind of data—it does not have to originate from a user terminal. In our example, program A could move any data into the named data area, and invoke program B with INPUTMSG referencing the data.
5. The “terminal-data” passed on INPUTMSG also ceases to be available when control is eventually returned to the program that issued the link with INPUTMSG. In our example, if C returns to B, and B returns to A, and neither B nor C issues an EXEC CICS RECEIVE command, the data is assumed by A to have been received. If A then invokes another program (for example, D), the original INPUTMSG data is no longer available to D, unless the INPUTMSG option is specified.

Using the INPUTMSG option on the EXEC CICS RETURN command

You can specify INPUTMSG to pass data to the next transaction specified on an EXEC CICS RETURN command with the TRANSID option. To do this, the EXEC CICS RETURN command must be issued at the highest logical level to return control to CICS, and the command must also specify the IMMEDIATE option. If you specify INPUTMSG with TRANSID, and do not also specify IMMEDIATE, the next real input from the terminal overrides the INPUTMSG data, which is therefore lost. See page 424 for details of the EXEC CICS RETURN command.

Other ways of passing data

Data can also be passed between application programs and transactions in other ways. For example, the data can be stored in a CICS storage area outside the local environment of the application program, such as the transaction work area (TWA). Another way is to store the data in temporary storage; see Chapter 24, “Temporary storage control,” on page 219 for details.

Examples of passing data

Figures 41 and 42 show how, in COBOL and ILE C, the EXEC CICS LINK command causes data to be passed to the program being linked to; the EXEC CICS XCTL command is coded in a similar way.

Figures 43 and 44 show how, in COBOL and C, the EXEC CICS RETURN command is used to pass data to a new transaction.

```

                                Invoking program
IDENTIFICATION DIVISION.
PROGRAM ID. 'PROG1'.
.
.
WORKING-STORAGE SECTION.
01 COM-REGION.
    02 FIELD PICTURE X(3).
.
.
PROCEDURE DIVISION.
    MOVE 'ABC' TO FIELD.
    EXEC CICS LINK PROGRAM('PROG2')
        COMMAREA(COM-REGION)
        LENGTH(3) END-EXEC.
.
.

                                Invoked program
IDENTIFICATION DIVISION.
PROGRAM-ID. 'PROG2'.
.
.
LINKAGE SECTION.
01 DFHCOMMAREA.
    02 FIELD PICTURE X(3).
.
.
PROCEDURE DIVISION.
    IF EIBCALEN GREATER ZERO
    THEN
        IF FIELD EQUALS 'ABC' ....

```

Figure 41. COBOL example—EXEC CICS LINK command

```

                Invoking program
                .
                .
struct com_struct
{
    unsigned char field[3];
} com_reg;
                .
                .
main()
{
    memcpy(com_reg.field,"ABC",3);
    EXEC CICS LINK PROGRAM("PROG2")
                COMMAREA(&com_reg)
                LENGTH(3);
}
                .
                .
                Invoked program
                .
                .
struct comm_struct
{
    unsigned char field[3];
} *commarea;
                .
                .
main()
{
    EXEC CICS ADDRESS EIB(dfheiptr);
    EXEC CICS ADDRESS COMMAREA(commarea);
    if (dfheiptr->eibcalen > 0)
    {
        if (memcmp(commarea->field,"ABC",3) == 0)
        {
            ....
        }
    }
}

```

Figure 42. ILE C example—EXEC CICS LINK command

```

                                Invoking program
IDENTIFICATION DIVISION.
PROGRAM-ID. 'PROG1'.
.
.
WORKING-STORAGE SECTION.
01  TERMINAL-STORAGE.
    02  FIELD PICTURE X(3).
    02  DATAFLD PICTURE X(17).
.
.
PROCEDURE DIVISION.
MOVE 'XYZ' TO FIELD.
EXEC CICS RETURN TRANSID('TRN2')
    COMMAREA(TERMINAL-STORAGE)
    LENGTH(20) END-EXEC.
.
.

```

```

                                Invoked program
IDENTIFICATION DIVISION.
PROGRAM-ID. 'PROG2'
.
.
LINKAGE SECTION.
01  DFHCOMMAREA.
    02  FIELD PICTURE X(3).
    02  DATAFLD PICTURE X(17).
.
.
PROCEDURE DIVISION.
IF EIBCALEN GREATER ZERO
THEN
    IF FIELD EQUALS 'XYZ'
        MOVE 'ABC' TO FIELD.
EXEC CICS RETURN END-EXEC.

```

Figure 43. COBOL example—EXEC CICS RETURN command

```

                                Invoking program
                                .
                                .
struct ter_struct
{
  unsigned char field [3];
  unsigned char datafld [17];
} ter_stor;
                                .
                                .
main()
{
  memcpy(ter_stor.field,"XYZ",3);
  EXEC CICS RETURN TRANSID("TRN2")
        COMMAREA(&ter_stor)
        LENGTH(sizeof(ter_stor));
}
                                .
                                .
                                Invoked program
                                .
                                .
struct term_struct
{
  unsigned char field[3];
  unsigned char datafld[17];
} *commarea;
                                .
                                .
main()
{
  EXEC CICS ADDRESS EIB(dfheiptr);
  EXEC CICS ADDRESS COMMAREA(commarea);
  if (dfheiptr->eibcalen > 0)
  {
    if (memcmp(commarea->field,"XYZ",3) == 0)
    {
      memcpy(commarea->field,"ABC",3);
    }
  }
  EXEC CICS RETURN;
}

```

Figure 44. ILE C example—EXEC CICS RETURN command

Chapter 21. Access to system information

You can write many application programs using the CICS command-level interface without any knowledge of, or reference to, the fields in the CICS control blocks and storage areas. However, you might sometimes need to get information that is valid outside the local environment of your application program. You use the EXEC CICS ADDRESS and EXEC CICS ASSIGN commands to access such information. For information about these commands, see page 324 and page 327 respectively.

When using the EXEC CICS ADDRESS and EXEC CICS ASSIGN commands, various fields can be read but should not be set or used in any other way. This means that you should not use any of the CICS fields as arguments in CICS commands, because these fields may be altered by the EXEC interface modules.

System programming commands

The EXEC CICS DISCARD, EXEC CICS INQUIRE, EXEC CICS SET, and EXEC CICS PERFORM commands allow application programs to access and modify information about CICS resources. The application program can retrieve and modify information for CICS files, terminals, system entries, system attributes, connections, queued requests, TD queues, tracing, TS queues, programs, and transactions. These commands are primarily for the use of the system administrator. For information, see Chapter 33, "System programming reference," on page 477.

EXEC interface block (EIB)

In addition to the usual CICS control blocks, each task in a command-level environment has a control block known as the EXEC interface block (EIB) associated with it.

An application program can access all of the fields in the EIB by name. The EIB contains information that is useful during the execution of an application program, such as the transaction identifier, the time and date (initially when the task is started, and subsequently, if updated by the application program), and the cursor position on a display device. The EIB also contains information that is helpful when a dump is being used to debug a program. For information about EIB fields, see Appendix A, "EXEC interface block," on page 529.

Chapter 22. Storage control

The CICS storage control program controls requests for main storage to provide intermediate work areas and other main storage not provided automatically by CICS but needed to process a transaction.

If you need working storage in addition to the working storage provided automatically by CICS, you can use the following commands:

- EXEC CICS GETMAIN to get and initialize main storage
- EXEC CICS FREEMAIN to release main storage

You can initialize the acquired main storage to any bit configuration by supplying the INITIMG option on the EXEC CICS GETMAIN command; for example, zeros or EBCDIC blanks.

CICS releases all main storage associated with a task when the task is ended normally or abnormally. This includes any storage acquired, and not subsequently released, by your application program, except for areas obtained with the SHARED option. This option of the EXEC CICS GETMAIN command prevents storage being released automatically when a task completes.

There are two types of storage that can be obtained through the EXEC CICS GETMAIN command:

Shared

Storage areas are available beyond a CICS task boundary and are accessible by any transaction. Shared storage must be returned to the system by using an EXEC CICS FREEMAIN command, or automatically at CICS control region shutdown.

Nonshared

Nonshared storage areas are available only to the CICS task that acquires them and only for the life of the task. Nonshared storage is returned to the system by using an EXEC CICS FREEMAIN command or automatically when the task terminates normally or abnormally.

For shared storage GETMAIN requests, storage control supports an implied SUSPEND option if sufficient storage for the request cannot be obtained. When this condition is present, storage control places the requesting task in a wait state until sufficient storage becomes available.

For a more thorough discussion of the EXEC CICS FREEMAIN and EXEC CICS GETMAIN commands and their options, see page 365 and page 367.

Note: If you do not specify NOSUSPEND on the EXEC CICS GETMAIN command, and the associated transaction has PURGE(*YES) and WAITTIME greater than zero, your transaction or CICS may fail. The PURGE and WAITTIME parameters are specified in the ADDCICSPCT CL command, when defining the CICS transaction.

OS/400 makes storage available for each COBOL/400 program, including the CICS COBOL program. Chapter 5, "Designing efficient applications," on page 61 describes storage within individual programs.

Chapter 23. Transient data control

The CICS transient data control program provides a generalized queuing facility. Data can be queued (stored) for subsequent internal or external processing. Selected data, specified in the application program, can be routed to or from predefined symbolic destinations, either **intrapartition** or **extrapartition**.

Destinations are intrapartition if associated with a facility allocated to the CICS system, and extrapartition if the data is directed to a destination that is external to the CICS system. The destinations must be defined in the destination control table (DCT) by the system administrator when the CICS control region is generated. A DCT entry can be added by using the CEDA transaction or by using the ADDCICSDCT CL command. Both the initial entry and the use of the CEDA transaction are described in the *CICS for iSeries Administration and Operations Guide*.

You can:

- Write data to a transient data queue (EXEC CICS WRITEQ TD command)
- Read data from a transient data queue (EXEC CICS READQ TD command)
- Delete an intrapartition transient data queue (EXEC CICS DELETEQ TD command)

If the TD keyword is omitted, the command is assumed to be for temporary storage. (See Chapter 24, “Temporary storage control,” on page 219.)

The EXEC CICS READQ TD command also causes the entry to be deleted (that is, it can be read only once).

Intrapartition destinations

“Intrapartition” refers to data on direct-access storage devices for use with one or more programs running as separate tasks. Data directed to or from these internal destinations is referred to as intrapartition data; it must consist of variable-length records. Intrapartition destinations can be associated with either a terminal or an output file. Intrapartition data may ultimately be transmitted upon request to the destination terminal or retrieved sequentially from the output file.

Typical uses of intrapartition data include:

- Message switching
- Broadcasting
- Database access
- Routing of output to several terminals (for example, for order distribution)
- Queuing of data (for example, for assignment of order numbers or priority by arrival)

Extrapartition destinations

Extrapartition destinations are queues (files) residing on any sequential device (DASD, tape, printer, and so on) that are accessible by programs outside (or within) the CICS system. In general, sequential extrapartition destinations are used for storing and retrieving data outside the CICS system. For example, one task may read data from a remote terminal, edit the data, and write the results to a

physical file for subsequent processing in another system. Logging data, statistics, and transaction error messages are examples of data that can be written to extrapartition destinations. In general, extrapartition data created by CICS is intended for subsequent batched input to non-CICS programs. Data can also be routed to an output device such as a printer.

Data directed to or from an external destination is referred to as extrapartition data and consists of sequential records that are fixed-length or variable-length. The record format for an extrapartition destination must be defined in the DCT by the system administrator. (Refer to the *CICS for iSeries Administration and Operations Guide* for details.)

All extrapartition transient data queue requests are processed through the control region. The security level of the individual user is not used.

Note that you cannot delete an extrapartition queue. If you try to do so, you cause an INVREQ condition.

Indirect destinations

Intrapartition and extrapartition destinations can be used as indirect destinations. Indirect destinations provide some flexibility in program maintenance in that data can be routed to one of several destinations with only the DCT, not the program, having to be changed.

When the DCT has been changed, application programs continue to route data to the destination using the original symbolic name; however, this name is now an indirect destination that refers to the new symbolic name. Because indirect destinations are established by means of DCT table entries, the application programmer need not usually be concerned with how this is done. Further information is available in the *CICS for iSeries Administration and Operations Guide*.

Automatic transaction initiation (ATI)

For intrapartition destinations, CICS provides the option of automatic transaction initiation (ATI). A basis for ATI is established by the system programmer by specifying a nonzero trigger level for a particular intrapartition destination in the DCT. When the number of entries (created by EXEC CICS WRITEQ TD commands issued by one or more programs) in the queue (destination) reaches the specified trigger level, a task specified in the definition of the destination is automatically initiated. Control is passed to a program that processes the data in the queue; the program must issue repetitive EXEC CICS READQ TD commands to deplete the queue.

When the queue has been emptied, a new ATI cycle begins. That is, a new task is scheduled for initiation when the specified trigger level is again reached, whether or not execution of the earlier task has ended.

If an automatically initiated task does not empty the queue, access to the queue is not inhibited.

The task may be normally or abnormally ended before the queue is emptied (that is, before a QZERO condition occurs in response to an EXEC CICS READQ TD command).

If the destination is a terminal, the fact that QZERO has not been reached means that trigger processing has not been reset and the same task will be reinitiated later. Without the reset of trigger processing, a subsequent EXEC CICS WRITEQ TD command will not trigger a new task.

If the destination is a file, the task termination has the same effect as QZERO (that is, trigger processing is reset) and the next EXEC CICS WRITEQ TD command initiates the trigger transaction (if the trigger level has been reached).

In addition, the task is not reinitiated if a resource (terminal, program, or transaction) is not available. If the trigger level of a queue is zero, no task is automatically initiated.

If a queue is logically recoverable, initiation of the trigger transaction is deferred until the next syncpoint.

If the trigger level is already exceeded because the last triggered transaction abended before clearing the queue, then for DEVTYPE(*TERMINAL) on the ADDCICSDCT CL command, another task is not scheduled because QZERO has not been raised to reset trigger processing. The already scheduled task is initiated again when possible. For DEVTYPE(*FILE) on the ADDCICSDCT CL command, the termination of the task resets trigger processing and so the next EXEC CICS WRITEQ TD command triggers a new task.

To ensure that completion of an automatically initiated task occurs when the queue is empty, the application program should test for a QZERO condition rather than for some application-dependent factor such as an anticipated number of records; only the QZERO condition indicates an emptied queue.

Chapter 24. Temporary storage control

The CICS temporary storage control program provides the application programmer with the ability to store data in temporary storage queues, either in main storage, or in auxiliary storage on a direct-access storage device. Data stored in a temporary storage queue is known as temporary data. Temporary storage provides a scratch-pad area for holding data created by one transaction, to be used later by the same transaction or by a different one.

You can:

- Write data to a temporary storage queue (EXEC CICS WRITEQ TS command)
- Update data in a temporary storage queue (EXEC CICS WRITEQ TS REWRITE command)
- Read data from a temporary storage queue (EXEC CICS READQ TS command)
- Read the next data from a temporary storage queue (EXEC CICS READQ TS NEXT command)
- Delete a temporary storage queue (EXEC CICS DELETEQ TS command)

Exception conditions that occur during execution of a temporary storage control command are handled as described in Chapter 6, “Dealing with exception conditions,” on page 87.

Temporary storage queues

Temporary storage queues are identified by symbolic names that may be up to eight characters assigned by the user application during program execution. Temporary data can be retrieved by the originating task or by any other task using the symbolic name assigned to it. To avoid conflicts caused by duplicate names, a naming convention should be established, for example, the operator identifier, terminal identifier, or transaction identifier could be used as a prefix or suffix to each programmer-supplied symbolic name. Specific items (logical records) within a queue are referred to by relative position numbers.

Temporary storage queues remain intact until they are deleted by the originating task, by any other task, or by a cold start; before deletion, they can be accessed any number of times. Even after the originating task is terminated, data on temporary storage queues can be accessed by other tasks through references to the symbolic name under which it is stored.

Temporary data can be stored either in main storage or in auxiliary storage.

- Main storage

Generally, main storage should be used if the data is needed for short periods of time and does not require recovery. Data in main storage does not survive from one CICS run to the next. Main storage might be used to pass data from task to task, or for unique storage that allows programs to meet the requirement of CICS that they be quasi-reentrant (that is, serially reusable between entry and exit points of the program).

- Auxiliary storage

In CICS, auxiliary storage is a physical file. Auxiliary storage should be used to store large amounts of data or data needed for a long period of time. Data

stored in auxiliary storage is retained after CICS termination and can be recovered in a subsequent restart. In CICS, data in auxiliary storage can be designated as either recoverable or nonrecoverable. This is indicated in the temporary storage table.

For a description of main and auxiliary storage queue usage, see “Temporary storage” on page 73.

Temporary storage commands

The EXEC CICS READQ TS command is used to read records from a temporary storage queue. Queue records may be read sequentially by using the NEXT option, or randomly by using the ITEM option. The EXEC CICS WRITEQ TS command is used to write records to a TS queue. The EXEC CICS DELETEQ TS command is used to delete the entire temporary storage queue; you cannot delete an individual record. For a discussion of these commands, see Chapter 32, “Application programming commands - reference,” on page 323.

Typical uses of temporary storage control

A temporary storage queue having only one record can be treated as a single unit of data that can be accessed using its symbolic name. Using temporary storage control in this way provides a typical “scratch-pad” capability. This type of storage should be accessed using the EXEC CICS READQ TS command with the ITEM option; not doing so may cause the ITEMERR condition to be raised.

In general, temporary storage queues of more than one record should be used only when direct access or repeated access to records is necessary. (Transient data control is better for efficient handling of sequential files.)

Some uses of temporary storage queues are:

Terminal paging

A task could retrieve a large master record from a direct-access file, format it into several screen images (using basic mapping support), store the screen images temporarily in auxiliary storage, and then ask the terminal operator which “page” (screen image) is desired. The application programmer can provide a program (as a generalized routine or unique to a single application) to advance page by page, advance or back up a relative number of pages, and so on.

A suspend file

Suppose a data collection task is in progress at a terminal. The task reads one or more units of input and then allows the terminal operator to interrupt the process by some kind of coded input. If not interrupted, the task repeats the data collection process. If interrupted, the task writes its “incomplete” data to temporary storage and terminates. The terminal is now free to process a different transaction (perhaps a high-priority inquiry). When the terminal is available to continue data collection, the operator initiates the task in a “resume” mode, causing the task to recall its suspended data from temporary storage and continue as though it had not been interrupted.

Preprinted forms

An application program can accept data to be written as output on a preprinted form. This data can be stored in temporary storage as it arrives.

When all the data has been stored, it can first be validated and then transmitted in the order required by the format of the preprinted form.

Chapter 25. Printer spooling

The CICS/400 printer spooling support allows access to OS/400 printer files. These printer files can be set up to print directly to an active writer or to spool the output.

The printer files must be defined to the system before the execution of the EXEC CICS SPOOLOPEN command. The CLASS option on the EXEC CICS SPOOLOPEN command is used to identify the printer file. Refer to the *CICS for iSeries Administration and Operations Guide* for information about creating these files.

After the EXEC CICS SPOOLOPEN command has been issued successfully, the TOKEN returned by CICS is used for subsequent printer spooling requests.

You must specify either the RESP option or the NOHANDLE option when you use any of the printer spooling commands. Failure to do so causes your program to abend with abend code APST.

When are printer spooling files closed?

Unlike other files opened by CICS, files opened by way of the EXEC CICS SPOOLOPEN command are closed under two conditions; at task termination, or when a syncpoint or commit occurs.

If an EXEC CICS SPOOLCLOSE or EXEC CICS SPOOLWRITE command is issued after a syncpoint occurs, without an intervening EXEC CICS SPOOLOPEN command, the EXEC CICS SPOOLCLOSE or EXEC CICS SPOOLWRITE command will fail.

Part 6. Supplied transactions

Chapter 26. Introduction to CICS-supplied transactions 227

Chapter 27. Execution diagnostic facility (EDF) 229

Getting started	229
Restrictions when using EDF	229
Where does EDF intercept the program?	230
What does EDF display?	231
The header	231
The body	231
How you can intervene in program execution	234
EDF menu functions	235
How to use EDF	238
Using EDF in single-screen mode.	238
Checking pseudoconversational programs	239
Using EDF in dual-screen mode	240
Stopping EDF	240
Overtyping to make changes	240
EDF responses	242

Chapter 28. Temporary storage browse (CEBR) 243

How to use the CEBR transaction	243
What does the CEBR transaction display?	244
The header	244
The command area	244
The body	244
The message line	245
The CEBR options on function keys	245
The CEBR commands	246
Using the CEBR transaction with transient data	248
Security considerations	248

Chapter 29. Command-level interpreter (CECI) 249

How to use CECI	249
What does CECI display?	250
The command line	250
The status line	251
Command syntax check	251
About to start command	252
Command completed.	252
The body	253
The message line	254
CECI options on function keys	254
Additional displays	255
Expanded area	255
Variables	255
Defining variables	256
The EXEC interface block (EIB)	257
Error messages display	257
Making changes	258
How CECI runs	258
CECI sessions	258
Abends	259
Exception conditions	259
Program control commands	259

Terminal Sharing	259
Saving commands	260
Security considerations	261

Chapter 26. Introduction to CICS-supplied transactions

In general, you start a CICS-supplied transaction by entering its CICS transaction identification code (transid) in the top left hand corner of the screen (column 1 line 1). The transid is used by CICS to identify the CICS application programs that handle the specified CICS transactions, and to establish a CICS task to process them.

CICS-supplied transactions have identification codes that start with “C”, and are 4 characters long; for example, CECI.

Three CICS-supplied transactions (CEBR, CECI, and CEDF) are used by applications programmers to debug their programs.

The **CEDF** transaction is the tool most commonly used to debug CICS programs. Execution Diagnostic Facility (EDF) lets you monitor a command-level program by intercepting all the CICS commands that the program issues.

As a CICS task is executed it is interrupted by CICS at several key points. At any of these points you can examine or change the contents of working storage or the EXEC interface block (EIB). These interception points are listed below:

- Before a new program begins to be executed
- Before any CICS command is executed
- After a CICS command is executed
- After a program ends
- When a CICS task ends
- Whenever an abend occurs

For full details on the use of the CEDF transaction, see Chapter 27, “Execution diagnostic facility (EDF),” on page 229.

The **CEBR** transaction lets you browse the contents of a specified CICS temporary storage queue. It is therefore useful if your program wrote data out to a CICS temporary storage queue. If you do not specify a default queue name, your terminal ID preceded by CEBR is used. For full details, see Chapter 28, “Temporary storage browse (CEBR),” on page 243.

The **CECI** transaction, the command-level interpreter, is a CICS-supplied transaction that interactively checks the syntax of CICS commands, and optionally executes the command.

You can enter part or all of a CICS command. The CECI transaction responds by displaying the complete syntax for the command. When you have selected the options you want, and the command passes the syntax check, you can then execute the command. Variables can be defined to pass data between commands. See “Variables” on page 255 for details on how to use this function in the CECI transaction.

The other CICS-supplied transactions available in CICS/400 are described in the *CICS for iSeries Administration and Operations Guide*.

Chapter 27. Execution diagnostic facility (EDF)

The CICS Execution Diagnostic Facility (EDF) can be used with the OS/400 COBOL and ILE C debugging tools to debug CICS application programs. See either the *COBOL/400 User's Guide* or the *WebSphere Development Studio: ILE COBOL Programmer's Guide* for debugging information.

You can use the execution diagnostic facility (EDF) to test an application program online, without modifying the program or the program-preparation procedure. The names of your programs must not begin with the letters "AEG" or "DFH" because these prefixes are used for CICS system modules and samples. Attempting to use the CEDF transaction on a CICS-supplied transaction has no effect. However, you can use the CEDF transaction with CICS sample programs.

EDF intercepts the execution of CICS commands in the application program at various points, allowing you to see what is happening. Each command is displayed before execution, and most are displayed after execution is complete. Screens sent by the application program are preserved, so you can converse with the application program during testing, just as a user would on a production system.

Each time EDF interrupts the execution of the application program a new CEDF task is started. Each CEDF task is short lived, lasting only long enough for the appropriate display to be processed.

Getting started

You can use EDF on the same terminal as the transaction you are testing or on a different one. On the same terminal, you **must** start by clearing the screen and entering the transaction code CEDF, otherwise you may get unpredictable results. The message: **THIS TERMINAL: EDF MODE ON** is displayed at the top of an empty screen. You clear the screen again and run your transaction in the normal way.

If you are using two terminals, you enter:

```
CEDF tttt
```

at one, naming the second in "tttt". Then you run your transaction on the second terminal. "How to use EDF" on page 238 gives all the details.

Restrictions when using EDF

Before you start using EDF, you should be aware of the following points:

- EDF can be used to test only user application programs that use the CICS application programming interface (API).
- EDF can be used only in single-screen mode when running a remote transaction.
- User application programs that are to be debugged using EDF must be compiled with *DEBUG as one of the CICSOPT options on the CRTICSCBL or CRTICSC CL command. If *DEBUG is specified, the translator inserts additional COBOL/400 or ILE C statements used to interface the application program with EDF.

Note that there is a performance advantage in specifying *NODEBUG, and the *NODEBUG option is useful in preventing EXEC CICS commands in well-debugged programs from appearing in EDF.

- EDF runs as a CICS transaction. You invoke it using the CEDF transaction identifier.
- The program you want to test must be defined in the PPT as being executable with EDF. The *CICS for iSeries Administration and Operations Guide* provides information about how to do this.
- Note that you can use EDF only from a 3270, 5250, or APPC terminal that has a screen width of 80 columns or more and a screen depth of 24 lines or more.
- When using CEDF on a 5250 screen, the cursor is not positioned on the first input field, but in the top left hand corner of the screen.
- An APPC session ID cannot be specified as the termid for EDF. If an APPC session ID is entered, an error message is displayed indicating the error.

Where does EDF intercept the program?

When a transaction runs under EDF control, EDF intercepts it at the following points, allowing you to interact with it:

- At **program initiation**, after the EXEC interface block (EIB) has been updated, but before the program is given control. For general-usage programming interface information about EIB values, see Appendix A, “EXEC interface block,” on page 529.
- At the **start of execution of every request** that goes through the CICS control region, EDF displays the command, including keywords, options, and argument values. You can display the information in hexadecimal or character format (switching from one to the other) by pressing the PF2 key. The command is identified by transaction identifier, program name, the hexadecimal offset within the program, and the line number of the command as given in the translator source listing.
- At the **end of the execution of every command** except for EXEC CICS ABEND, EXEC CICS XCTL, and EXEC CICS RETURN commands (although these commands could raise an error condition that EDF displays). EDF intercepts the transaction when it has finished processing the command, but before the EXEC CICS HANDLE CONDITION mechanism is invoked, and before the response trace entry is made.
- At **program termination**. Indication that this point has been reached is returned to the terminal.
- At **normal task termination**. Indication that this point has been reached is returned to the terminal.
- When an **ABEND** occurs and after **abnormal task termination**. EDF displays the values of the fields in the EIB and the abend code.

Note: For a program translated with the option *NODEBUG, this still applies, apart from before and after the execution of each command. For a program with CEDF defined as NO by resource definition online, the program initiation and termination screens are suppressed as well.

What does EDF display?

All EDF displays have the same general format, but the contents depend on the point at which the task was interrupted. The display indicates which of these interception points has been reached and shows information relevant to that point. Figure 45 shows a typical display; it occurred after execution of an EXEC CICS SEND MAP command.

```
TRANSACTION: ACCT PROGRAM: ACCT00 TASK NUMBER: 0236191 DISPLAY: 00
STATUS: COMMAND COMPLETED 1
EXEC CICS SEND MAP
MAP 'ACCTMNU' 2
MAPSET 'ACCTSET'
MAPONLY
ERASE
FREEKB

LINE: 31 EIBFN= X'1804'
RESPONSE: NORMAL EIBRESP= 0
ENTER: CONTINUE 3
PF1 : UNDEFINED PF2 : SWITCH HEX/CHAR PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY 4
PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: UNDEFINED PF12: ABEND USER TASK
```

Note: 1Header 2Body 3Message line 4Menu of options

Figure 45. Typical EDF display

The display consists of a header, a body (the primary display area), a message line, and a menu of functions you can select at this point. If the body does not fit on one screen, EDF creates multiple screens, which you can scroll through using PF7 and PF8. The header, menu and message areas are repeated on each screen.

The header

The header shows:

- The identifier of the transaction being executed
- The name of the program being executed
- The internal task number assigned by CICS to the transaction
- A display number
- STATUS, that is, the reason for the interception by EDF

The body

The body or main part of the display contains the information that varies with the point of intercept.

- At **program initiation**, as shown in Figure 46 on page 232, EDF displays the COMMAREA and the contents of the principal fields in the EIB. For information about these EIB fields, see Appendix A, “EXEC interface block,” on page 529. If there is no COMMAREA value supplied then line 3 on the screen is left blank and EIBCALEN has a value of zero.

```

TRANSACTION: ACCT PROGRAM: ACCT00 TASK NUMBER: 0236191 DISPLAY: 00
STATUS: PROGRAM INITIATION
  COMMAREA =
  EIBTIME = 181447
  EIBDATE = 93252
  EIBTRNID = 'ACCT'
  EIBTASKN = 0236191
  EIBTRMID = 'T1S1'
  EIBCPOSN = 5
  EIBCALEN = 0
  EIBAID = X'7D' AT X'0000001E'
  EIBFN = X'0000' AT X'0000001F'
  EIBRCODE = X'000000000000' AT X'00000021'
  EIBDS = '.....'
  EIBREQID = '.....'
+ EIBRSRCE = '.....'
ENTER: CONTINUE
PF1 : UNDEFINED PF2 : SWITCH HEX/CHAR PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: UNDEFINED PF12: UNDEFINED

```

Figure 46. Typical EDF display at program initiation

- At the **start of execution of a CICS command**, EDF displays the command, including keywords, options, and argument values, as shown in Figure 47. You can display the information in hexadecimal or character form (and switch from one to the other) by pressing PF2. If character format is requested, numeric arguments are shown in signed numeric character format.

```

TRANSACTION: ACCT PROGRAM: ACCT00 TASK NUMBER: 0236191 DISPLAY: 00
STATUS: ABOUT TO START COMMAND
EXEC CICS SEND MAP
  MAP 'ACCTMNU'
  MAPSET 'ACCTSET'
  MAPONLY
  ERASE
  FREEKB
LINE: 31 EIBFN= X'1804'
ENTER: CONTINUE
PF1 : UNDEFINED PF2 : SWITCH HEX/CHAR PF3 : UNDEFINED
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: UNDEFINED PF12: ABEND USER TASK

```

Figure 47. Typical EDF display at start of execution of a CICS command

- At the **end of execution of a command**, EDF provides a display in the same format as at the start of the command. At this point, you can see the effects of executing the command, in the values of the variables returned or changed and in the response code. EDF does not provide this display for the EXEC CICS ABEND, EXEC CICS XCTL, and EXEC CICS RETURN commands (although these commands could raise an error condition that EDF displays). The completion screen corresponding to the **about to start command** screen in Figure 47 is shown in Figure 48 on page 233.

```

TRANSACTION: ACCT PROGRAM: ACCT00 TASK NUMBER: 0236191 DISPLAY: 00
STATUS: COMMAND COMPLETED
EXEC CICS SEND MAP
MAP 'ACCTMNU'
MAPSET 'ACCTSET'
MAPONLY
ERASE
FREEKB

RESPONSE: NORMAL LINE: 31 EIBFN= X'1804'
ENTER: CONTINUE EIBRESP= 0
PF1 : UNDEFINED PF2 : SWITCH HEX/CHAR PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: UNDEFINED PF12: ABEND USER TASK

```

Figure 48. Typical EDF display at completion of a CICS command

For CICS commands, response codes are described both by name (for example, NORMAL or NOTFND) and by the corresponding EIBRESP value in decimal form.

- At **program termination** and **normal task termination**, there is no body information; all the pertinent information is in the header. Figure 49 and Figure 50 show typical screens for program and task termination.

```

TRANSACTION: ACCT PROGRAM: ACCT00 TASK NUMBER: 0236191 DISPLAY: 00
STATUS: PROGRAM TERMINATION
ENTER: CONTINUE
PF1 : UNDEFINED PF2 : SWITCH HEX/CHAR PF3 : UNDEFINED
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : UNDEFINED PF8 : UNDEFINED PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: UNDEFINED PF12: ABEND USER TASK

```

Figure 49. Typical EDF display at program termination

```

TRANSACTION: ACCT PROGRAM: TASK NUMBER: 0036032 DISPLAY: 00
STATUS: NORMAL TASK TERMINATION
TO CONTINUE EDF SESSION REPLY YES REPLY: NO
ENTER: CONTINUE
PF1 : UNDEFINED PF2 : UNDEFINED PF3 : END EDF SESSION
PF4 : UNDEFINED PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : UNDEFINED PF8 : UNDEFINED PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: UNDEFINED PF12: UNDEFINED

```

Figure 50. Typical EDF display at task termination

- When an **abend** or **abnormal task termination** occurs, EDF displays the screens shown in Figure 51 on page 234 and Figure 52 on page 234.

```

TRANSACTION: ACCT PROGRAM: ACCT00 TASK NUMBER: 0236191 DISPLAY: 00
STATUS: AN ABEND HAS OCCURRED
  COMMAREA =
  EIBTIME = 181447
  EIBDATE = 93252
  EIBTRNID = 'ACCT'
  EIBTASKN = 0236191
  EIBTRMID = 'T1S1'
  EIBCPOSN = 5
  EIBCALEN = 0
  EIBAID = X'7D' RETURN AT X'0000001E'
  EIBFN = X'0E08' RETURN AT X'0000001F'
  EIBRCODE = X'000000000000' AT X'00000021'
  EIBDS = '.....'
  EIBREQID = '.....'
+ EIBRSRCE = '.....'
  ABEND: ASRB
ENTER: CONTINUE
PF1 : UNDEFINED PF2 : SWITCH HEX/CHAR PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: UNDEFINED PF12: UNDEFINED

```

Figure 51. Typical EDF display when an abend occurs

```

TRANSACTION: ACCT PROGRAM: TASK NUMBER: 0236191 DISPLAY: 00
STATUS: ABNORMAL TASK TERMINATION
  COMMAREA =
  EIBTIME = 181447
  EIBDATE = 93252
  EIBTRNID = 'ACCT'
  EIBTASKN = 0236191
  EIBTRMID = 'T1S1'
  EIBCPOSN = 5
  EIBCALEN = 0
  EIBAID = X'7D' RETURN AT X'0000001E'
  EIBFN = X'0E08' RETURN AT X'0000001F'
  EIBRCODE = X'000000000000' AT X'00000021'
  EIBDS = '.....'
  EIBREQID = '.....'
+ EIBRSRCE = '.....'
  ABEND: ASRB
TO CONTINUE EDF SESSION REPLY YES REPLY: NO
ENTER: CONTINUE
PF1 : UNDEFINED PF2 : SWITCH HEX/CHAR PF3 : END EDF SESSION
PF4 : UNDEFINED PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: UNDEFINED PF12: UNDEFINED

```

Figure 52. Typical EDF display at abnormal task termination

The body displays the COMMAREA and the values of the fields in the EIB as well as the abend code.

How you can intervene in program execution

The power of EDF lies in what you can do at each of the intercept points. For example, you can:

- Change the argument values before a command is executed. For CICS commands you can't change the actual command, or add or delete options, but you can change the value associated with any option. You can also suppress execution of the command entirely using NOOP. See page 241 for further details.
- Change the results of a command, either by changing the argument values returned by execution or by modifying the response code. This allows you to test branches of the program that are hard to reach using ordinary test data (for

example, what happens on an input/output error). It also allows you to bypass the effects of an error to check whether this eliminates a problem.

- Display any other location in the CICS region.
- Change the working storage of the program and most fields in the EIB. EDF stops your task from interfering with other tasks by preventing you from changing other areas of storage.
- Display the contents of temporary storage and transient data queues.
- Suppress EDF displays until one or more of a set of specific conditions is fulfilled. This speeds up testing.
- Retrieve up to 10 previous EDF displays or saved screens.
- Switch off EDF mode and run the application normally.
- Abend the task.

The first two types of changes are made by overtyping values in the body of the command displays. "Overtyping to make changes" on page 240 tells you how to do this. You use the function keys in the menu for the others; "EDF menu functions" tells you exactly what you can do and how to go about it.

EDF menu functions

The function keys that you can use at any given time are displayed in a menu at the bottom of every EDF display (see Figure 45 on page 231). The function of the ENTER key for that display is also shown. Functions that apply to all displays are always assigned to the same key, but definitions of some keys depend on the display and the intercept point. To select an option, press the indicated function key. Where a terminal has 24 function keys, EDF treats PF13 through PF24 as duplicates of PF1 through PF12 respectively. If your terminal has no PF keys, place the cursor under the option you want and press the ENTER key.

ABEND USER TASK

terminates the task being monitored. EDF asks you to confirm this action by displaying the message "ENTER ABEND CODE AND REQUEST ABEND AGAIN". After entering the code at the position indicated by the cursor, the user must request this function again to abend the task with a transaction dump identified by the specified code.

Abend codes beginning with the character A are reserved for use by CICS. Using a CICS abend code may cause unpredictable results.

You cannot use this function if an abend is already in progress or the task is terminating.

BROWSE TEMP STORAGE

produces a display of the temporary storage queue CEBRxxxx, where xxxx is the terminal identifier of the terminal running EDF. This function is only available from the working storage (PF5) screen. You can then use CEBR commands, discussed in "The CEBR commands" on page 246, to display or modify temporary storage queues and to read or write transient data queues.

CONTINUE

redispays the current screen if you have made any changes, incorporating the changes. If you had not made changes, CONTINUE causes the transaction under test to resume execution up to the next intercept point. To continue, press ENTER.

CURRENT DISPLAY

redispays the current screen if you have made any changes, with the

changes incorporated. If you have not made changes, it causes EDF to display the command screen for the last intercept point. To execute this function, press ENTER from the appropriate screen.

EIB DISPLAY

displays the contents of the EIB. This function is only available from the working-storage screen (PF5). See Figure 46 on page 232 for an example of an EIB display. See Appendix A, "EXEC interface block," on page 529 for information on EIB fields. If COMMAREA exists, EDF also displays its address and one line of data in the dump format.

END EDF SESSION

ends the EDF control of the transaction. The transaction continues running from that point but no longer runs in EDF mode.

NEXT DISPLAY

is the reverse of PREVIOUS DISPLAY. When you have returned to a previous display, this option causes the next one forward to be displayed and the display number to increase by one.

PREVIOUS DISPLAY

causes the previous display to be sent to the screen. This will be the previous command display, unless you saved other displays. The number of the display from the current intercept point is always 00. As you request previous displays, the display number decreases by 1 to -01 for the first previous display, -02 for the one before that, and so on, down to the oldest display, -10. When no more previous screens are available, the PREVIOUS option disappears from the menu, and the corresponding function key becomes inoperative.

REMEMBER DISPLAY

places a display that would not usually be kept in memory, such as an EIB display, in the EDF memory. (EDF automatically saves the displays at the start and completion of each command.) The memory can hold up to 10 displays. The displays are numbered in reverse chronological order (that is, -10 is the oldest display, and -01 is the newest). All pages associated with the display are kept in memory and can be scrolled when recalled. Note, however, that if you save a working-storage display, only the screen on view is saved.

SCROLL BACK

applies to an EIB or command display that does not all fit on one screen. When the screen on view is not the first one of the display, and there is a plus sign (+) before the first option or field, then you can view previous screens in the display by selecting SCROLL BACK.

SCROLL FORWARD

applies to an EIB or command display that does not all fit on one screen. When this happens, a plus sign (+) appears after the last option or field in the display, to show that there are more screens. See Figure 46 on page 232 for an example. Using SCROLL FORWARD brings up the next screen in the display.

SCROLL BACK FULL

has the same function for displays of working storage as the SCROLL BACK option for EIB displays. SCROLL BACK FULL gives a working-storage display one full screen backward, showing addresses lower in storage than those on the current screen.

SCROLL FORWARD FULL

has the same function for displays of working storage as the SCROLL FORWARD option for EIB displays. SCROLL FORWARD FULL gives a working-storage display one full screen forward, showing addresses higher in storage than those on the current screen.

SCROLL BACK HALF

is similar to SCROLL BACK FULL, except that the display of working storage is reversed by only half a screen.

SCROLL FORWARD HALF

is similar to SCROLL FORWARD FULL, except that the display of working storage is advanced by only half a screen.

STOP CONDITIONS

produces the menu screen shown in Figure 53. You use this screen to tell EDF when to resume its displays after you have pressed the SUPPRESS DISPLAYS key. You can use STOP CONDITIONS and SUPPRESS DISPLAYS together to cut down on the interaction between you and EDF when you are checking a program that you know is partly working.

```
TRANSACTION: ACCT PROGRAM: ACCT00 TASK NUMBER: 0236191 DISPLAY: 00
DISPLAY ON CONDITION:
COMMAND: EXEC CICS
LINE NUMBER:
CICS EXCEPTIONAL CONDITION:
ANY CICS EXCEPTIONAL CONDITION YES
TRANSACTION ABEND YES
NORMAL TASK TERMINATION YES
ABNORMAL TASK TERMINATION YES
ENTER: CURRENT DISPLAY
PF1 : UNDEFINED PF2 : UNDEFINED PF3 : UNDEFINED
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : UNDEFINED PF8 : UNDEFINED PF9 : UNDEFINED
PF10: UNDEFINED PF11: UNDEFINED PF12: REMEMBER DISPLAY
```

Figure 53. Typical EDF display for STOP CONDITIONS

You can specify any or all of these events as STOP CONDITIONS:

- A specific type of function and option, such as EXEC CICS READNEXT file or EXEC CICS ENQ resource, is encountered.
- The command at a specific offset or on a specific line number.
- A specific exception condition occurs.
- Any exception condition occurs for which the system action is to raise ERROR; for example, INVREQ or NOTFND.
- An abend occurs.
- The task ends normally.
- The task ends abnormally.

The line number, which is available on the source listing if the program has been translated, must be specified as it appears on the listing, and must specify the line on which a CICS command starts. The correct line number can be determined easily from the translator output listing.

SUPPRESS DISPLAYS

suppresses all EDF displays until one of the specified STOP CONDITIONS occurs.

SWITCH HEX/CHAR

switches displays between character and hexadecimal form. The switch applies only to the command display, and has no effect on previously remembered displays, STOP CONDITIONS displays, or working-storage displays.

UNDEFINED

means that the indicated function key is not defined for the current display at the current intercept point.

USER DISPLAY

causes EDF to display what would be on the screen if the transaction was not running in EDF mode. (You can use it only for single terminal checkout.) To return to EDF after using this key, press the ENTER key.

WORKING STORAGE

allows you to see the contents of the working-storage area in your program, or of any other address in the CICS region.

The working-storage contents are displayed in a form similar to that of a dump listing, that is, in both hexadecimal and character representation. The address of working storage is displayed at the top of the screen. You can browse through the entire area using the scroll commands, or you can simply enter a new address at the top of the screen. This address can be anywhere within the working storage area in your program, or TS and TD queues. When this key is used, two additional scrolling keys are provided, and other PF keys allow the EIB to be displayed.

Working storage can be changed at the screen; either the hexadecimal section or the character section can be used. Also, the ADDRESS field at the head of the display can be overtyped with a hexadecimal address; storage starting at that address is then displayed when ENTER is pressed. Further information on the use of overtyping is given in "Overtyping to make changes" on page 240.

If the initial part of a working-storage display line is blank, the blank portion is not part of working storage. This can occur because the display is doubleword aligned.

How to use EDF

You can run EDF on the same terminal as the transaction to be tested (this is called "single-screen mode"), or on a different terminal ("dual-screen mode"). Generally, you can use whichever method you prefer, but there are a few situations in which one or the other is required. You must use single-screen mode for remote transactions. See "Restrictions when using EDF" on page 229 for other conditions which affect your choice.

Using EDF in single-screen mode

When you use EDF with just one terminal, the EDF inputs and outputs are interleaved with those from the transaction. This sounds complicated, but works quite easily in practice. The only noticeable peculiarity is that when an EXEC CICS SEND command is followed by an EXEC CICS RECEIVE command, the display sent by the EXEC CICS SEND command appears twice: once when the EXEC CICS SEND is executed, and again when the EXEC CICS RECEIVE is executed. It is not necessary to respond to the first display, but if you do, EDF preserves anything that was entered from the first display to the second.

You start EDF by:

- Entering transaction code CEDF from a cleared screen, or
- Pressing the appropriate PF key (if one has been defined for EDF)

Next, you start the transaction to be tested by:

1. Pressing the CLEAR key to clear the screen
2. Entering the transaction code of the transaction you want to test

When both EDF and the user transaction are sharing the same terminal, EDF restores the user transaction display at the following times:

- When the transaction requires input from the operator
- When you change the transaction display
- At the end of the transaction
- When you suppress the EDF displays
- When you request USER DISPLAY

To enable restoration, user displays are remembered at the following times:

1. At start of task, before the first EDF screen for the task is displayed
2. Before the next EDF screen is displayed, if the user display has been changed
3. On leaving SCREEN SUPPRESS mode

If a program has been translated with option *NODEBUG, or has NO specified for CEDF in its resource definition, it is not possible for EDF to ascertain when the user display is being changed. This means that, unless either situation 1 or 3 also apply, the next EDF screen to be displayed overwrites any user display sent by this program without saving it first, so that it cannot be later restored.

When EDF restores the transaction display, it does not sound the alarm or affect the keyboard in the same way as the user transaction. The effect of the user transaction options is seen when the EXEC CICS SEND command is processed, but not when the screen is restored.

When EDF restores the transaction display on a device that uses color, programmed symbols, or extended highlighting, these attributes are no longer present and the display is monochrome without the programmed symbols or extended highlighting. Also, if the inbound reply mode in the application program is set to "character" to enable the attribute-setting keys, EDF resets this mode, causing these keys to be disabled. If these changes will prevent your transaction from executing properly, you should test in a dual-screen mode.

If you end your EDF session part way through the transaction, EDF restores the screen with the keyboard locked if the most recent EXEC CICS RECEIVE command has not been followed by an EXEC CICS SEND command; otherwise, the keyboard is unlocked.

Checking pseudoconversational programs

EDF makes a special provision for testing pseudoconversational transactions from a single terminal. If the terminal came out of EDF mode between the several tasks that make up a pseudoconversational transaction, it would be very hard to do any debugging after the first task. So, when a task terminates, EDF asks the operator whether EDF mode is to continue to the next task. If you are debugging a pseudoconversational task, press enter as the default is "yes". If you have finished, reply "no".

Using EDF in dual-screen mode

In dual-screen mode, you use one terminal for EDF interaction and another for sending input to, and receiving output from, the transaction under test.

You start by entering, at the EDF terminal, the transaction:

```
CEDF tttt
```

where tttt is the name of the terminal on which the transaction is to be tested. Terminal identifiers are assigned by the system administrator using the CICSDEV parameter of the ADDCICSTCT CL command, or alternatively for autoinstall terminals, the terminal name is generated by the autoinstall process. The terminal which you are using for the EDF interaction must be in transceive (ATI/TTI) status and be able to send and receive data. This is the most common status for display terminals, but you can find out by asking your system programmer to check its status, or you can use CEMT (or your site replacement):

```
CEMT INQ TERMINAL(tttt)
```

and change it if it is not already ATI/TTI:

```
CEMT SET TERMINAL(tttt) ATI TTI
```

Enter the transaction to be tested on this second terminal.

You can also use EDF in dual-screen mode to monitor a transaction that is already running. If, for example, you believe a transaction at a specific terminal to be looping, you can go to another terminal and enter a CEDF transaction naming the terminal at which this transaction is running. EDF picks up control at the next EXEC CICS command executed, and then you can observe the sequence of commands that are causing the loop, assuming that at least one EXEC CICS command is executed. (This applies to a transaction running in the same CICS control region.)

Stopping EDF

If you want to end EDF control of a terminal, the method you use depends on where you are in the testing. If the transaction under test is still executing and you want it to continue, but without EDF, press the END EDF SESSION function key. If you have reached the task termination intercept, EDF asks if you want to continue. If you don't, overwrite the reply as NO (YES is the default). If no transaction is executing at the terminal, clear the screen and enter:

```
CEDF OFF
```

Overtyping to make changes

Most of the changes you make with EDF involve changing information in memory. You do this simply by typing over the information shown on the screen with the information you want used instead. You can change any area where the cursor stops when you use the tab keys, except for the menu area at the bottom.

Note: If you are using 5250 terminals you may be able to tab to and type over any area of the screen. You should not attempt to do so.

When you change the screen, you must observe the following rules:

- On CICS command screens, any argument value can be overtyped, but not the keyword of the argument. An optional argument cannot be removed, nor can an option be added or deleted.

- When you change an argument in the command display (as opposed to the working storage screen), you can change only the part shown on the display. If you attempt to overwrite beyond the value displayed, the changes are not made and no diagnostic message is generated. If the argument is so long that only part of it appears on the screen, you should change the area in working storage to which the argument points. (To determine the address, display the argument in hexadecimal format; the address of the argument location also appears.)
- In character format, numeric values always have a sign field, which can be overtyped with a minus or a blank only.
- When an argument is to be displayed in character format, some of the characters may not be displayable (including lowercase characters). EDF replaces each nondisplayable character with a period. When overtyping a period, you must be aware that the storage may in fact contain a character other than a period. You should not overwrite any character with a period; if you do, the change is ignored and no diagnostic message is issued. If you need to overwrite a character with a period, you can do so by switching the display to hexadecimal format, using PF2, and overtyping with X'4B'.
- When storage is displayed in both character and hexadecimal format and changes are made to both, the value of the hexadecimal field takes precedence should the changes conflict; no diagnostic message is issued.
 - The arguments for some commands, such as EXEC CICS HANDLE CONDITION, are program labels rather than numeric or character data. If no label value is specified on an EXEC CICS HANDLE CONDITION command, EDF displays the condition name alone without the parentheses.
 - The response field can be overtyped with the name of any exception condition, including ERROR, that can occur for the current function, or with the word NORMAL. The effect when EDF continues will be that the program takes whatever action has been prescribed for the specified response.
 - If uppercase translation is not specified for the terminal you are using you must take care to always enter uppercase characters.
 - Any command can be overtyped with NOOP or NOP before processing; this suppresses processing of the command. Use of the ERASE EOF key, or overtyping with blanks, gives the same effect. When the screen is redisplayed with NOP, the original verb line can be restored by erasing the whole verb line with the ERASE EOF key and pressing the ENTER key.

When you overwrite a field representing a data area in your program, the change is made directly in application program storage and is permanent. However, if you change a field that represents a constant (a program literal), program storage is not changed, because this may affect other parts of the program that use the same constant or other tasks using the program. The command is executed with the changed data, but when the command is displayed after processing, the original argument values reappear. For example, suppose you are testing a program containing a command coded:

```
EXEC CICS SEND MAP('MENU') END-EXEC.
```

If you change the name MENU to MENU2 under EDF before executing the command, the map actually used is MENU2, but the map displayed on the response will be MENU. (You can use the “previous display” key to verify the map name you used.) If you process the same command more than once, you must enter this type of change each time.

EDF responses

The response of EDF to any keyboard entry follows the rules listed below, in the order shown:

1. If the CLEAR key is used, EDF redisplay the screen with any changes ignored.
2. If invalid changes are made, EDF accepts any valid changes and redisplay the screen with a diagnostic message.
3. If the display number is changed, EDF accepts any other changes and shows the requested display.
4. If a PF key is used, EDF accepts any changes and performs the action requested by the PF key. Pressing ENTER with the cursor under a PF key definition in the menu at the bottom of the screen is the same as pressing a PF key.
5. If the ENTER key is pressed and the screen has been modified (other than the REPLY field), EDF redisplay the screen with changes included.
6. If the ENTER key is pressed and the screen has not been modified (other than the REPLY field), the effect differs according to the meaning of the ENTER key. If the ENTER key means CONTINUE, the user transaction continues to execute. If it means CURRENT DISPLAY, EDF redisplay the current status display.

Chapter 28. Temporary storage browse (CEBR)

You can use the browse transaction (CEBR) to browse temporary storage queues and delete them. You can also use the CEBR transaction to transfer the contents of a transient data queue to temporary storage in order to look at them, and to reestablish the transient data queue when you have finished. The CEBR commands that perform these transfers allow you to add records to a transient data queue and remove all records from a transient data queue. See “The CEBR commands” on page 246 and “The CEBR options on function keys” on page 245 for more information about their use.

How to use the CEBR transaction

You start the CEBR transaction by entering the transaction identifier CEBR, followed by the name of the queue you want to browse. For example, to display the temporary storage queue named CEBRS209, you type:

```
CEBR CEBRS209
```

and press ENTER. CICS responds with a display of the queue, for example, as shown in Figure 54:

```
CEBR          TS QUEUE CEBRS209 RECORD    1 OF    3 COL    1 OF 22
ENTER COMMAND ===> _____
***** TOP OF QUEUE *****
00001 000055001234000001S209
00002 000056003456000002S209
00003 000102000564000001S209
***** BOTTOM OF QUEUE *****
PF1 : HELP          PF2 : SWITCH HEX/CHAR    PF3 : SESSION ENDED
PF4 : VIEW TOP      PF5 : VIEW BOTTOM       PF6 : REPEAT LAST FIND
PF7 : SCROLL BACK HALF PF8 : SCROLL FORWARD HALF PF9 : UNDEFINED
PF10: SCROLL BACK FULL PF11: SCROLL FORWARD FULL PF12: UNDEFINED
```

Figure 54. Typical CEBR display of temporary storage queue contents

Alternatively, you can start the CEBR transaction from the CEDF transaction. You do this by pressing PF5 from the initial CEDF screen (see Figure 45 on page 231) which takes you to the working-storage screen, and then pressing PF2 from that screen to browse temporary storage (that is, invoke the CEBR transaction). The CEBR transaction responds by displaying the temporary storage queue whose name consists of the four letters CEBR followed by the four letters of your terminal identifier. (CICS uses this same default queue name if you invoke the CEBR transaction directly and do not supply a queue name.) The result of invoking the CEBR transaction without a queue name or from an EDF session at terminal S21A is shown in Figure 55. If you enter the CEBR transaction from the CEDF transaction, you return to the EDF panel when you press PF3 from the CEBR screen.

```

CEBR          TS QUEUE  CEBRS21A RECORD    1 OF    0 COL    1 OF    0
ENTER COMMAND ==>
***** TOP OF QUEUE *****
***** BOTTOM OF QUEUE *****

TEMPORARY STORAGE QUEUE CEBRS21A does not exist.
PF1 : HELP          PF2 : SWITCH HEX/CHAR    PF3 : SESSION ENDED
PF4 : VIEW TOP      PF5 : VIEW BOTTOM      PF6 : REPEAT LAST FIND
PF7 : SCROLL BACK HALF PF8 : SCROLL FORWARD HALF PF9 : UNDEFINED
PF10: SCROLL BACK FULL PF11: SCROLL FORWARD FULL PF12: UNDEFINED

```

Note: 1 Header 2 Command area 3 Body 4 Message line 5 Menu of options

Figure 55. Typical CEBR display of default temporary storage queue

What does the CEBR transaction display?

As shown in Figure 55, a CEBR transaction display consists of a header, a command area, a body (the primary display area), a message line, and a menu of functions you can select at this point.

The header

The header shows:

- The transaction being run, that is, CEBR.
- The identifier of the temporary storage queue (CEBRS209 in Figure 54 on page 243 and CEBRS21A in Figure 55). You can overwrite this field in the header if you want to switch the screen to another queue.
- The number of the highlighted record.
- The number of records in the queue (three in CEBRS209 and none in CEBRS21A).
- The position in each record at which the screen starts (position 1 in both cases) and the length of the longest record (22 for queue CEBRS209 and zero for queue CEBRS21A).

The command area

The command area is where you enter commands that control what is to be displayed and what function is to be performed. “The CEBR commands” on page 246 describes these commands. You can also modify the screen with function keys shown in the menu of options at the bottom of the screen. The function keys are explained in “The CEBR options on function keys” on page 245.

The body

The body is where the queue records are shown. Each line of the screen corresponds to one queue record. If a record is too long for the line, it is truncated. You can change the portion of the record that is displayed, however, so that you can see an entire record on successive screens. If the queue contains more records than will fit on the screen, you can page forward and backward through them, or specify at what record to start the display, so that you can see all the records you want.

The message line

CEBR uses the message line between the body and menu to display messages to the user, such as the "TEMPORARY...exist" message in Figure 55 on page 244.

The CEBR options on function keys

The function keys that you can use at any time are displayed at the bottom of every CEBR transaction screen. The keys have the same meaning on all screens. If your terminal does not have PF keys, you can simulate their use by placing the cursor under the description and pressing ENTER. Where a terminal has 24 function keys, The CEBR transaction treats PF13 through PF24 as duplicates of PF1 through PF12 respectively.

PF1 HELP

Displays a help screen that lists all the commands you can use when the CEBR transaction is running. You can return to the main screen by pressing ENTER.

PF2 SWITCH HEX/CHAR

Switches the screen from character to hexadecimal format, and back again.

PF3 SESSION ENDED

Terminates the CEBR transaction. If you entered the CEBR transaction directly, it frees up your terminal for the next transaction. If you entered from an EDF session, it returns you to the working-storage screen from which you entered.

PF4 VIEW TOP

Displays the first records in the queue and has the same effect as the TOP command.

PF5 VIEW BOTTOM

Displays the last records in the queue and has the same effect as the BOTTOM command.

PF6 REPEAT LAST FIND

Repeats the previous FIND command.

PF7 SCROLL BACK HALF

Moves the display backward by one-half the number of records that fit on the screen, so that the records on the top half of the screen move to the bottom half.

PF8 SCROLL FORWARD HALF

Advances the display by one-half the number of records that fit on the screen, so that the records on the bottom half of the screen move to the top half.

PF9 VIEW RIGHT (or VIEW LEFT)

Changes the screen to show the columns immediately after (to the right of) or before (to the left of) the columns currently on display. The key is not defined if the entire record fits on one line of the screen. It moves you to the right until the end of the record is reached, and then reverses to move left back to the beginning of the record. You can also use the COLUMN command to change the column at which the display begins.

PF10 SCROLL BACK FULL

Moves the screen backward by the number of records that fit on the screen, to show the records immediately before those currently on display.

PF11 SCROLL FORWARD FULL

Advances the screen by the number of records that will fit on the screen, to show the records immediately after those currently on display.

The CEBR commands

Here is a list of the CEBR commands that you can use to view and manipulate the records in the temporary storage queue.

BOTTOM

(Abbreviation: B)

Shows the last records in the temporary storage queue (as many as fill up the body of the screen, with the last record on the last line).

COLUMN nnnn

(Abbreviation: C nnnn)

Displays the records starting at character position (column) nnnn of each record. The default starting position, assumed when you initiate the CEBR transaction, is the first character in the record.

FIND /string

(Abbreviation: F /string)

Finds the next occurrence of the specified string. The search starts in the record *after* the **current record**. The current record is the one that is highlighted. In the initial display of a queue, the current record is set to one, and therefore the search begins at record two.

If the string is found, the record containing the string becomes the highlighted line, and the display is changed to show this record on the second line. If you cannot see the search string after a successful FIND, it is in columns of the record beyond those on display; use the scroll key or the COLUMN command to shift the display right or left to show the string.

For example:

```
FIND /05-02-93
```

will locate the next occurrence of the string "05-02-93". The / character is a delimiter. It does not have to be /, but it must not be a character that appears in the search argument. For example, if the string you were looking for was "05/02/93" instead of "05-02-93", you could not use the following:

```
FIND /05/02/93
```

There is a slash in the search string. The following examples would work:

```
FIND X05/02/93    or    FIND S05/07/93
```

Any delimiter except a / or one of the digits in the string works. If there are any spaces in the search string, you must repeat the delimiter at the end of the string. For example:

```
FIND /CLARE JACKSON/
```

The search string is not case-sensitive.

When you have entered a FIND command, you can repeat it (that is, find the next occurrence of the string) by pressing PF6.

GET xxxx

(Abbreviation: G xxxx)

Transfers the named transient data queue to the end of the temporary storage queue currently on display. This enables you to browse the contents of the queue. xxxx must be either the name of an intrapartition transient data queue, or the name of an extrapartition transient data queue that has been opened for input. See “Using the CEBR transaction with transient data” on page 248 for more information about browsing transient data queues.

LINE nnnn

(Abbreviation: L nnnn)

Starts the body of the screen at the queue record one prior to nnnn, and sets the current line to nnnn. (This arrangement causes a subsequent FIND command to start the search after record nnnn.)

PURGE

(Abbreviation: PUR)

Deletes the queue being browsed.

Do not use PURGE to delete the contents of an internally generated queue, such as a BMS logical message.

Note: If you purge a recoverable temporary storage queue, no other task can update that queue (add a record, change a record, or purge) until your task ends. You should not attempt to access a queue you have purged in the same CEBR session either; doing so will cause abend ATSP.

PUT xxxx

(Abbreviation: P xxxx)

Copies the temporary storage queue that is being browsed to the named transient data queue. xxxx must be either the name of an intrapartition transient data queue, or the name of an extrapartition transient data queue that has been opened for output. See “Using the CEBR transaction with transient data” on page 248 for more about creating or restoring a transient data queue.

QUEUE xxxxxxxx

(Abbreviation: Q xxxxxxxx)

Changes the name of the queue you are browsing. The value that you specify can be in character format (for example, QUEUE ABCD) or in hexadecimal format (for example, QUEUE X'C1C2C3C4'). The CEBR transaction responds by displaying the data that is in the named queue.

You can also change the queue name by overtyping the current value in the header.

TERMINAL xxxx

(Abbreviation: TE xxxx)

Changes the name of the queue you are browsing, but is tailored to applications that use the convention of naming temporary storage queues that are associated with a terminal by a constant in the first four characters and the terminal name in the last four. The new queue name is formed from the first four characters of the current queue name, followed by xxxx.

TOP (Abbreviation: T)

Causes the CEBR transaction to start the display at the first record in the queue.

Using the CEBR transaction with transient data

The GET command reads each record in the transient data queue that you specify and writes it at the end of the temporary storage queue you are browsing, until the transient data queue is empty. You can then view the records that were in the transient data queue. When you have finished your inspection, you can copy the temporary storage queue back to the transient data queue (using the PUT command). This usually leaves the transient data queue as you found it, but not always. Here are some points you need to be aware of when using the GET and PUT commands:

- If you want to restore the transient data queue unchanged after you have browsed it, make sure that the temporary storage queue on display at the time of the GET command is empty. Otherwise, the existing temporary storage records will be copied to the transient data queue when the subsequent PUT command is issued.
- After you get a transient data queue and before you put it back, other tasks may write to that transient data queue. When you issue your PUT command, the records in the temporary storage queue will be copied **after** the new records, so that the records in the queue are no longer in the order in which they were originally created. Some applications depend on sequential processing of the records in a queue.
- After you get a **recoverable** transient data queue, no other task can access that queue until your transaction ends. If you entered the CEBR transaction from the CEDF transaction, the CEDF transaction must end, although you can respond “yes” to the “continue” question if you are debugging a pseudoconversational sequence of transactions. If you invoked the CEBR transaction directly, you must end it.
- Likewise, after you issue a PUT command to a recoverable transient data queue, no other task can access that queue until your transaction ends.

The GET and PUT commands do not need to be used as a pair. You can add to a transient data queue from a temporary storage queue with a PUT command at any time. If you are debugging code that reads a transient data queue, you can create a queue in temporary storage (with the CECI transaction, or the CEBR GET command, or by program) and then refresh the transient data queue as many times as you like from temporary storage. Similarly, you can empty a transient data queue by using a GET command without a corresponding PUT command.

Security considerations

Some installations restrict the use of the CEBR transaction, particularly in production systems, to prevent modifications that were not intended or not authorized. Installations may also secure individual resources, including the recoverable and non-recoverable control-region data files used for temporary storage and transient data queues. If CEBR reports that it is unable to locate a queue or other resource, the job log may contain security-check information to help you with problem determination.

Chapter 29. Command-level interpreter (CECI)

You can use the command-level interpreter (CECI) transaction to check the syntax of CICS commands and process these commands interactively on a 3270 screen. CECI allows you to follow through most of the commands to execution and display the results. It also provides you with a reference to the syntax of the whole of the CICS command-level application programming and system programming interface.

CECI interacts with your test system to allow you to create or delete test data, temporary storage queues, or to deliberately introduce wrong data to test out error logic. You can also use CECI to repair corrupted database records on your production system.

How to use CECI

You start the command-level interpreter by entering either of two transaction identifiers, CECS or CECI, followed by the name of the command you want to test. You can list command options too, although you can also do this later. For example:

```
CECS READ FILE('FILEA')
```

or

```
CECI READ FILE('FILEA')
```

CICS responds with a display of the command and its associated functions, options, and arguments, as shown in Figure 56 on page 250. If you leave out the command, CECI provides a list of possible commands to get you started. You can use any of the commands described in Part 7, “Programming reference,” on page 263..

```

READ FILE('FILEA')
STATUS:  COMMAND SYNTAX CHECK          NAME=
EXEC CICS READ
  File()
  Into() |
  SEt()
  < Length() >
  RIDfld()
  < Keylength()
    < GEneric > |
    RBa |
    RRn >
  < SYsid() >
  < GTeq |
  Equal >
  < Update >
S RIDFLD must be specified

PF 1 HELP 2 HEX 3 END 4 EIB 5 VAR 6 USER 7 SBH 8 SFH 9 MSG 10 SB 11 SF

```

Note: 1 Command line 2 Status line 3 Body 4 Message line 5 Menu of options

Figure 56. Typical CECI display for command syntax check

If you use the transaction code CECS, the interpreter simply checks your command for correct syntax. If you use CECI, you have the option of executing your command once the syntax is correct. (CICS uses two transaction identifiers to allow different security to be assigned to syntax checking and execution.)

What does CECI display?

All CECI screens have the same basic layout. As shown in Figure 56, CECI displays consist of a command input line, a status line, the body or main part of the screen, a message line, and a menu of functions you can select at this point.

The command line

The command line is the first line of the screen. You enter the command you want to process or whose syntax you want to check here. This can be the full or abbreviated syntax. The rules for entering and abbreviating the command are:

- The keywords EXEC CICS are optional.
- The options of a command can be abbreviated to the number of characters sufficient to make them unique. Valid abbreviations are shown in uppercase characters in syntax displays in the body of the screen.
- The quotation marks around character strings are optional, and all strings of characters are treated as character-string constants unless they are preceded by an ampersand (&), in which case they are treated as variables.
- Options associated with the EXEC CICS ASSIGN or EXEC CICS INQUIRE commands that receive a value from CICS when the command is processed are called **receivers**, and need not be specified. The value received from CICS is included in the syntax display, and stored in the variable if one has been specified, after the command has been processed.

The following example shows the abbreviated form of a command. The file control command:

```
EXEC CICS READ FILE('FILEA')
        RIDFLD('009000') INTO(&REC)
```

can be entered on the command input line, as:

```
READ FIL(FILEA) RID(009000)
```

or at a minimum, as:

```
READ F(FILEA) RI(009000)
```

In the first form, the INTO specification creates a variable, &REC, into which the data is to be read. However, INTO is a receiver (as defined above) and you can omit it. When you do, CICS creates a variable for you automatically.

The status line

As you go through the process of interpreting a command, CECI presents a sequence of displays. The format of the body of the screen is essentially the same for all; it shows the syntax of the command and the option values selected. The status line on these screens tells you where you are in the processing of the command, and is one of:

- COMMAND SYNTAX CHECK
- ABOUT TO START COMMAND
- COMMAND COMPLETED
- COMMAND NOT EXECUTED

From any of these screens, you can select additional displays. When you do, the body of the screen shows the information requested, and the status line identifies the display, which may be any of:

- EXPANDED AREA
- VARIABLES
- EXEC INTERFACE BLOCK
- SYNTAX MESSAGES

These screens are described in “Additional displays” on page 255. You can request them at any time during processing and then return to the command interpretation sequence.

There is also one input field in the status line called NAME=. This field is used to create and name variables, as explained in “Variables” on page 255 and “Saving commands” on page 260.

Command syntax check

When the status line shows **command syntax check** (as shown in Figure 56 on page 250), it indicates that the command entered on the command input line has been syntax checked but is not about to be processed. This is always the status if you enter CECS or if you precede your command with a question mark. It is also the status when the syntax check of the command gives severe error messages.

In addition, you will get this status if you attempt to execute one of the commands that the interpreter cannot execute. Although any command can be syntax-checked, using either CECS or CECI, the interpreter cannot process the following commands any further:

```
HANDLE ABEND
HANDLE AID
HANDLE CONDITION
IGNORE CONDITION
PUSH HANDLE
POP HANDLE
GETMAIN
FREEMAIN
WAIT EVENT
FREE
SEND LAST
```

About to start command

This display (as shown in Figure 57) appears when none of the reasons for stopping at **command syntax check** applies.

```
READ FILE('FILEA') RIDFLD('009000')
STATUS: ABOUT TO START COMMAND                                NAME=
EXEC CICS READ
  File()                FILEA
  Into() |
  SET()
  < Length() >          +32767
  RIDfld()              009000
  < Keylength()
    < GEneric > |
  RBa |
  RRn >
  < SYsid() >
  < GTeq |
  Equal >
  < Update >
PF 1 HELP 2 HEX 3 END 4 EIB 5 VAR 6 USER 7 SBH 8 SFH 9 MSG 10 SB 11 SF
```

Figure 57. Typical CECI display for about to start command

If you press the ENTER key at this point without changing the screen, CECI will execute the command. You can still modify it at this point, however. If you do, CECI ignores the previous command and processes the new one from scratch. This means that the next screen displayed will be **command syntax check** if the command cannot be executed or else **about to start command** if the command is correct.

Command completed

This display (as shown in Figure 58 on page 253) appears after the interpreter has executed a command, in response to the ENTER key from an unmodified **about to start command** screen.

```

READ FILE('FILEA') RIDFLD('009000') INTO(&IN)
STATUS:  COMMAND COMPLETED                                NAME=
EXEC CICS READ
  File()                               FILEA
  Into() |
  SEt()
  < Length() >                          +00100
  RIDfld()                               009000
  < Keylength()
  < GEneric > |
  RBa |
  RRn >
  < SYsid() >
  < GTeq |
  Equal >
  < Update >
RESPONSE: FILENOTFOUND                                EIBRESP = +00000012
PF 1 HELP 2 HEX 3 END 4 EIB 5 VAR 6 USER 7 SBH 8 SFH 9 MSG 10 SB 11 SF

```

Figure 58. Typical CECI display for command completed

The command has been processed and the results are displayed on the screen.

Any **receivers**, whether specified or not, together with their CICS-supplied values, are displayed intensified.

The body

The body of **command syntax check**, **about to start command**, and **command completed** screens contains information common to all three displays.

The full syntax of the command is displayed. Options specified in the command line or assumed by default are intensified, to show that they will be used in executing the command, as are any **receivers**. The < > brackets indicate that you can select an option from within these brackets. If you make an error in your syntax, CECI diagnoses it in the message area that follows the body, described in “The message line” on page 254. If there are too many diagnostic messages, the rest of the messages can be displayed using PF9.

Arguments can be displayed in either character or hexadecimal format. You can use PF2 to switch between formats. In character format, some characters are not displayable (including lowercase characters on some terminals); CECI shows them as periods. You need to switch to hexadecimal to show the real values, and you need to use caution when modifying them, as explained in “Making changes” on page 258.

If the value of an option is too long for the line, only the first part is displayed followed by “...” to indicate there is more. You can display the full value by positioning the cursor at the start of the option value and pressing ENTER. This produces an expanded display described in “Expanded area” on page 255.

If the command has more options than can fit on one screen, a plus sign (+) appears at the left-hand side of the last option of the current display to indicate that there are more. An example of this is shown in Figure 60 on page 257. You can display additional pages with the PF keys for scrolling.

The message line

CECI uses the message line to display error messages. After execution of a command, the message line shows the response code. Figure 56 on page 250 shows an error message, where the user has omitted a required field. The **S** that precedes the message indicates that it is severe (bad enough to prevent execution). There are also warning messages (flagged by **W**) and error messages (flagged by **E**), which provide information without preventing execution. **E** messages indicate option combinations unusual enough that they may not be intended and warrant a review of the command before you proceed with execution.

Where there are multiple error messages, CECI creates a separate display containing all of them, and uses the message line to tell you how many there are, and of what severity. You can get the message display with PF9, as explained in “Additional displays” on page 255.

Figure 58 on page 253 shows the second use of the message line, to show the result of executing a command. CECI provides the information in both text (FILENOTFOUND in the example in Figure 58 on page 253) and in decimal form (the EIBRESP value).

CECI options on function keys

The single line at the foot of the screen provides a menu indicating the effect of the PF keys for the display. See Figure 56 on page 250.

The PF keys are described below. If the terminal has no PF keys, the same effect can be obtained by positioning the cursor under the required item in the menu and pressing ENTER.

PF1 HELP

displays a HELP panel giving more information on how to use the command interpreter and on the meanings of the PF keys.

PF2 HEX

(SWITCH HEX/CHAR)

switches the display between hexadecimal and character format. This is a mode switch; all subsequent screens stay in the chosen mode until the next time this key is pressed.

PF3 END

(END SESSION)

ends the current session of the interpreter.

PF4 EIB

(EIB DISPLAY)

shows the contents of the EXEC interface block (EIB). An example of this screen is shown in Figure 60 on page 257.

PF5 VAR

(VARIABLES)

shows all the variables associated with the current command interpreter session, giving the name, length, and value of each. See “Variables” on page 255 for more information about the use of this PF key.

PF6 USER

(USER DISPLAY)

shows the current contents of the user display panel (that is, what would appear on the terminal if the commands processed thus far had been executed by an ordinary program rather than the interpreter). This key is not meaningful until a terminal command is executed, such as EXEC CICS SEND MAP.

PF7 SBH

(SCROLL BACK HALF)

scrolls the body half a screen backward.

PF8 SFH

(SCROLL FORWARD HALF)

scrolls the body half a screen forward.

PF9 MSG

(DISPLAY MESSAGES)

shows all the messages generated during the syntax check of a command.

PF10 SB

(SCROLL BACK)

scrolls the body one full screen backward.

PF11 SF

(SCROLL FORWARD)

scrolls the body one full screen forward.

Additional displays

Additional screens of information are available when you press the relevant PF key. You can get back to your original screen by pressing ENTER from an unmodified screen.

Expanded area

This display uses the whole of the body of the screen to display a variable selected with the cursor. The cursor can be positioned at the start of the value of an option on a syntax display, or under the ampersand of a variable in a variables display. Pressing ENTER then gives the expanded area display. The scrolling keys can be used to display all the information if it exceeds a full screen.

Variables

Figure 59 shows the result of requesting a **variables** display, obtained by pressing PF5. For each variable associated with the current interpreter session, it shows the name, length, and value.

```
READ FILE('FILEA') RIDFLD('009000') INTO(&REC)
VARIABLES
&AEGC      +00016  THIS IS A SAMPLE
&AEGR      +00045  EXEC CICS READQ QUEUE(' CIS200') INTO(&AEGC)
&AEGW      +00046  EXEC CICS WRITEQ QUEUE(' CIS200') FROM(&AEGC)
&REC       +00080  482554 D694 72 WIDGET, .007 TEST 100
PF 1 HELP 2 HEX 3 END 4 EIB          6 USER                                9 MSG
```

Figure 59. Typical CECI display of variables associated with CECI session

The first three variables displayed are created for you by CECI and always appear unless you explicitly delete them. They are designed to help you create command lists, as described in “Saving commands” on page 260, as well as to serve as examples.

After these three, you will see any variables that you have created. The fourth one in Figure 59 on page 255, &REC, is the result of executing:

```
READ FILE('FILEA') RID('009000') INTO(&REC)
```

Normally, the value supplied for an option in the command line is taken as a character string constant. However, sometimes you need to specify a variable to represent this value, as when you want to connect two commands through option values.

For example, to change a record with CECI, you might first enter:

```
EXEC CICS READ UPDATE INTO(&REC)
      FILE('FILEA') RID('009000')
```

You would then modify the record as required by changing the variable &REC, and then enter:

```
EXEC CICS REWRITE FROM(&REC) FILE('FILEA')
```

The ampersand (&) in the first position tells CECI that you are specifying a variable.

A variable is also useful when the values of the options cause the command to exceed the line length of the command input area. Creating variables with the required values and specifying the variable names in the command overcomes the line length limitation.

Defining variables

Variables can have a data type of character, fullword, halfword, or packed decimal, and you can create them in any of the following ways:

- By naming the variable in a receiver (&REC in Figure 59 on page 255, for example). The variable is created when the command is processed. The data type and length are implied by the option.
- By adding new entries to the list of variables already defined. To create a new variable, simply type its name and length in the appropriate columns on the first unused line of the variables display, and then press ENTER. For character variables, use the length with which the variable has been defined. For fullwords or halfwords, type **F** or **H**. For packed variables, use the length in bytes, preceded by a **P**.

Character variables are initialized to blanks. The others are initialized to zero in the appropriate form. Once a variable is created, you can change the value by modifying the data field on the **variables** display.

- By using the NAME field on the status line when you have produced an expanded area display of a particular option. You do this by positioning the cursor under the option on a syntax display and pressing ENTER. Then you assign the variable name you want associated with the displayed option value by typing it into the NAME field and pressing ENTER again.
- By copying an existing variable. You do this by obtaining an expanded area display of the variable to be copied, overkeying the name displayed with the name of the new variable, and pressing ENTER.

- By using the NAME field directly on a syntax display. This creates a character variable whose contents are the character string on the command line, for use in command lists as explained in “Saving commands” on page 260.

You can also delete a variable, although you don’t usually need to, as CECI discards all variables at session end. To delete one before session end, position the cursor under the ampersand that starts the name, press ERASE EOF and then press ENTER.

The EXEC interface block (EIB)

You can display the EIB associated with the CECI transaction by pressing PF4. Figure 60 shows an example of the contents of the EXEC interface block (EIB).

```

READ FILE('FILEA') RIDFLD('009000')
EXEC INTERFACE BLOCK
EIBTIME      = +0124613
EIBDATE      = +0091175
EIBTRNID     = 'CECI'
EIBTASKN     = +0000046
EIBTRMID     = 'S200'
EIBCPOSN     = +000004
EIBCALEN     = +000000
EIBAID       = X'7D'
EIBFN        = X'0000'          (READ)
EIBRCODE     = X'000000000000'
EIBDS        = 'FILEA...'
EIBREQID     = '.....'
EIBRSRCE     = 'FILEA '
EIBSYNC      = X'00'
EIBFREE      = X'00'
EIBRECV      = X'00'
+ EIBATT     = X'00'
PF 1 HELP 2 HEX 3 END 4 EIB 5 VAR 6 USER 7 SBH 8 SFH 9 MSG 10 SB 11 SF

```

Figure 60. Typical CECI display of the EIB

The fields in the EIB are defined in Appendix A, “EXEC interface block,” on page 529..

Error messages display

When there are more messages than CECI can display on the message line, you can display all of them by pressing PF9.

```

READ
MESSAGES
  S FILE must be specified.
  S RIDFLD must be specified.
PF 1 HELP      3 END 4 EIB 5 VAR 6 USER 7 SBH 8 SFH      10 SB 11 SF

```

Figure 61. Typical CECI display of the message display

Making changes

Until CICS executes a command, you can change it by changing the contents of the command line, by changing the option values shown in the syntax display in the body, or by changing the values of variables on the **variables** screen. (You can still make changes after a command is executed, but, unless they are in preparation for another command, they have no effect.)

When you make your changes in the command line or on the **variables** screen, they last for the duration of the CECI transaction. If you make them in the body of the syntax screen, however, they are temporary. They last only until the command is executed and are not reflected in the command line.

As noted earlier, not all characters are displayable on all terminals. When the display is in character rather than hexadecimal format, CECI shows these characters as periods (X'4B'). When you overwrite a period, you should be aware that the current value may not be a period but an undisplayable character.

Furthermore, you cannot change a character to a period when the display is in character mode. If you attempt this, CECI ignores your change, and does not issue a diagnostic message. To make such a change, you have to switch the display to hexadecimal and enter the value (X'4B') that represents a period.

There is a restriction on changes in hexadecimal format as well. If you need to change a character to a blank, you cannot enter the code (X'40') from a hexadecimal display. Again, your change is ignored and CECI does not issue a message. Instead, you must switch to character mode and blank out the character. Take care not to enter invalid characters when making changes in hexadecimal format as this may cause an abend.

After every modification, CECI rechecks your syntax to ensure that no errors have appeared. It restarts processing at the **command syntax check** if there are any execution-stoppers, and at **about to start command** if not. Only after you press ENTER on an unmodified **about to start command** screen does CECI execute your command.

How CECI runs

There are several things you should know about how the interpreter works, in order to use it properly. These include:

- CECI sessions
- Abends
- Exception conditions
- Program control commands
- Terminal sharing
- Saving commands

CECI sessions

The interpreter runs as a transaction, using programs supplied by CICS. It is conversational, which means that everything you do between the start of a session (entering CECI) and the end (PF3) is a single logical unit of work in a single task. This means that locks and enqueues produced by commands you execute remain for the duration of your session. If you read a record for update from a recoverable file, for example, that record is not available to any other task until you end CECI.

Abends

CECI executes all commands with the NOHANDLE option, so that execution errors do not ordinarily cause abends.

CECI also issues an EXEC CICS HANDLE ABEND command at the beginning of the session, so that it will not lose control even if an abend occurs. Consequently, when you get one, CECI handles it and there is no resource backout. If you are doing a series of related updates to protected resources, you need to be sure that you do not do some of them and then find you cannot complete the others. If you find yourself in this situation, you can execute an EXEC CICS SYNCPOINT ROLLBACK command or an EXEC CICS ABEND command with the CANCEL option to remove the effects of your earlier commands on recoverable resources.

Exception conditions

For some commands, CECI may return exception conditions even when all specified options are correct. This occurs because, on some commands, CECI uses options that you do not specify explicitly. For example, the EXEC CICS ASSIGN command always returns the exception condition INVREQ under CECI. Even though it may return the information you requested correctly, it will have attempted to get information from other options, some of which will be invalid.

Program control commands

Because the interpreter is itself an application program, the interpretation of some program control commands may produce different results from an application program executing those commands. For example, EXEC CICS ABEND command is intercepted, as noted above, unless you use the CANCEL option.

If you execute an EXEC CICS LINK command, the target program executes, but in the environment of the interpreter, which may not be the one expected. In particular, if you modify a user display during a linked-to program, the interpreter will not be aware of the changes.

Similarly, if you interpret an EXEC CICS XCTL command, CECI passes control to the named program and never gets control back, so that the CECI session is ended.

Terminal Sharing

When the command being interpreted is one that uses the same screen as the interpreter, the command interpreter manages the sharing of the screen between the interpreter display and the user display.

The user display is restored:

- When the command being processed requires input from the operator
- When the command being processed is about to modify the user display
- When USER DISPLAY is requested

Thus, when an EXEC CICS SEND command is followed by an EXEC CICS RECEIVE command, the display sent by the EXEC CICS SEND command appears twice, once when the EXEC CICS SEND command is processed, and again when the EXEC CICS RECEIVE command is processed. It is not necessary to respond to the EXEC CICS SEND command, but, if a response is made, the interpreter stores it and redisplay it when the screen is restored for the EXEC CICS RECEIVE command.

When the interpreter restores the user display, it does not sound the alarm or affect the keyboard in the same way as when an EXEC CICS SEND command is processed. This is similar to EDF (see “Using EDF in single-screen mode” on page 238 for more information).

Saving commands

Sometimes you may want to execute a command, or a series of commands, under CECI, repeatedly. One technique for doing this is to create a temporary storage queue containing the commands. You then alternate reading the command from the queue and executing it.

CECI provides shortcuts both for creating the queue and for executing commands from it. To create the queue:

1. Start a CECI session.
2. Enter the first (or next) command you want to save on the command line, put &AEGC in the NAME field in the status line, and press ENTER. This action causes the usual syntax check, and it also stores your command as the value of &AEGC, which is the first of those three variables that CECI always defines for you. (See Figure 59 on page 255.) If you select the variables display at this point, you will see that &AEGC is the value of your command.
3. After the syntax is correct but before execution (on the **about to start command** screen), change the command line to &AEGW and press ENTER. This causes CECI to use the value of &AEGW for the command to be executed. &AEGW is the third of the variables CECI supplies, and it contains a command to write the contents of variable &AEGC (that is, your command) to the temporary storage queue named “ CI`ttt`”, where “`ttt`” is the name of your terminal and two blanks precede the letters “CI”.
4. Execute this EXEC CICS WRITEQ command (through the **command completed** screen). This stores your command on the queue.
5. If there is more than one command you want to save, repeat steps (2) through (4) for each.

When you want to execute the saved commands from the list, do the following:

1. Enter &AEGR on the command line and press ENTER. &AEGR is the second of the CECI-supplied variables, and it contains a command to read the queue that was written earlier. Execute this command; it will bring the first (next) of the commands you saved into the variable &AEGC
2. Then enter &AEGC on the command line and press ENTER. CECI replaces the command line with the value of &AEGC, which is your command. Press ENTER to execute your command.
3. Repeat these two steps, alternating &AEGR and &AEGC on the command line, until you have executed all of the commands you saved.

You can vary this procedure to suit your needs. For example, you can skip commands in the sequence by simply skipping step (2). You can change the options of the saved command before executing it in the same way as a command entered normally.

If you want to repeat execution of the saved sequence, you need to specify the option ITEM(1) on the first execution of the EXEC CICS READQ command, in order to reposition your read to the beginning of the queue.

Security considerations

The interpreter is such a powerful tool that your installation may restrict its use. For information about security, see *AS/400 Security - Reference* manual and the *CICS for iSeries Administration and Operations Guide*.

Part 7. Programming reference

Chapter 30. OS/400 control language (CL)

commands	265
Interpreting the syntax diagrams	265
CRTCICSCBL	266
CRTCICSC	286
CRTCICSMAP	301

Chapter 31. Programming reference 305

Introduction to EXEC CICS commands	305
Command format	305
CICS syntax notation used	306
Argument values	307
COBOL argument values	308
ILE C argument values	309
CICS-value data areas (CVDAs)	309
DATASET option	311
INTO and SET options	311
LENGTH options	312
NOHANDLE option	312
RESP and RESP2 options	313
System programming commands	314
INQUIRE and SET commands	315
Browsing resource definitions	315
Null values	317
PERFORM command	318
DISCARD commands	318
Commands by function	319
Abend support	319
APPC mapped conversation	319
BMS	319
Built-in function	319
Diagnostic services	319
Environment services	319
Exception support	320
File control	320
Interval control	321
Journaling	321
Printer spooling	321
Program control	321
Storage control	321
Syncpoint	321
Task control	321
Temporary storage control	321
Terminal control	322
Transient data control	322

Chapter 32. Application programming

commands - reference	323
ABEND	323
ADDRESS	324
ALLOCATE	325
ASKTIME	326
ASSIGN	327
BIF DEEDIT	332
CANCEL	333
CONNECT PROCESS	335

CONVERSE (APPC)	337
CONVERSE (5250 or 3270 logical)	339
DELAY	341
DELETE	344
DELETEQ TD	348
DELETEQ TS	349
DEQ	350
DUMP TRANSACTION	352
ENDBR	353
ENQ	355
ENTER TRACENUM	357
EXTRACT ATTRIBUTES (APPC)	359
EXTRACT PROCESS	360
FORMATTIME	361
FREE (APPC)	364
FREEMAIN	365
GETMAIN	367
HANDLE ABEND	369
HANDLE AID	371
HANDLE CONDITION	372
IGNORE CONDITION	373
ISSUE ABEND	374
ISSUE CONFIRMATION	375
ISSUE ERASEAUP	376
ISSUE ERROR	377
ISSUE PREPARE	378
ISSUE SIGNAL (APPC)	379
LINK	380
LOAD	385
POP HANDLE	386
POST	387
PUSH HANDLE	389
READ	390
READNEXT	395
READPREV	400
READQ TD	404
READQ TS	407
RECEIVE (APPC)	410
RECEIVE (5250 or 3270 logical)	412
RECEIVE MAP	415
RELEASE	417
RESETBR	418
RETRIEVE	421
RETURN	424
REWRITE	427
SEND (APPC)	430
SEND (SCS)	432
SEND (5250 or 3270 logical)	433
SEND CONTROL	435
SEND MAP	436
SEND TEXT	439
SPOOLCLOSE	441
SPOOLOPEN OUTPUT	442
SPOOLWRITE	444
START	445
STARTBR	452

SUSPEND	457
SYNCPOINT	457
SYNCPOINT ROLLBACK	458
UNLOCK	458
WAIT CONVID	461
WAIT EVENT	461
WAIT JOURNALNUM	462
WRITE	463
WRITE JOURNALNUM.	467
WRITEQ TD.	469
WRITEQ TS.	471
XCTL	474

Chapter 33. System programming reference	477
DISCARD commands	477
DISCARD AUTINSTMODEL	477
DISCARD FILE.	478
DISCARD PROGRAM	478
DISCARD TRANSACTION.	479
INQUIRE commands	480
INQUIRE AUTINSTMODEL	480
INQUIRE AUTINSTMODEL (browse)	480
INQUIRE CONNECTION	481
INQUIRE CONNECTION (browse)	483
INQUIRE FILE	484
INQUIRE FILE (browse).	487
INQUIRE JOURNALNUM	488
INQUIRE JOURNALNUM (browse).	489
INQUIRE PROGRAM	490
INQUIRE PROGRAM (browse)	492
INQUIRE SYSTEM	493
INQUIRE TASK	494
INQUIRE TDQUEUE.	496
INQUIRE TDQUEUE (browse)	499
INQUIRE TERMINAL or NETNAME	500
INQUIRE TERMINAL (browse)	504
INQUIRE TRACEDEST	505
INQUIRE TRANSACTION	506
INQUIRE TRANSACTION (browse).	508
PERFORM SHUTDOWN command	509
SET commands.	509
SET CONNECTION	509
SET FILE	511
SET JOURNALNUM	514
SET PROGRAM	515
SET SYSTEM	517
SET TASK	517
SET TDQUEUE.	518
SET TERMINAL	520
SET TRACEDEST	522
SET TRANSACTION	524

Chapter 30. OS/400 control language (CL) commands

CICS/400 has three control language (CL) commands for use by application programmers:

CRTCICSCBL CRTCICSCBL precompiles a COBOL/400 or ILE COBOL program with embedded CICS commands.

When you are ready to compile your program, you must first run it through a precompiler to translate the CICS commands and SQL commands (if they are used) into COBOL/400 or ILE COBOL statements.

CRTCICSCBL is used in the same way as the CRTSQLCBL and CRTCLPGM commands. The authority assigned to this command is PUBLIC(*USE).

See page 266 for further details.

CRTCICSC CRTCICSC preprocesses and translates ILE C source code, containing EXEC CICS and SQL commands, and produces one of the following:

- A program module
- A bound program

CRTCICSC is used in the same way as other CL commands. The authority assigned to this command is PUBLIC(*USE).

See page 286 for further details.

CRTCICSMAP

CRTCICSMAP creates BMS physical and symbolic maps.

The physical map is created as a *USRSPC in the library specified. The symbolic map is created as a source member in the source library specified. You can then copy the resulting structure into any application program that refers to the map set before compiling a COBOL/400, COBOL or ILE C program.

Using CICS/400 BMS maps can be considered similar to using DDS for defining OS/400 display files. CRTCICSMAP should have the same security restrictions as any other command that generates program objects. The authority assigned to this command is PUBLIC(*USE).

See page 301 for further details.

If you enter just the command with no parameters in the command area and press the PF4 key, the first prompt screen is displayed, showing all the available parameters and their defaults. You can scroll through the screens of parameters to select those you want.

Interpreting the syntax diagrams

Table 12 on page 306 explains the command syntax conventions used in this book. You interpret the syntax by following the arrows from left to right.

Syntax diagrams show all the parameters and values used by each CL command. Each syntax diagram specifies, for one command, the parameters that can be coded on the command and the choice of values that are valid for each parameter.

Some parameters for a command may be entered without their keywords. The system determines which parameter is indicated by its position in the list. Refer to the relevant syntax diagrams for details. All parameters are shown in each diagram in the order required by the system for positional coding.

All required parameters precede all optional parameters. The required parameters (if any) are grouped on the same line as the command name at the beginning of the diagram. All the other parameters are optional and do not have to be coded; a default value (shown above the line) is assumed for each uncoded parameter for most commands.

For each parameter that can have a repetition of values, the maximum number of repetitions that can be coded is shown in the diagram with the parameter's values. The syntax diagrams also show (by the use of flow lines and notes) which parameters are dependent on the values of other parameters (mutually dependent) and which can be used only if another parameter or value does not cause a conflict (mutually exclusive).

Entry codes shown in the upper right corner of the diagram indicate the environment in which the command can be specified. Notes[®] are also included to give information needed to interpret the syntax.

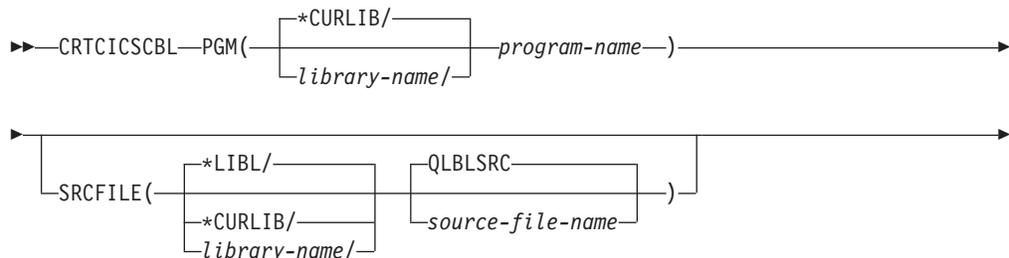
For an explanation of the naming convention used in the diagrams, see "Conventions and terminology used in this book" on page xiii.

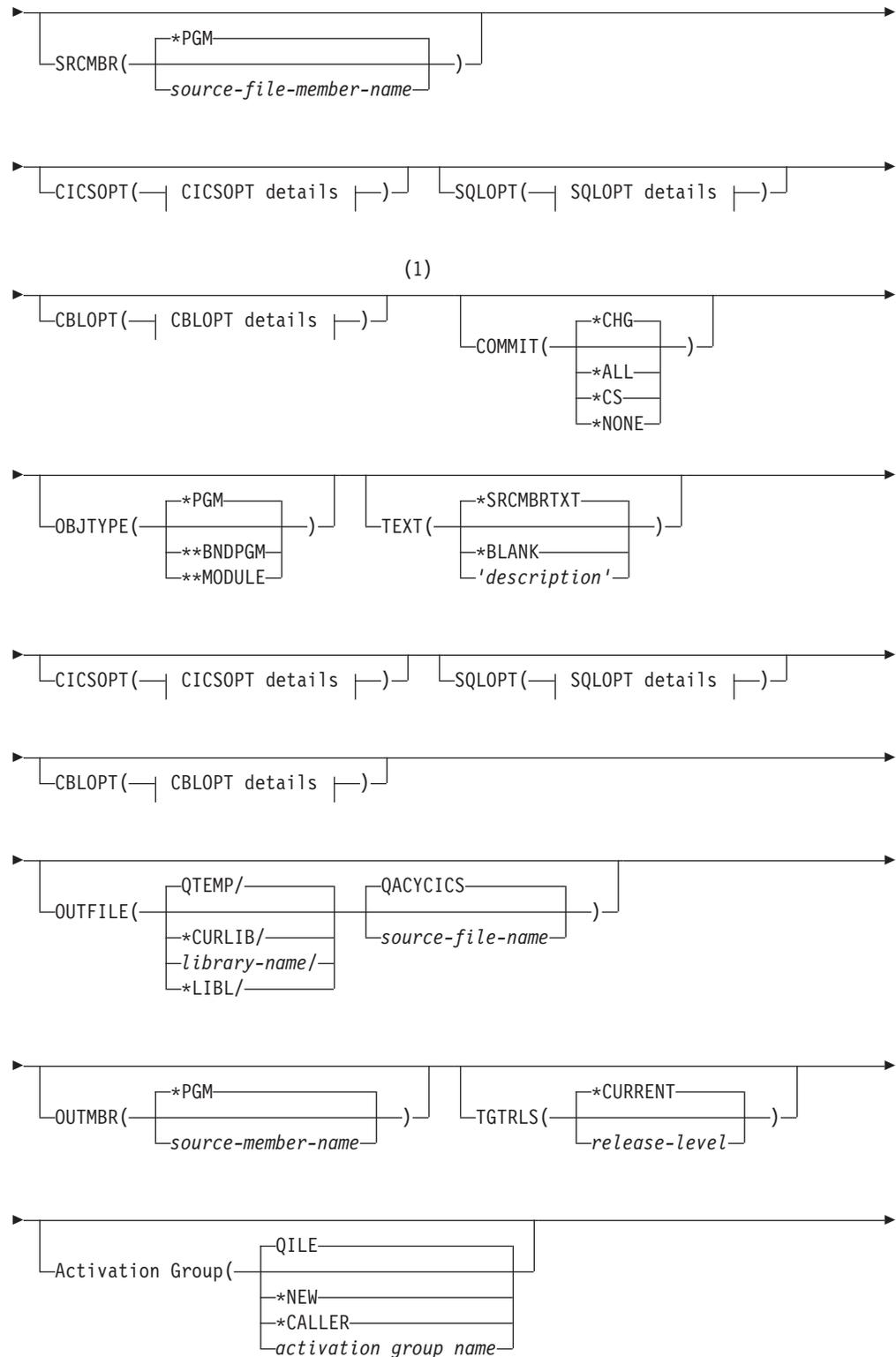
CL command defaults

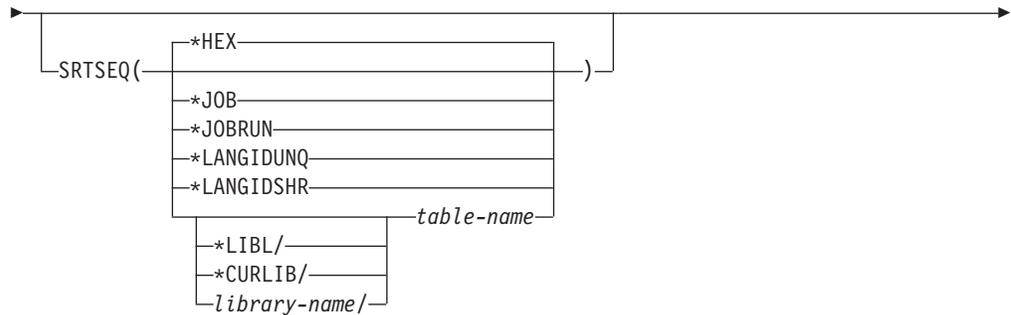
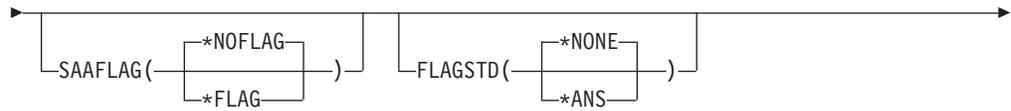
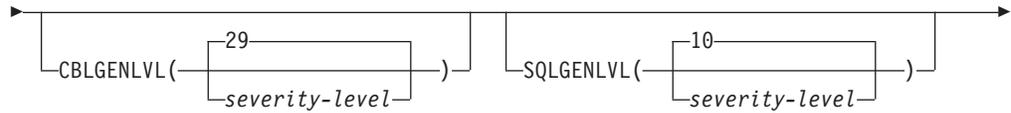
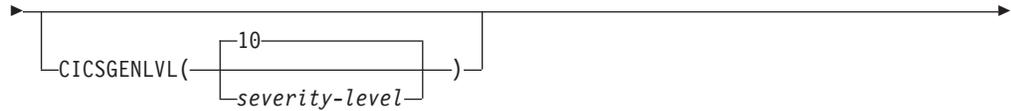
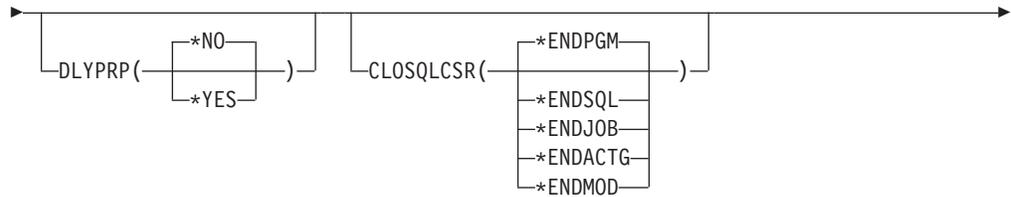
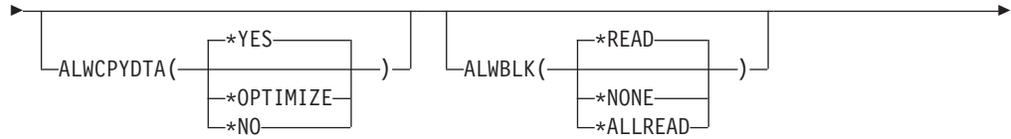
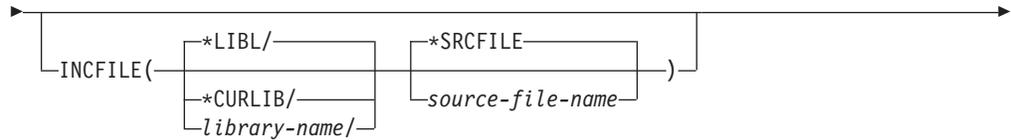
The defaults given in the CL command descriptions are those that are supplied with the iSeries system. You should check that your installation has not made any changes to these command default parameters.

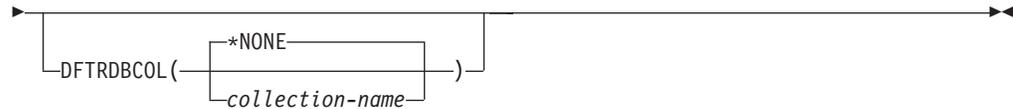
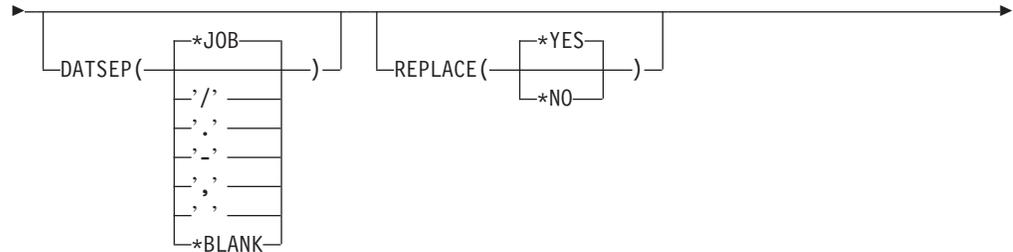
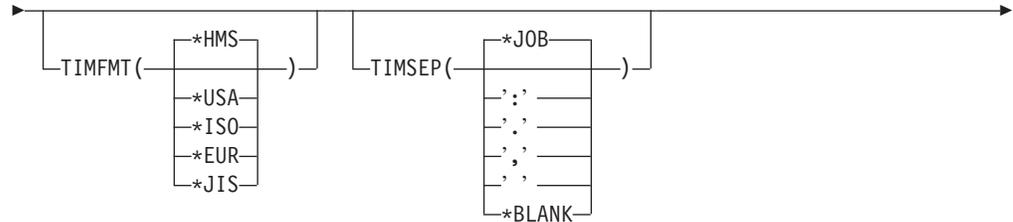
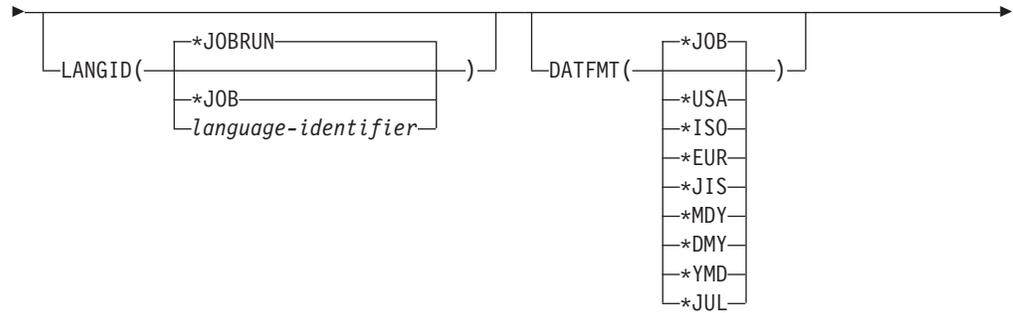
CRTCICSCBL

Job: B,I Pgm: B,I REXX: B,I Exec





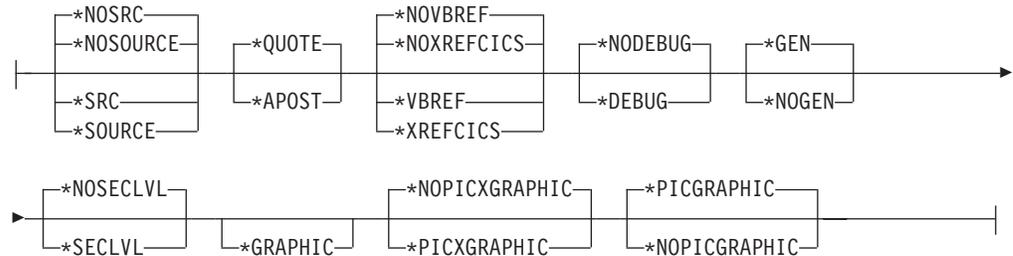




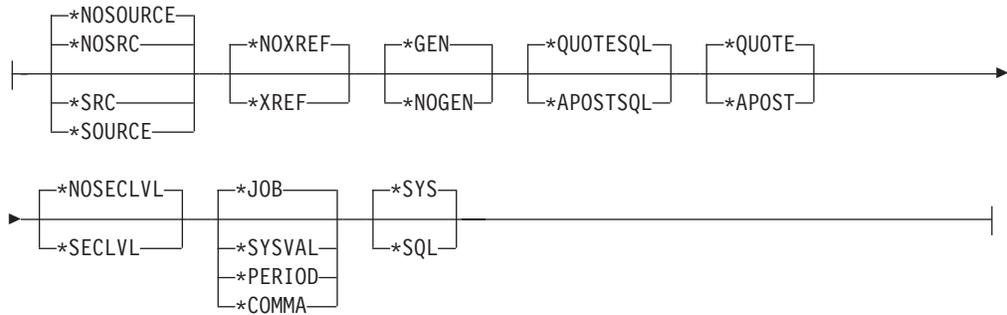
Notes:

- 1 All parameters preceding this point can be specified positionally.

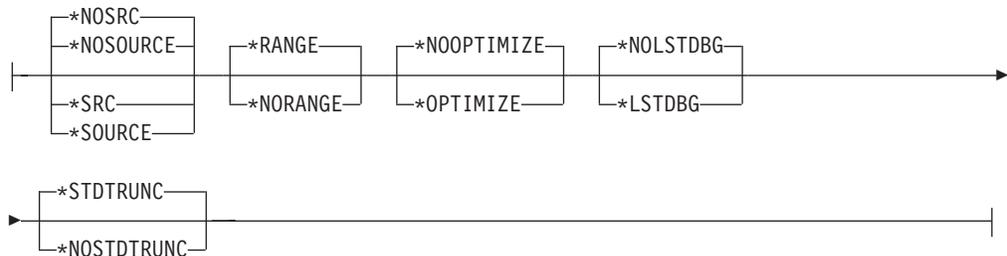
CICSOPT details:



SQLOPT details:



CBLOPT details:



Function

The Create CICS COBOL (CRTICSCBL) command calls the CICS precompiler, which precompiles COBOL source containing CICS statements and produces a temporary source member. If the source program also contains Structured Query Language (SQL) commands, it optionally calls the SQL precompiler following a successful CICS precompile. The resulting precompiler output is placed into a temporary source member. Following the CICS precompilation and the optional SQL precompilation, the COBOL compiler may optionally be called to compile the program.

This command calls either the COBOL/400 compiler or the ILE COBOL compiler based on the value selected in the OBJTYPE parameter.

If the program uses any copybooks, you should make sure that library QCICS and the library containing your COBOL copybook are on the library list before attempting to compile the program. For further information, see the *CICS for iSeries Administration and Operations Guide*.

Note: It is possible to precompile your SQL statements first by running the CRTSQLCBL command and specifying the *NOGEN option on the OPTION parameter. This puts the SQL precompiler output in QSQLTEMP in library QTEMP. You must then run the CRTICSCBL command specifying SRCFILE as QTEMP/QSQLTEMP; the SRCMBR name can be obtained by looking in this file for your SQL precompiled program source. The rest of the parameters on CRTICSCBL can be entered as normal.

It is not recommended that applications be compiled in this way but, if necessary, it can be done.

Required parameters

Program (PGM)

specifies the qualified name by which the compiled program is known.

The possible library values are:

***CURLIB:** If a library is not specified, the program is created in the current library. If no current library entry exists in the library list, QGPL is used.

library-name: Specify the name of the library where the compiled program is created.

program-name: Specify the name of the program being created that contains the CICS statements.

Attention: If the program name you specify is the same name as an existing program, your new program replaces the existing one if the REPLACE parameter is specified as *YES (the default).

The following parameters are optional for the CICS precompiler source commands. If you choose not to specify any of the following keywords or their values, the defaults are used. SQLOPT lists those options that apply to SQL. For more information, see the Database and file systems topic in the iSeries Information Center.

Source file (SRCFILE)

specifies the qualified name of the source file that contains the COBOL source with the EXEC CICS or EXEC SQL statements.

The possible library values are:

***LIBL:** Specifies that the library list is used to locate the source file.

***CURLIB:** Specifies that the current library for the job is used to locate the source file. If no current library entry exists in the library list, QGPL is used.

library-name: Specify the name of the library where the source file is located.

QLBLSRC: If a COBOL source file name is not specified, the supplied source file QLBLSRC contains the COBOL source.

source-file-name: Specify the name of the source file that contains the COBOL source. This source file should have a record length of 92 bytes. The source file can be a database file, device file, or an inline data file.

Source member (SRCMBR)

specifies the name of the source file member that contains the COBOL source. This parameter is only specified if the source file name in the SRCFILE parameter is that of a database file.

***PGM:** Specifies that the COBOL source is in the source file member that has the same member name as that specified in the PGM parameter for the precompiler command.

source-file-member-name: Specify the name of the source file member that contains the COBOL source.

Compile type (OBJTYPE)

specifies COBOL compiler to be used and the type of object to be created.

***PGM:** Specifies that the translated CICS source is to be passed to the CRTCPBLPGM command to create a COBOL program object.

***BNDPGM:** Specifies that the translated CICS source is passed to the CRTBNDCBL command to create a bound ILE COBOL program object.

***MODULE:** Specifies that the translated CICS source is passed to the CRTCPBLMOD command to create a module object.

Note: You can then bind the module with others into a run-time program using the CRTCPGM CL command.

Commitment control (COMMIT)

specifies whether SQL statements in the compiled program are run under commitment control. Files referred to in the host language source are not affected by this parameter. Only SQL tables, views, and SQL packages referred to in SQL statements are affected.

***CHG:** Specifies that the objects referred to in SQL COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements and the rows updated, deleted, and inserted are locked until the end of the unit of work (transaction). Uncommitted changes in other jobs can be seen.

***ALL:** Specifies that the objects referred to in SQL COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements and the rows updated, deleted, and inserted are locked until the end of the unit of work (transaction). Uncommitted changes in other jobs cannot be seen.

***CS:** Specifies that the objects referred to in SQL COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements and the rows updated, deleted, and inserted are locked until the end of the unit of work (transaction). A row that is selected, but not updated, is locked until the next row is selected. Uncommitted changes in other jobs cannot be seen.

***NONE:** Specifies that commitment control is not used. COMMIT and ROLLBACK statements are not allowed. SQL COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements and the rows updated, deleted, and inserted are locked until the end of the unit of work (transaction). A row that is selected, but not updated, is locked until the next row is selected. Uncommitted changes in other jobs cannot be seen.

Note: If *CHG, *CS, or *ALL is specified, DROP COLLECTION cannot be included in the application. The default for this parameter for the Start SQL (STRSQL) command is *NONE.

Text description (TEXT)

specifies the text relating to the program.

***SRCMBRTXT:** Specifies that the text is to be taken from the source file member being used to create the program. Text for a database source member can be added or changed by using the Source Entry Utility (STRSEU) command, or by using either the Add Physical File Member (ADDPFM) or the Change Physical File Member (CHGPFM) command. If the source file is an inline file or a device file, the text is blank.

***BLANK:** No text is specified.

'description': Specify no more than 50 characters of text, enclosed by apostrophes.

CICS options (CICSOPT)

specifies whether one or more of the following options are to be used when the source is translated. If an option is specified more than once, or if two options conflict, the last option specified is used. If an option is not specified, the default is used.

Source listing options

***NOSRC** or ***NOSOURCE**: Specifies that the CICS translator is not to produce a source listing.

***SRC** or ***SOURCE**: Specifies that the CICS translator is to produce a source listing and error messages.

COBOL String delimiter options

***QUOTE**: Specifies that a double quotation mark (") is used for string delimiters within CICS statements, and also for nonnumeric literals and Boolean literals in the COBOL statements.

***APOST**: Specifies that an apostrophe (') is used for string delimiters within CICS statements, and also for nonnumeric literals and Boolean literals in the COBOL statements.

Cross-reference options

***NOVBREF** or ***NOXREFCICS**: Specifies that the CICS translator is not to produce a cross-reference of EXEC CICS names.

***VBREF** or ***XREFCICS**: Specifies that the CICS translator is to produce a cross-reference between EXEC CICS names in the program and the statement numbers in the program that refer to them.

Debug Options

***NODEBUG**: Specifies that the CICS translator is not to produce code that will be passed through to CICS to be displayed by the CICS execution diagnostic facility (EDF).

***DEBUG**: Specifies that the CICS translator is to produce code that will be passed through to CICS to be displayed by EDF.

Program creation options

***GEN**: Specifies that the SQL precompiler or ILE C compiler is to be called after a successful CICS translation.

Note: The SQL precompiler is to be called only if, during the CICS translation stage, any EXEC SQL statements were found in the ILE C source code being CICS translated.

***NOGEN**: Specifies that the compilation is to terminate at the end of the CICS translation.

String delimiter and literal options

Second-level help text

***NOSECLVL**: Specifies that no second-level help text is to be printed.

*SECLVL: Specifies that the second-level help text is to be printed.

Note: The first-level help text is printed automatically each time an error occurs.

DBCS Enablement Options

***GRAPHIC**: The precompiler recognizes double-byte character set (DBCS) data when it is found in program literals. The precompiler recognizes shift out (SO) and shift in (SI) control characters in program literals, and does not recognize characters on an SBCS basis inside the bounds of a SO/SI pair. However, the precompiler does not attempt to impose the rules of COBOL/400 with regard to DBCS only and mixed literals. It assumes that all literals containing DBCS characters can be continued over one or more lines and can be of any length. DBCS characters are accepted in program comments but must not be used in EXEC CICS commands, either as keywords or as literal arguments.

***NOPIXGRAPHIC**: DBCS-graphic data types are declared as FILLER fields.

***PICXGRAPHIC**: Fixed-length DBCS-graphic data types are declared as fixed length alphanumeric fields and are accessible to the ILE COBOL source program. When the *VARCHAR option is also in use, variable length DBCS-graphic data types are declared as fixed-length group items and are accessible to the ILE COBOL source program.

***PICGRAPHIC**: Fixed-length DBCS-graphic data types are declared as fixed-length G-type fields and are accessible to the ILE COBOL source program.

***NOPICGGRAPHIC**: DBCS-graphic data types are declared as FILLER fields.

SQL options

specifies whether one or more of the following options are to be used when the source is translated. If an option is specified more than once, or if two options conflict, the last option specified is used. If an option is not specified, the default is used.

Source listing options

***NOSRC** or ***NOSOURCE**: Specifies that the CICS translator is not to produce a source listing.

***SRC** or ***SOURCE**: Specifies that the CICS translator is to produce a source listing and error messages.

Cross-reference options

***NOXREF**: Specifies that the SQL pre-compiler is not to produce a cross-reference of EXEC SQL names.

***XREF**: Specifies that the SQL compiler is to produce a cross-reference between EXEC SQL names in the program and the statement numbers in the program that refer to them.

Program creation options

***GEN**: Specifies that the COBOL/400 compiler is to be called after a successful SQL translation.

***NOGEN**: Specifies that the compilation is to terminate at the end of the SQL translation.

SQL Decimal point options

***JOB:** The representation for the decimal point for the job at SQL pre-compile time is used. If the QDECFMT specifies that the value used as the decimal point is a comma, any numeric constants in lists (such as in the SELECT clause or the VALUES clause) must be separated by a comma followed by a blank. For example, VALUES(1,1, 2,23, 4,1) is equivalent to VALUES(1.1,2.23,4.1) where the decimal point is a period.

***SYSVAL:** Specifies that the value used for the decimal point in the SQL pre-compiler is from the QDECFMT system value. If the QDECFMT specifies that the value used as the decimal point is a comma, any numeric constants in lists (such as in the SELECT clause or the VALUES clause) must be separated by a comma followed by a blank. For example, VALUES(1,1, 2,23, 4,1) is equivalent to VALUES (1.1,2.23,4.1) where the decimal point is a period.

***PERIOD:** Specifies that the value used for the decimal point in the SQL pre-compiler is a period.

***COMMA:** Specifies that the value used for the decimal point in the SQL pre-compiler is a comma. If the QDECFMT specifies that the value used as the decimal point is a comma, any numeric constants in lists (such as in the SELECT clause or the VALUES clause) must be separated by a comma followed by a blank. For example, VALUES(1,1, 2,23, 4,1) is equivalent to VALUES(1.1,2.23,4.1) where the decimal point is a period.

SQL String delimiter options

***QUOTESQL:** Specifies that a double quotation mark (") is used for string delimiters within SQL statements.

***APOSTSQL:** Specifies that an apostrophe (') is used for string delimiters within SQL statements.

COBOL String delimiter option

***QUOTE:** Specifies that a double quotation mark (") is used for string delimiters for nonnumeric literals and Boolean literals in the COBOL statements.

***APOST:** Specifies that an apostrophe (') is used for string delimiters for nonnumeric literals and Boolean literals in the COBOL statements.

Naming Convention option

***SYS:** Specifies that the OS/400 naming convention will be used (library-name/file-name).

***SQL:** Specifies that the SQL naming conventions will be used (collection-name.table-name).

Second-level help text

***NOSECLVL:** Specifies that no second-level help text is to be printed.

***SECLVL:** Specifies that the second-level help text is to be printed.

Note: The first-level help text is printed automatically each time an error occurs.

COBOL options (CBLOPT)

specifies whether one or more of the following options are used when the

COBOL source is compiled. If an option is specified more than once, or if two options conflict, the last option specified is used. If an option is not specified, the default is used.

Source listing options

***NOSRC** or ***NOSOURCE**: Specifies that a source listing is not produced by the compiler.

***SRC** or ***SOURCE**: Specifies that a source listing is produced by the compiler, consisting of all the source input and error messages.

Verify ranges

***RANGE**: At run time, the system verifies that subscripts are within the correct ranges, but does not verify index ranges. It also checks for reference modification and compiler-generated substring operations.

***NORANGE**: The system does not verify ranges at run time.

Optimization

***NOOPTIMIZE**: The compiler performs only standard optimizations for the program.

***OPTIMIZE**: The program object created may run more efficiently and may require less storage. However, specifying ***OPTIMIZE** can substantially increase the time required to compile a program.

CODE/400 option

This option determines the kind of information you see on your programmable workstation when using the IBM CoOperative Development Environment/400 (CODE/400) product.

***NOLSTDBG**: The compiler does not produce a listing view or listing-level debugging information.

***LSTDBG**: The compiler produces a listing view, source-level error information, and listing-level debugging information.

Data Truncation Option

This option determines how the data is truncated if necessary by the COBOL compiler. This option only applies to USAGE BINARY data. This option is valid only for ILE COBOL programs.

***STDTRUNC**: When ***STDTRUNC** is specified, USAGE BINARY data is truncated to the number of digits in the PICTURE clause of the BINARY receiving field.

***NOSTDTRUNC**: When ***NOSTDTRUNC** is specified, BINARY receiving fields are truncated only at half-word, full-word and double-word boundaries. BINARY sending fields are also handled as half-words, fullwords, and double-words. Thus, the full binary content of the field is significant. Also, the DISPLAY statement will convert the entire content of the BINARY field, with no truncation.

CICS output file (OUTFILE)

specifies the qualified name of the intermediate precompiler output.

The possible library values are:

***QTEMP**: The supplied source file QTEMP is used.

***CURLIB:** The current library is searched. If no library is specified as the current library for the job, the QGPL library is used.

library-name: Specify the name of the library where to generate the intermediate precompiler output.

***LIBL:** All libraries in the user and system portions of the job's library are searched.

QACYCICS: If an output source file name is not specified, the supplied source file QACYCICS will contain the intermediate precompiler output.

source-file-name: Specify the name of the source file to contain the intermediate precompiler output.

CICS output member (OUTMBR)

specifies the name of the source file member that is to contain the intermediate precompiler output. If this parameter is not specified, the object name on the OBJ parameter is used.

***OBJ:** Specifies that the intermediate precompiler output has the same member name as that specified in the OBJ parameter.

source-member-name: specifies that the name of the source file member to which the intermediate precompiler output should be copied.

Target release (TGTRLS)

specifies the release level of the CICS/400 system on which you intend to use the object being created.

You can specify an exact release level in the format VxRxMx, where Vx is the version, Rx is the release, and Mx is the modification level. For example, V5R3M0 would be version 5, release 2, modification 0.

The possible values are:

***CURRENT:** Specifies that the object is to be used on the release of CICS/400 currently running on your system. For example, if V5R3M0 is running on the system, *CURRENT means that you intend to use the object on a system with V5R3M0 installed.

***PRV:** The object is to be used on the previous release of CICS/400. If V5R3M0 is running on your system, *PRV means that you intend to use the object on a system with V5R3M0 installed.

release-level: Specify the release level in the format VxRxMx. The object can be used on a system with the specified release.

Valid values depend on the current version, release, and modification level, and they change with each new release. The valid values for this release are V5R3M0, V5R2M0 and V5R1M0.

Activation Group (ACTGRP)

Specifies which activation group is to be used when this program is called. This parameter is applied only when the OBJTYPE parameter is *BNDPGM and if there are no SQL statements in the program. To create a program which contains both CICS and SQL statements and also uses an activation group other than QILE, you should use the *MODULE option for the OBJTYPE parameter value to create a module object and then use the CRTPGRM command to create a the bound program.

QILE: Specifies that the program is to be activated in the QILE activation group. The program remains active as long as the activation group remains

in place. Note that this program's static variables are initialized based on their VALUE clause the first time the program is called. Secondary calls to the program use the current values of the data unless the program uses the INITIAL option for COBOL or uses the ALWRINZ option for CICS. Note also that data fields that are not initialized based on the VALUE clause are not assigned any specific value like X'00' (low values).

***NEW:** Specifies that this program is to be activated in a new activation group each time it is called by CICS. This means that all variables are initialized to their initial values each time the program is called. When this program returns to CICS, the activation is terminated and all programs are deactivated and all files are closed.

***CALLER:** Specifies that this program is activated in the same activation group as its caller. For CICS, this means that it is activated in the same activation group as the program that LINKed to it. This program remains active as long as the activation group remains in place. Note that this program's static variables are initialized based on their VALUE clause the first time the program is called. Secondary calls to the program use the current values of the data unless the program uses the INITIAL option for COBOL or uses the ALWRINZ option for CICS. Note also that data fields that are not initialized based on the VALUE clause are not assigned any specific value like X'00' (low values).

activation group name: Specifies that this program is activated in the named activation group. This program remains active as long as the activation group remains in place. Note that this program's static variables are initialized based on their VALUE clause the first time the program is called. Secondary calls to the program use the current values of the data unless the program uses the INITIAL option for COBOL or uses the ALWRINZ option for CICS. Note also that data fields are not initialized based on the VALUE clause are not assigned any specific value like X'00' (low values).

INCLUDE file (INCFILE)

specifies the qualified name of the source file that contains the members included in the program with any SQL INCLUDE statement.

The possible library values are:

***LIBL:** Specifies that the library list is used to locate the source file.

***CURLIB:** Specifies that the current library for the job is used to locate the source file. If no current library entry exists in the library list, QGPL is used.

library-name: Specify the name of the library where the source file is located.

***SRCFILE:** Specifies the qualified source file you specified in the SRCFILE parameter that contains the source file members specified on any SQL INCLUDE statement.

source-file-name: Specify the name of the source file that contains the source file members specified on any SQL INCLUDE statement. The record length of the source file you specify here must be no less than the record length of the source file you specified for the SRCFILE parameter.

Allow copy of data (ALWCPYDTA)

specifies whether a copy of the data is allowed in a SELECT statement.

***YES:** A copy of the data can only be used, if necessary, to run a SELECT statement.

***OPTIMIZE:** The system chooses whether or not to use the data retrieved directly from the database or to use a copy of the data. The decision is based on which choice will provide the best performance.

This value decreases the time required for the total query. Because the copy of the data must be made before returning the first row of the result table, the time to retrieve the first row may be increased.

Note: If *CS or *ALL is specified on the COMMIT parameter, SQL run time ignores this parameter and uses current data.

***NO:** A copy of the data is not allowed. This option could return a negative SQLCODE if the clauses in the SELECT statement require a copy of the data. If the SELECT statement runs successfully, then current data was used.

Allow blocking (ALWBLK)

specifies whether the database manager can use record blocking, and the extent to which blocking can be used for read-only cursors.

***READ:** Records are blocked for read-only retrieval of data for cursors when:

- *NONE is specified on the COMMIT parameter, to indicate that commitment control is not used.
- The cursor is declared with a FOR FETCH ONLY clause or there are no dynamic statements that could run a positioned UPDATE or DELETE statement for the cursor.

Specifying *READ can improve the overall performance of queries that meet the above conditions and retrieve a large number of records.

***NONE:** Rows are not blocked for retrieval of data for cursors. Specifying *NONE:

- Guarantees that the data retrieved is current.
- May reduce the amount of time required to retrieve the first row of data for a query.
- Stops the database manager from retrieving a block of data rows that is not used by the program when only the first few rows of a query are retrieved before the query is closed.
- Can degrade the overall performance of a query that retrieves a large number of rows.

***ALLREAD:** Rows are blocked for read-only cursors if *NONE or *CHG is specified on the COMMIT parameter. All read-only cursors in a program are opened for read-only processing even though there may be EXECUTE statements in the program. Specifying *ALLREAD:

- Allows record blocking under commitment control. Specifying *READ does not.
- Improves the performance of almost all read-only cursors in programs, but limits queries in the following ways:
 - A ROLLBACK statement in host languages, or the ROLLBACK HOLD SQL statement, does not position the read-only cursor again when *ALLREAD is specified.

- Dynamic running of a positioned UPDATE or DELETE statement (for example, EXECUTE IMMEDIATE), cannot be used to update a row in a cursor unless the DECLARE statement for the cursor includes the FOR UPDATE clause.

Delay prepare (DLYPRP)

specifies whether the dynamic statement validation for a PREPARE statement is delayed until an OPEN, EXECUTE, or DESCRIBE statement is run. Delaying validation improves performance by eliminating duplicate validation.

***NO:** Dynamic statement validation is not delayed. When the dynamic statement is prepared, the access plan is validated. When the dynamic statement is used in an OPEN or EXECUTE statement, the access plan is revalidated. Because the authority or the existence of objects referred to by the dynamic statement may change, you must still check the SQLCODE or SQLSTATE after issuing the OPEN or EXECUTE statement, to ensure that the dynamic statement is still valid.

***YES:** Dynamic statement validation is delayed until the dynamic statement is used in an OPEN, EXECUTE, or DESCRIBE SQL statement. When the dynamic statement is used, the validation is completed and an access plan is built. If you specify *YES on this parameter for precompiled programs, you should check the SQLCODE and SQLSTATE after running an OPEN, EXECUTE, or DESCRIBE statement to ensure that the dynamic statement is valid. If you specify *YES, performance is not improved if the INTO clause is used on the PREPARE statement or if a DESCRIBE statement uses the dynamic statement before an OPEN is issued for the statement.

Close SQL cursor (CLOSQLCSR)

specifies when SQL cursors are implicitly closed, SQL prepared statements are implicitly discarded, and LOCK TABLE locks are released. SQL cursors are explicitly closed by issuing the CLOSE, COMMIT (without HOLD), or ROLLBACK (without HOLD) SQL statements.

***OBJTYPE:** Specifies that SQL cursors are handled according to the value defined in the *OBJTYPE parameter. If *PGM was specified then CRTCBPLPGM uses *ENDPGM. If *BNDPGM was specified then CRTBNDCBL uses *ENDACTGRP.

***ENDPGM:** Specifies that SQL cursors are closed, SQL prepared statements are discarded, and LOCK TABLE locks are released when the program ends.

***ENDSQL:** Specifies that SQL cursors remain open between calls and can be fetched without running another SQL OPEN. One of the programs higher on the call stack must have run at least one SQL statement. SQL cursors are closed, SQL prepared statements discarded, and LOCK TABLE locks released when the first SQL program on the call stack ends. If *ENDSQL is specified for a program that is the first SQL program called (the first SQL program on the call stack), the program is treated as if *ENDPGM were specified.

***ENDJOB:** Specifies that SQL cursors remain open between calls and can be fetched without running another SQL OPEN. None of the programs higher on the call stack needs to have run SQL statements. SQL cursors are left open, SQL prepared statements are preserved, and LOCK TABLE locks are held when the first SQL program on the call stack ends. SQL cursors are closed, SQL prepared statements are discarded, and LOCK TABLE

locks are released when the job ends. If you are running distributed database and are connected remotely, the connection is dropped.

|
| ***ENDACTGRP:** Specifies that SQL cursors are to be closed, SQL prepared
| statements are to be released, and LOCK TABLE locks are to be released
| when the activation group ends. This option is only allowed when using
| an ILE program.

|
| ***ENDMOD:** Specifies that SQL cursors are to be closed and SQL prepared
| statements are to be discarded when the module is exited. LOCK TABLE
| locks are to be released when the activation group ends. This option is
| only allowed when using an ILE program.

CICS generation severity level (CICSGENLVL)

specifies a level of CICS translator errors. If errors occur with a severity level greater than the value specified in this parameter, either the SQL precompiler is not called (if the CICS translator located any SQL in the source) or the compiler is not called.

10: If a severity-level value is not specified, the default severity level is 10.

severity-level: Specify a number in the range 10 through 40. Some suggested values are listed below:

- 10** The level value for warnings.
- 20** The level value for general error messages.
- 30** The level value for serious error messages.
- 40** The level value for system-detected error messages.

|
| **Note:** The value of CICSGENLVL applies only to messages generated as a
| result of CICS translation errors. The specified CICSGENLVL value
| is **not** passed to the SQL precompiler or the compiler.

COBOL generation severity level (CBLGENLVL)

specifies a level of COBOL compiler errors. If errors occur with a severity level greater than the value specified in this parameter, the COBOL program object is not created.

29: If a severity-level value is not specified, the default severity level is 29.

severity-level: Specify a number in the range 0 through 29.

SQL generation severity level

specifies a level of SQL precompiler errors. If errors occur with a severity level greater than the value specified in this parameter, the compiler is not called.

10: If no value is specified, the default severity level is 10.

severity-level: Specify a number in the range 10 through 40. Some suggested values are listed below:

- 10** The level value for warnings.
- 20** The level value for general error messages.
- 30** The level value for serious error messages.
- 40** The level value for system-detected error messages.

Note: The value of SQLGENLVL only applies to messages generated as a result of SQL precompilation errors. The specified SQLGENLVL value is **not** passed to the compiler.

SAA flagging (SAAFLAG)

specifies whether SQL statements that do not conform to SAA Level 2 Database standards are flagged.

***NOFLAG:** Specifies that the precompiler will *not* check for conformity to SAA standards.

***FLAG:** Specifies that the precompiler will check for conformity to SAA standards.

ANS flagging (FLAGSTD)

specifies whether nonstandard statements are flagged. This parameter allows you to flag SQL statements to verify that they conform to ANSI X3.135-1-1989, ANSI X3.168-1989, ISO 9075-1989, and FIPS 127.1 standards.

***NONE:** Specifies that the precompiler will *not* check for conformity to ANSI standards.

***ANS:** Specifies that the precompiler will check for conformity to ANSI standards.

Flagging severity (FLAG)

specifies the minimum severity level of messages to be printed. The possible values are:

0: All messages are printed.

severity-level: Enter a one or two-digit number that specifies the minimum severity level of messages to be printed. Messages that have severity levels of the specified value or higher are printed.

Sort sequence (SRTSEQ)

specifies the sort sequence used when NLSSORT is associated with an alphabet-name in the ALPHABET clause. The SRTSEQ parameter is used in conjunction with the LANGID parameter to determine which system-defined or user-defined sort sequence table the program will use.

The possible values are:

***HEX:** No sort sequence table is used, and the hexadecimal values of the characters are used to determine the sort sequence.

***JOB:** The sort sequence of the program is resolved and associated with the program at compile time. The sort sequence table must exist in the system at compile time.

***JOBRUN:** The sort sequence of the program is resolved and associated with the program at run time. At compile time, the compiler associates the sort sequence of the compile job with the program. At run time, this sort sequence is replaced by the sort sequence associated with the job at run time.

***LANGIDUNQ:** The sort sequence table being used must contain a unique weight for each character in the code page. The sort sequence table used is the unique weighted table associated with the language specified in the LANGID parameter.

***LANGIDSHR:** The sort sequence table being used can contain the same weight for multiple characters in the code page. The sort sequence table used is the shared weighted table associated with the language specified in the LANGID parameter.

table-name: The name of the sort sequence table to be used. The table contains weights for all characters in a given code page. A weight is associated with the character that is defined at the code point. When a sort sequence table name is specified, you can specify the library in which the table resides. Valid values are:

***LIBL:** Specifies that the library list is used to locate the sort sequence table.

***CURLIB:** Specifies that the current library for the job is used to locate the sort sequence table. If no current library entry exists in the library list, QGPL is used.

library-name: Specify the name of the library where the sort sequence table is located.

The valid PROCESS statement options for the SRTSEQ parameter are:

SRTSEQ(HEX)

SRTSEQ(JOB)

SRTSEQ(JOBRUN)

SRTSEQ(LANGIDUNQ)

SRTSEQ(LANGIDSHR)

SRTSEQ(*table-name*)

SRTSEQ(*library-name/table-name*)

SRTSEQ(LIBL/*table-name*)

SRTSEQ(CURLIB/*table-name*)

Language identifier (LANGID)

specifies the language identifier that is used in conjunction with the sort sequence. The LANGID parameter is used only when the SRTSEQ parameter value specified is either *LANGIDUNQ or *LANGIDSHR.

The possible values are:

***JOBRUN:** The language identifier of the program is resolved at run time. When the compiled program is run, the language identifier of the job is used. This allows a program to be compiled once and used with different language identifiers at run time.

***JOB:** The language identifier of the program is resolved at compile time.

language-identifier: A three-character language identifier, which must be one of the following:

Table 11. Language identifiers

AFR	Afrikaans	SQI	Albanian
ARA	Arabic	NLB	Belgian Dutch - Flemish
FRB	Belgian French	PTB	Brazilian Portuguese
BGR	Bulgarian	BEL	Byelorussian
FRC	Canadian French	CAT	Catalan
HRV	Croatian	CSY	Czech
DAN	Danish	NLD	Dutch
ENA	English Australian	ENP	English Upper Case
FIN	Finnish	FRA	French

Table 11. Language identifiers (continued)

DEU	German	ELL	Greek
HEB	Hebrew	HUN	Hungarian
ISL	Icelandic	GAE	Irish Gaelic
ITA	Italian	JPN	Japanese Katakana
KOR	Korean	MKD	Macedonian
NOR	Norwegian - Bokmal	NON	Norwegian - Nynorsk
PLK	Polish	PTG	Portuguese
RMS	Rhaeto-Romanic	ROM	Romanian
RUS	Russian	SRB	Serbian Cyrillic
SRL	Serbian Latin	CHS	Simplified Chinese
SKY	Slovakian	SLO	Slovenian
ESP	Spanish	SVE	Swedish
FRS	Swiss French	DES	Swiss German
ITS	Swiss Italian	THA	Thai
CHT	Traditional Chinese	TRK	Turkish
ENG	UK English	ENU	US English

The valid PROCESS statement options for the LANGID parameter are:

LANGID(JOBRUN)
 LANGID(JOB)
 LANGID(*language-identifier*)

Date format (DATFMT)

specifies the format used when accessing date result columns. All output date fields are returned in the specified format. For input date strings, the specified value is used to determine whether the date is in a valid format.

Note: An input date string that uses format *USA, *ISO, *EUR, or *JIS is always valid. If you connect to a relational database that is on a system that is not an OS/400 system, *USA, *ISO, *EUR, or *JIS must be used.

***JOB:** Specifies the format used for the job at precompilation. Use the Display Job (DSPJOB) command to determine the current date format for the job.

***USA:** The United States date format mm/dd/yyyy is used.

***ISO:** The International Organization for Standardization (ISO) date format yyyy-mm-dd is used.

***EUR:** The European date format dd.mm.yyyy is used.

***JIS:** The Japanese Industrial Standard date format yyyy-mm-dd is used.

***MDY:** The date format mm/dd/yy is used.

***DMY:** The date format dd/mm/yy is used.

***YMD:** The date format yy/mm/dd is used.

***JUL:** The Julian date format yy/ddd is used.

Time format (TIMFMT)

specifies the format used when accessing time result columns. All output

time fields are returned in the specified format. For input time strings, the specified value is used to determine whether the time is specified in a valid format.

Note: An input time string that uses the format *USA, *ISO, *EUR, or *JIS is always valid. If you connect to a relational database that is on a system that is not an OS/400 system, the time format must be *USA, *ISO, *EUR, *JIS, or *HMS, with a time separator of colon or period.

***HMS:** The hh:mm:ss format is used.

***USA:** The United States time format hh:mm xx is used, where xx is AM or PM.

***ISO:** The International Organization for Standardization (ISO) time format hh.mm.ss is used.

***EUR:** The European time format hh.mm.ss is used.

***JIS:** The Japanese Industrial standard time format hh:mm:ss is used.

Time separator character (TIMSEP)

specifies the separator used when accessing time result columns.

Note: This parameter applies only when *HMS is specified on the TIMFMT parameter.

***JOB:** The time separator specified for the job at precompile time is used. Use the Display Job (DSPJOB) command to determine the current value for the job.

'!': A colon (:) is used as the time separator.

'.'': A period (.) is used as the time separator.

','': A comma (,) is used as the time separator.

' ': A blank space is used as the time separator.

***BLANK:** A blank is used as the time separator.

Date separator character (DATSEP)

specifies the separator used when accessing date result columns.

***JOB:** The date separator specified for the job at precompile time is used. Use the Display Job (DSPJOB) command to determine the current value for the job.

'/': A slash (/) is used as the date separator.

'.'': A period (.) is used as the date separator.

'-': A hyphen (-) is used as the date separator.

','': A comma (,) is used as the date separator.

' ': A blank space is used as the date separator.

***BLANK:** A blank is used as the date separator.

Replace (REPLACE)

specifies whether a program is created when there is an existing program of the same name in the same library. The value of this parameter is passed to the CRTCLPGM command.

***YES:** A program object is created, and any existing program of the same name in the specified library is moved to QRPLOBJ. The *YES value is passed to the CRTCLPGM command that creates the program object.

***NO:** A program is not created if a program of the same name already exists in the specified library.

Default collection (DFTRDBCOL)

specifies the name of the collection identifier to be used for unqualified names of tables, views, indexes, and SQL packages. This parameter applies only to static SQL statements.

***NONE:** The naming convention specified on the SQLOPT parameter is used.

collection-name: Specify the name of the collection identifier to be used instead of the naming convention specified on the SQLOPT parameter.

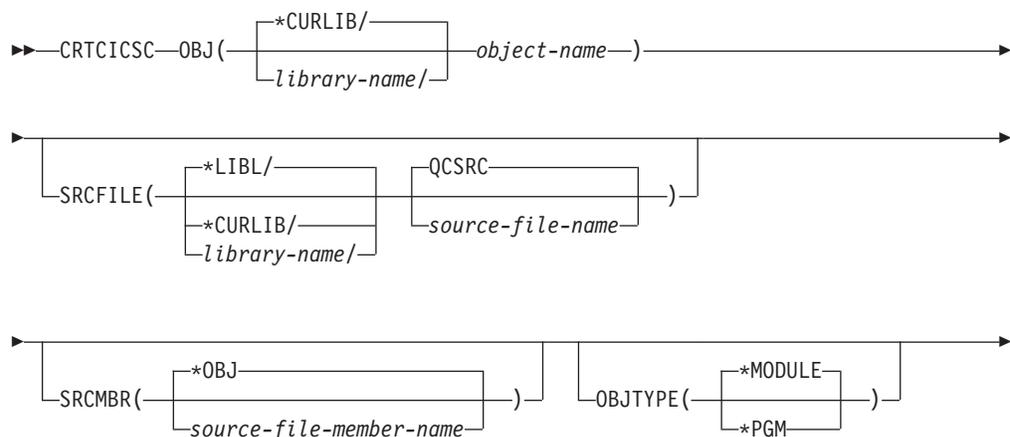
Examples

```
CRTCLPGM PGM(ACCTS/STATS) SRCFILE(ACCTS/ACTIVE)
TEXT('Statistical analysis program')
```

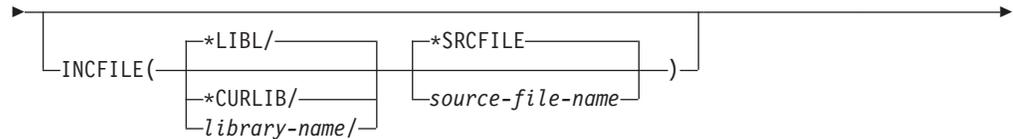
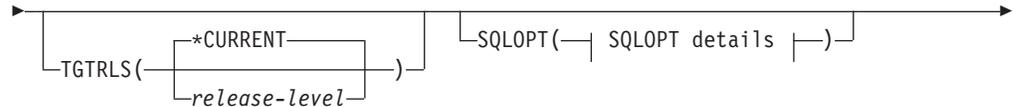
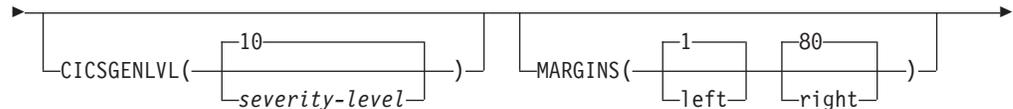
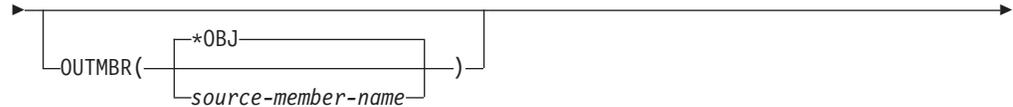
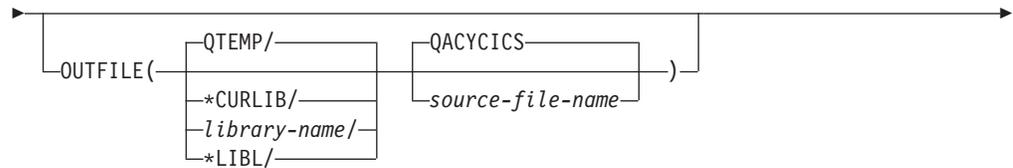
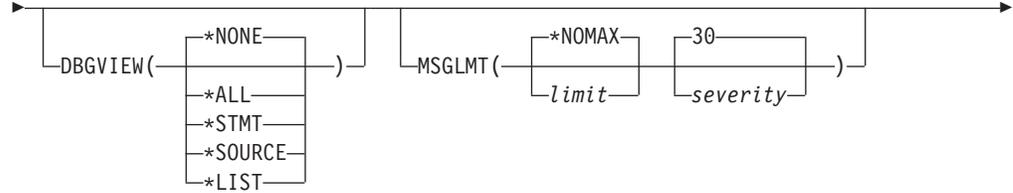
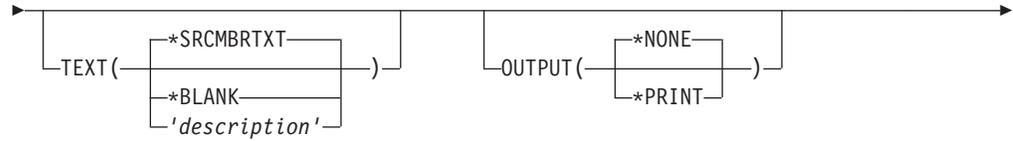
This command runs the CICS precompiler, which precompiles the source in file ACTIVE in library ACCTS and then stores the changed source in the member STATS in file QACYCICS in library QTEMP. If the source member contains any EXEC SQL statements, the member STATS in QACYCICS is used as input to the SQL precompiler. The changed source is stored in the member STATS in file QSQLTEMP in library QTEMP. If the SQL precompiler is needed, this member is used as input to the COBOL compiler; otherwise, the member STATS in file QACYCICS is used as input to the COBOL compiler. The compiled program object, STATS, is placed in library ACCTS.

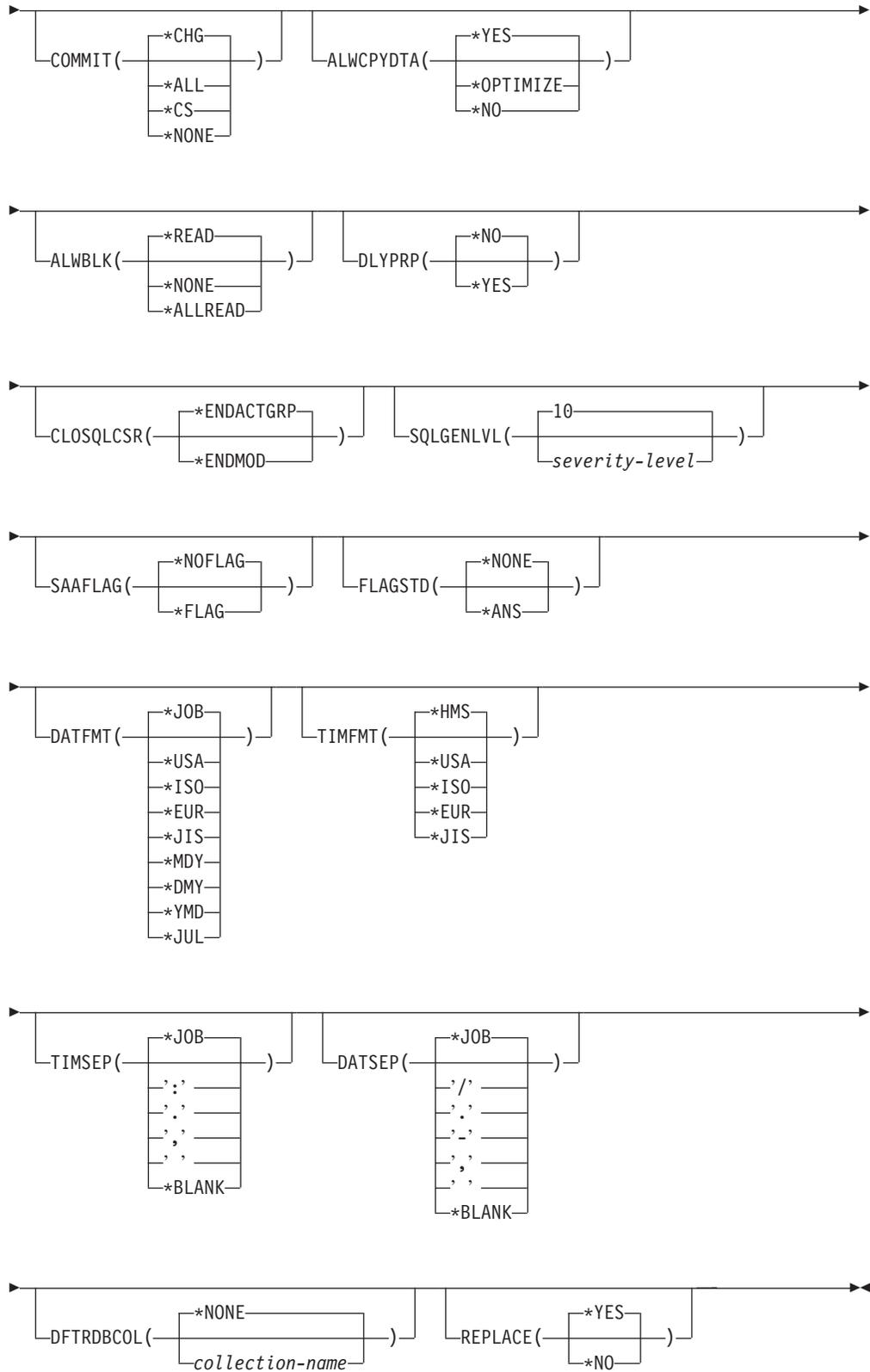
CRTCLPGM

Job: B,I Pgm: B,I REXX: B,I Exec



(1)

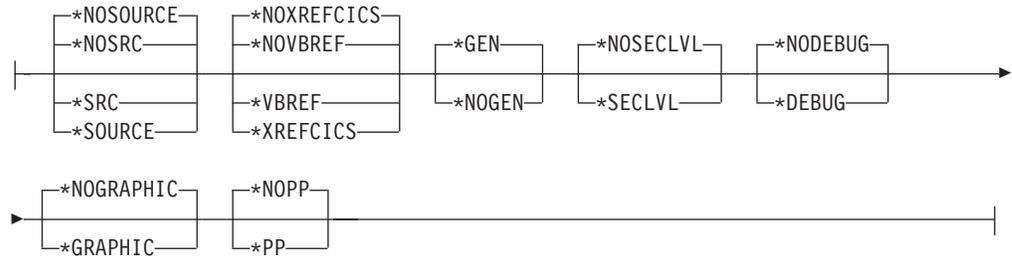




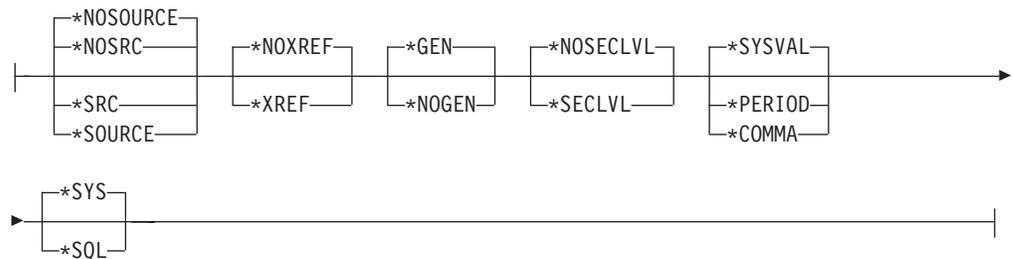
Notes:

- 1 All parameters preceding this point can be specified positionally.

CICSOPT details:



SQLOPT details:



Function

CRTCICSC calls several functions:

- Preprocessing of any #define and #include statements resolved before the translator is called.
- Translating of an ILE C program with embedded CICS commands. If the source program contains any SQL statements, the SQL translator may be called. The resulting translator output is placed into a temporary source member.

Note: It is possible to translate your SQL statements first by running the CRTSQLCI command and specifying the *NOGEN option on the OPTION parameter. This puts the SQL translator output in QSQLTEMP in library QTEMP. You must then run the CRTCICSC command specifying SRCFILE as QTEMP/QSQLTEMP; the SRCMBR name can be obtained by looking in this file for your SQL translated program source. The rest of the parameters on CRTCICSC can be entered as normal.

It is not recommended that applications be compiled in this way but, if necessary, it can be done.

- If the translator stages were successful, what happens next depends on the type of object you wish to produce.
 - If the OBJTYPE parameter is set to *MODULE, and provided that the *NOGEN option has not been specified, CRTCICSC calls CRTCMOD CL command to create a program module. You can then bind the module with others into a run-time program using the CRTPGM CL command.
 - If the OBJTYPE parameter is set to *PGM, CRTCICSC calls the CRTBNDC CL command to create a program module and bound program, in one step. You should use this option only if the program is a simple, one-module program.

Required parameters

Object (OBJ)

specifies the name or qualifier of the object being created.

The name of the object can be qualified by one of the following library values:

***CURLIB:** specifies that the object is to be created in the current library for the job. If no library is specified as the current library for the job, the object is created in QGPL.

library-name: Specify the name of the library in which the object is to be created.

object-name: Specify the name of the object that is to be created.

Attention: If the object name you specify is the same as that of an existing object, and if the REPLACE parameter is specified as *YES (the default), your new object replaces the existing one.

The following parameters are optional for the CICS translator source commands. If you choose not to specify any of the following keywords or their values, the defaults are used. SQLOPT lists those options that apply to SQL. For more information, see the Database and file systems topic in the iSeries Information Center.

Source file (SRCFILE)

specifies the qualified name of the source file that contains the ILE C source code with the EXEC CICS or EXEC SQL statements.

The source file name can be qualified by one of the following library values:

***LIBL:** Specifies that the library list is used to locate the source file.

***CURLIB:** Specifies that the current library for the job is used to locate the source file. If no library is specified as the current library for the job, the QGPL library is used.

library-name: Specify the library where the source file is located.

QCSRC: specifies that, if a ILE C source file name is not specified, the supplied source file QCSRC contains the ILE C source.

source-file-name: Specify the name of the source file that contains the ILE C source. The source file can be a database file, device file, or an inline data file.

Source member (SRCMBR)

specifies the name of the source file member that contains the ILE C source. This parameter is specified only if the source file name in the SRCFILE parameter is that of a database file.

***OBJ:** Specifies that the ILE C source is in the source file member that has the same member name as that specified in the OBJ parameter of this command.

source-file-member-name: Specify the name of the source file member that contains the ILE C source.

Compile type (OBJTYPE)

specifies the type of object to be used.

***MODULE:** Specifies that the translator is to issue the CRTCMOD command to create a module.

Note: You can then bind the module with others into a run-time program using the CRTPGM CL command.

***PGM:** Specifies that the translator is to issue the CRTBNDC command to create a module and bound program in one step.

Text description (TEXT)

specifies the text relating to the program.

***SRCMBRTXT:** Specifies that the text is to be taken from the source file member being used to create the program. Text for a database source member can be added or changed by using the Source Entry Utility (STRSEU) command, or by using either the Add Physical File Member (ADDPFM) or the Change Physical File Member (CHGPFM) command. If the source file is an inline file or a device file, the text is blank.

***BLANK:** No text is specified.

'description': Specify no more than 50 characters of text, enclosed by apostrophes.

Compiler output (OUTPUT)

specifies whether a compiler listing is to be generated.

The possible values are:

***NONE:** Specifies that a compiler listing is not to be generated.

***PRINT:** Specifies that a compiler listing is to be generated.

Debugging view (DBGVIEW)

specifies which level of debugging is available for the compiled module or program, and which source views are available for source-level debugging.

The possible values are:

***NONE:** Disables all of the debug options for debugging the compiled module or program.

***ALL:** Enables all of the debug options for debugging the compiled module or program, and produces a source view, as well as a listing view.

***STMT:** Allows the compiled module or program to be debugged using program statement numbers and symbolic identifiers.

***SOURCE:** Generates the source view for debugging the compiled module or program.

***LIST:** Generates the listing view for debugging the compiled module or program.

Compiler messages (MSGLMT)

Specifies the maximum number of messages that can occur before the C compilation stops.

message-limit: Specifies the number of messages that can occur.

***NOMAX:** Compilation continues regardless of the number of messages that have occurred at the specified message severity level. This is the default.

maximum-message-limit: Specify the maximum number of messages that can occur at, or above, the specified message severity level, before compilation stops. The valid range is 1 to 32767.

message-severity: Specifies the message severity that can occur before compilation stops.

0: Specifies that a *message-limit* of messages at severity 0 or above can occur before compilation stops.

10: Specifies that a *message-limit* of messages at severity 10 or above can occur before compilation stops.

30: Specifies that a *message-limit* of messages at severity 30 or above can occur before compilation stops.

CICS options (CICSOPT)

specifies whether one or more of the following options are to be used when the source is translated. If an option is specified more than once, or if two options conflict, the last option specified is used. If an option is not specified, the default is used.

Source listing options

***NOSRC** or ***NOSOURCE:** Specifies that the CICS translator is not to produce a source listing.

***SRC** or ***SOURCE:** Specifies that the CICS translator is to produce a source listing and error messages.

Cross-reference options

***NOVBREF** or ***NOXREFCICS:** Specifies that the CICS translator is not to produce a cross-reference of EXEC CICS names.

***VBREF** or ***XREFCICS:** Specifies that the CICS translator is to produce a cross-reference between EXEC CICS names in the program and the statement numbers in the program that refer to them.

Program creation options

***GEN:** Specifies that the SQL precompiler or ILE C compiler is to be called after a successful CICS translation.

Note: The SQL precompiler is to be called only if, during the CICS translation stage, any EXEC SQL statements were found in the ILE C source code being CICS translated.

***NOGEN:** Specifies that the compilation is to terminate at the end of the CICS translation.

Second-level help text

***NOSECLVL:** Specifies that no second-level help text is to be printed.

***SECLVL:** Specifies that the second-level help text is to be printed.

Note: The first-level help text is printed automatically each time an error occurs.

Debug options

***NODEBUG:** Specifies that the CICS translator is not to produce code that will be passed through to CICS to be displayed by the CICS execution diagnostic facility (EDF).

***DEBUG:** Specifies that the CICS translator is to produce code that will be passed through to CICS to be displayed by EDF.

DBCS option

***NOGRAPHIC:** specifies that the translator is not to accept double-byte data.

***GRAPHIC:** specifies that the translator is to accept double-byte data. This data can only appear in columns 7 through 72.

Preprocessing option

***NOPP:** specifies that the ILE C program is not to be preprocessed.

***PP:** specifies that the ILE C program is to be preprocessed.

CICS output file (OUTFILE)

specifies the qualified name of the intermediate precompiler output.

The possible library values are:

QTEMP: The supplied source file QTEMP is used.

***CURLIB:** The current library is searched. If no library is specified as the current library for the job, the QGPL library is used.

library-name: Specify the name of the library where to generate the intermediate precompiler output.

***LIBL:** All libraries in the user and system portions of the job's library are searched.

QACYCICS: If an output source file name is not specified, the supplied source file QACYCICS will contain the intermediate precompiler output.

source-file-name: Specify the name of the source file to contain the intermediate precompiler output.

CICS output member (OUTMBR)

specifies the name of the source file member that contains the intermediate precompiler output. If this parameter is not specified, the object name on the OBJ parameter is used.

***OBJ:** Specifies that the intermediate precompiler output has the same member name as that specified in the OBJ parameter.

source-member-name: specifies that the name of the source file member to which the intermediate precompiler output should be copied.

CICS message level (CICSGENLVL)

specifies a level of CICS translator errors. If errors occur with a severity level greater than the value specified in this parameter, either the SQL precompiler is not called (if the CICS translator located any SQL in the source) or the compiler is not called.

10: If a severity-level value is not specified, the default severity level is 10.

severity-level: Specify a number in the range 10 through 40. Some suggested values are listed below:

10 The level value for warnings.

20 The level value for general error messages.

30 The level value for serious error messages.

40 The level value for system-detected error messages.

Note: The value of CICSGENLVL applies only to messages generated as a result of CICS translation errors. The specified CICSGENLVL value is **not** passed to the SQL precompiler or the compiler.

Source margins (MARGINS)

specifies the part of the translator input record that contains source text.

The possible values are:

left-margin: Specify the left-hand, beginning position for the statements. Valid values range from 1 through 90.

right-margin: Specify the right-hand, ending position for the statements. Valid values range from 10 through 100.

Target release (TGTRLS)

specifies the release level of the CICS/400 system on which you intend to use the object being created.

You can specify an exact release level in the format VxRxMx, where Vx is the version, Rx is the release, and Mx is the modification level. For example, V5R2M0 would be version 5, release 2, modification 0.

The possible values are:

***CURRENT:** Specifies that the object is to be used on the release of CICS/400 currently running on your system. For example, if V5R3M0 is running on the system, *CURRENT means that you intend to use the object on a system with V5R3M0 installed.

***PRV:** The object is to be used on the previous release of CICS/400. If V5R3M0 is running on your system, *PRV means that you intend to use the object on a system with V5R2M0 installed.

release-level: Specify the release level in the format VxRxMx. The object can be used on a system with the specified release.

Valid values depend on the current version, release, and modification level, and they change with each new release. The valid values for this release are: V5R3M0, V5R2M0 and V5R1M0.

SQL options (SQLOPT)

specifies whether one or more of the following options are to be used when the source is SQL precompiled. If an option is specified more than once, or if two options conflict, the last option specified is used. If an option is not specified, the default is used.

Source listing options

Important note:

Due to a precompiler restriction, an SQL source listing is always produced, no matter which source listing option you select. Even if you specify *NOSRC or *NOSOURCE, a source listing is still produced. You are recommended not to specify an SQL source listing option at all, as this is liable to change in a future release of CICS/400.

***NOSRC or *NOSOURCE:** Specifies that a source listing is not to be produced by the SQL precompiler.

***SRC or *SOURCE:** Specifies that a source listing, consisting of all the source input and error messages, is to be produced by the SQL precompiler.

Cross-reference options

***NOXREF:** Specifies that the SQL precompiler is not to produce a cross-reference of EXEC SQL names.

***XREF:** Specifies that the SQL precompiler is to produce a cross-reference between items in the program and the numbers of the statements in the program that refer to these items.

Program creation options

***GEN:** Specifies that the compiler will be called after a successful SQL precompilation.

***NOGEN:** specifies that compilation is to terminate at the end of the SQL precompilation.

Second-level help text

***NOSECLVL:** Specifies that no second-level help text is to be printed.

***SECLVL:** Specifies that the second-level help text is to be printed.

Note: The first-level help text is printed automatically each time an error occurs.

Decimal point options

***SYSVAL:** Specifies that the value to be used as the decimal point is from the QDECFMT system value.

Note: If QDECFMT specifies that the value to be used as the decimal point is a comma, any numeric constants in lists (such as in the SELECT clause or the VALUES clause) must be separated by a comma followed by a blank. For example, VALUES(1,1, 2,23, 4,1) is equivalent to VALUES(1.1,2.23,4.1) in which the decimal point is a period.

***PERIOD:** Specifies that the value to be used as a decimal point is a period.

***COMMA:** Specifies that the value to be used as a decimal point is a comma.

Note: Any numeric constants in lists (such as in the SELECT clause, the VALUES clause, and so on) must be separated by a comma followed by a blank. For example, VALUES(1,1, 2,23, 4,1) is equivalent to VALUES(1.1,2.23,4.1), where the decimal point is the period.

Naming convention options

***SYS:** Specifies that the OS/400 system naming convention is to be used (*library-name/file-name*).

***SQL:** Specifies that the SQL naming convention is to be used (*collection-name.table-name*).

SQL INCLUDE file (INCFILE)

specifies the qualified name of the source file that contains the members included in the program with any SQL INCLUDE statement.

The possible library values are:

***LIBL:** Specifies that the library list is used to locate the source file.

***CURLIB:** Specifies that the current library for the job is used to locate the source file. If no current library entry exists in the library list, QGPL is used.

library-name: Specify the name of the library where the source file is located.

***SRCFILE:** Specifies the qualified source file you specified in the SRCFILE parameter that contains the source file members specified on any SQL INCLUDE statement.

source-file-name: Specify the name of the source file that contains the source file members specified on any SQL INCLUDE statement. The record length of the source file you specify here must be no less than the record length of the source file you specified for the SRCFILE parameter.

SQL commitment control (COMMIT)

specifies whether SQL statements in the compiled program are run under commitment control. Files referred to in the host language source are not affected by this parameter. Only SQL tables, views, and SQL packages referred to in SQL statements are affected.

***CHG:** Specifies that the objects referred to in SQL COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements and the rows updated, deleted, and inserted are locked until the end of the unit of work (transaction). Uncommitted changes in other jobs can be seen.

***ALL:** Specifies that the objects referred to in SQL COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements and the rows updated, deleted, and inserted are locked until the end of the unit of work (transaction). Uncommitted changes in other jobs cannot be seen.

***CS:** Specifies that the objects referred to in SQL COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements and the rows updated, deleted, and inserted are locked until the end of the unit of work (transaction). A row that is selected, but not updated, is locked until the next row is selected. Uncommitted changes in other jobs cannot be seen.

***NONE:** Specifies that commitment control is not used. COMMIT and ROLLBACK statements are not allowed. SQL COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements and the rows updated, deleted, and inserted are locked until the end of the unit of work (transaction). A row that is selected, but not updated, is locked until the next row is selected. Uncommitted changes in other jobs cannot be seen.

Note: If *CHG, *CS, or *ALL is specified, DROP COLLECTION cannot be included in the application. The default for this parameter for the Start SQL (STRSQL) command is *NONE.

SQL allow copy of data (ALWCPYDTA)

specifies whether a copy of the data is allowed in a SELECT statement.

***YES:** A copy of the data can only be used, if necessary, to run a SELECT statement.

***OPTIMIZE:** The system chooses whether or not to use the data retrieved directly from the database or to use a copy of the data. The decision is based on which choice will provide the best performance.

This value decreases the time required for the total query. Because the copy of the data must be made before returning the first row of the result table, the time to retrieve the first row may be increased.

Note: If *CS or *ALL is specified on the COMMIT parameter, SQL run time ignores this parameter and uses current data.

***NO:** A copy of the data is not allowed. This option could return a negative SQLCODE if the clauses in the SELECT statement require a copy of the data. If the SELECT statement runs successfully, then current data was used.

SQL allow blocking (ALWBLK)

specifies whether the database manager can use record blocking, and the extent to which blocking can be used for read-only cursors.

***READ:** Records are blocked for read-only retrieval of data for cursors when:

- *NONE is specified on the COMMIT parameter, to indicate that commitment control is not used.
- The cursor is declared with a FOR FETCH ONLY clause or there are no dynamic statements that could run a positioned UPDATE or DELETE statement for the cursor.

Specifying *READ can improve the overall performance of queries that meet the above conditions and retrieve a large number of records.

***NONE:** Rows are not blocked for retrieval of data for cursors. Specifying *NONE:

- Guarantees that the data retrieved is current.
- May reduce the amount of time required to retrieve the first row of data for a query.
- Stops the database manager from retrieving a block of data rows that is not used by the program when only the first few rows of a query are retrieved before the query is closed.
- Can degrade the overall performance of a query that retrieves a large number of rows.

***ALLREAD:** Rows are blocked for read-only cursors if *NONE or *CHG is specified on the COMMIT parameter. All read-only cursors in a program are opened for read-only processing even though there may be EXECUTE statements in the program. Specifying *ALLREAD:

- Allows record blocking under commitment control. Specifying *READ does not.
- Improves the performance of almost all read-only cursors in programs, but limits queries in the following ways:
 - A ROLLBACK statement in host languages, or the ROLLBACK HOLD SQL statement, does not position the read-only cursor again when *ALLREAD is specified.
 - Dynamic running of a positioned UPDATE or DELETE statement (for example, EXECUTE IMMEDIATE), cannot be used to update a row in a cursor unless the DECLARE statement for the cursor includes the FOR UPDATE clause.

SQL delay prepare (DLYPRP)

specifies whether the dynamic statement validation for a PREPARE statement is delayed until an OPEN, EXECUTE, or DESCRIBE statement is run. Delaying validation improves performance by eliminating duplicate validation.

***NO:** Dynamic statement validation is not delayed. When the dynamic statement is prepared, the access plan is validated. When the dynamic statement is used in an OPEN or EXECUTE statement, the access plan is revalidated. Because the authority or the existence of objects referred to by the dynamic statement may change, you must still check the SQLCODE or SQLSTATE after issuing the OPEN or EXECUTE statement, to ensure that the dynamic statement is still valid.

***YES:** Dynamic statement validation is delayed until the dynamic statement is used in an OPEN, EXECUTE, or DESCRIBE SQL statement. When the dynamic statement is used, the validation is completed and an access plan is built. If you specify *YES on this parameter for precompiled programs, you should check the SQLCODE and SQLSTATE after running an OPEN, EXECUTE, or DESCRIBE statement to ensure that the dynamic statement is valid. If you specify *YES, performance is not improved if the INTO clause is used on the PREPARE statement or if a DESCRIBE statement uses the dynamic statement before an OPEN is issued for the statement.

SQL close cursor (CLOSQLCSR)

specifies when SQL cursors are to be explicitly closed; when SQL prepared statements are to be implicitly disregarded; and when LOCK TABLE locks are to be released. SQL cursors are explicitly closed by issuing the CLOSE COMMIT (without HOLD), or ROLLBACK (without HOLD) SQL statements.

***ENDACTGRP:** Specifies that SQL cursors are to be closed, SQL prepared statements are to be released, and LOCK TABLE locks are to be released when the activation group ends.

***ENDMOD:** Specifies that SQL cursors are to be closed and SQL prepared statements are to be discarded when the module is exited. LOCK TABLE locks are to be released when the activation group ends.

SQL message level (SQLGENLVL)

specifies a level of SQL precompiler errors. If errors occur with a severity level greater than the value specified in this parameter, the compiler is not called.

10: If no value is specified, the default severity level is 10.

severity-level: Specify a number in the range 10 through 40. Some suggested values are listed below:

- 10 The level value for warnings.
- 20 The level value for general error messages.
- 30 The level value for serious error messages.
- 40 The level value for system-detected error messages.

Note: The value of SQLGENLVL only applies to messages generated as a result of SQL precompilation errors. The specified SQLGENLVL value is **not** passed to the compiler.

SQL SAA flagging (SAAFLAG)

specifies whether SQL statements that do not conform to SAA Level 2 Database standards are flagged.

***NOFLAG:** Specifies that the precompiler will *not* check for conformity to SAA standards.

***FLAG:** Specifies that the precompiler will check for conformity to SAA standards.

SQL ANS flagging (FLAGSTD)

specifies whether nonstandard statements are flagged. This parameter allows you to flag SQL statements to verify that they conform to ANSI X3.135-1-1989, ANSI X3.168-1989, ISO 9075-1989, and FIPS 127.1 standards.

***NONE:** Specifies that the precompiler will *not* check for conformity to ANSI standards.

***ANS:** Specifies that the precompiler will check for conformity to ANSI standards.

SQL date format (DATFMT)

specifies the format used when accessing date result columns. All output date fields are returned in the specified format. For input date strings, the specified value is used to determine whether the date is in a valid format.

Note: An input date string that uses format ***USA**, ***ISO**, ***EUR**, or ***JIS** is always valid. If you connect to a relational database that is on a system that is not an OS/400 system, ***USA**, ***ISO**, ***EUR**, or ***JIS** must be used.

***JOB:** Specifies the format used for the job at precompilation. Use the Display Job (DSPJOB) command to determine the current date format for the job.

***USA:** The United States date format mm/dd/yyyy is used.

***ISO:** The International Organization for Standardization (ISO) date format yyyy-mm-dd is used.

***EUR:** The European date format dd.mm.yyyy is used.

***JIS:** The Japanese Industrial Standard date format yyyy-mm-dd is used.

***MDY:** The date format mm/dd/yy is used.

***DMY:** The date format dd/mm/yy is used.

***YMD:** The date format yy/mm/dd is used.

***JUL:** The Julian date format yy/ddd is used.

SQL time format (TIMFMT)

specifies the format used when accessing time result columns. All output time fields are returned in the specified format. For input time strings, the specified value is used to determine whether the time is specified in a valid format.

Note: An input time string that uses the format ***USA**, ***ISO**, ***EUR**, or ***JIS** is always valid. If you connect to a relational database that is on a system that is not an OS/400 system, the time format must be ***USA**, ***ISO**, ***EUR**, ***JIS**, or ***HMS**, with a time separator of colon or period.

***HMS:** The hh:mm:ss format is used.

***USA:** The United States time format hh:mm xx is used, where xx is AM or PM.

***ISO:** The International Organization for Standardization (ISO) time format hh.mm.ss is used.

***EUR:** The European time format hh.mm.ss is used.

***JIS:** The Japanese Industrial standard time format hh:mm:ss is used.

SQL time separator (TIMSEP)

specifies the separator used when accessing time result columns.

Note: This parameter applies only when *HMS is specified on the TIMFMT parameter.

***JOB:** The time separator specified for the job at precompile time is used. Use the Display Job (DSPJOB) command to determine the current value for the job.

' ': A colon (:) is used as the time separator.

'.' : A period (.) is used as the time separator.

',' : A comma (,) is used as the time separator.

' ' : A blank space is used as the time separator.

***BLANK:** A blank is used as the time separator.

SQL date separator (DATSEP)

specifies the separator used when accessing date result columns.

***JOB:** The date separator specified for the job at precompile time is used. Use the Display Job (DSPJOB) command to determine the current value for the job.

'/': A slash (/) is used as the date separator.

'.' : A period (.) is used as the date separator.

'-' : A hyphen (-) is used as the date separator.

',' : A comma (,) is used as the date separator.

' ' : A blank space is used as the date separator.

***BLANK:** A blank is used as the date separator.

SQL default collection (DFTRDBCOL)

specifies the name of the collection identifier to be used for unqualified names of tables, views, indexes, and SQL packages. This parameter applies only to static SQL statements.

***NONE:** The naming convention specified on the SQLOPT parameter is used.

collection-name: Specify the name of the collection identifier to be used instead of the naming convention specified on the SQLOPT parameter.

Replace object (REPLACE)

specifies whether a CICS module or program is to be created when there is an existing program of the same name in the same library. The value of this parameter is passed to either the CRTCMOD or the CRTBNDC command.

***YES:** Specifies that a CICS module, program, or service program is to be created, and any existing object of the same name and type in the specified library is to be moved to QRPLOBJ. The *YES value is passed to either the CRTCMOD or the CRTBNDC command.

***NO:** Specifies that a CICS module or program is not to be created if an object of the same name and type already exists in the specified library.

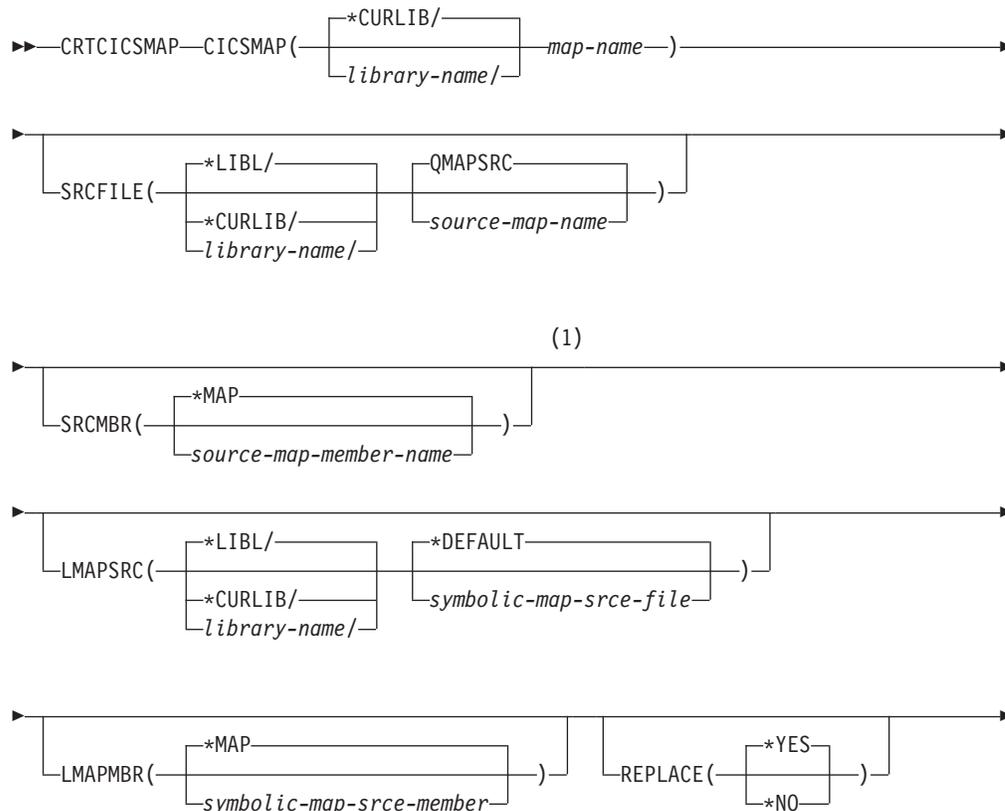
Examples

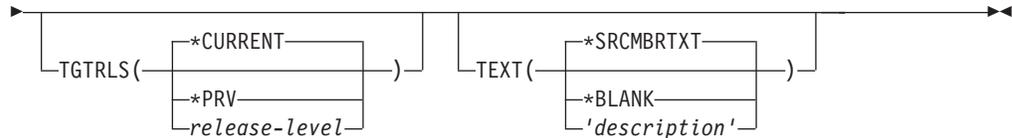
```
CRTCICSC OBJ(ACCTS/STATS) SRCFILE(ACCTS/ACTIVE)
TEXT('Statistical analysis program')
```

This command runs the CICS translator, which translates the source in file ACTIVE in library ACCTS and then stores the changed source in the member STATS in file QACYCICS in library QTEMP. If the source member contains any EXEC SQL statements, the member STATS in QACYCICS is used as input to the SQL translator. The changed source is stored in the member STATS in file QSQLTEMP in library QTEMP. If the SQL translator is needed, this member is used as input to the SQL compiler; otherwise, the member STATS in file QACYCICS is used as input to the ILE C compiler. The compiled program object, STATS, is placed in library ACCTS.

CRTCICSMAP

Job: B,I Pgm: B,I REXX: B,I Exec





Notes:

- 1 All parameters preceding this point can be specified positionally.

Function

The Create CICS MAP (CRTCICSMAP) command allows you to create BMS physical and symbolic maps.

Required parameters

Map object name (CICSMAP)

specifies the qualified name of the generated 3270 or 5250 CICS BMS map.

The possible library values are:

***CURLIB:** Specifies the library in which the output physical map will reside.

library-name: Specify the name of the library in which the physical map is located.

map-name: Specify the name of the generated map file.

Note: If a suffix operand is specified in the input BMS macro source, the suffix is attached automatically to the name.

Map source file (SRCFILE)

specifies the qualified name of the input BMS map source from which the 3270 or 5250 maps are created.

The possible library values are:

***LIBL:** All libraries in the user and system portions of the job's library list are searched.

***CURLIB:** The current library is searched. If no library is specified as the current library for the job, the QGPL library is used.

library-name: Specify the name of the library where the BMS map source file is located.

QMAPSRC: If a BMS map source file is not specified, the supplied source file QMAPSRC contains the map source.

source-map-name: Specify the name of the BMS map source from which the physical map file is created.

Map source member name (SRCMBR)

specifies the name of the source file that contains the CICS map source.

***MAP:** Specifies that the CICS map source is in the source file member that has the same member name as that specified in the CICSMAP parameter.

source-map-member-name: Specify the name of the source file member that contains the CICS map source.

Symbolic map source (LMAPSRC)

specifies the qualified name of the output source file that will contain the symbolic map as either a COBOL copybook or a C header file.

The possible library values are:

***LIBL:** All libraries in the user and system portions of the job's library list are searched.

***CURLIB:** The current library is searched. If no library is specified as the current library for the job, the QGPL library is searched.

library-name: Specify the name of the library where the symbolic map will be generated.

The possible symbolic source file values are:

***DEFAULT:** The source file name defaults according to the value specified on the LANG operand in the DFHMSD macro in the map source:

- If LANG=COBOL, a source file name of QLBSRC will be used.
- If LANG=C, a source file name of H will be used.
- If no LANG operand is specified in the map source, COBOL will be assumed as the source file name will be QLBSRC.

symbolic-map-srce-file: Specify the name of the source file to which the language-specific source statements are to be copied.

Symbolic map source member (LMAPMBR)

specifies the name of the source file member that contains the symbolic map source. If this parameter is not specified, the map name specified in the CICS MAP parameter is used.

Note: You should take care when using this parameter that you do not specify the fully-qualified name of a source member that already exists. If you do, the existing source member will be replaced.

***MAP:** Specifies that the symbolic map source member name is the same as the name specified in the CICS MAP parameter of the PPT.

symbolic-map-srce-member: Specify the name of the source file member to which the symbolic map source statements are to be copied.

Replace output objects (REPLACE)

specifies whether a CICS BMS map is created when there is an existing CICS BMS map of the same name in the same library.

***YES:** A CICS BMS map is created. If there is a CICS BMS map of the same name in the specified library, it is replaced.

***NO:** A CICS BMS map is not created if a CICS BMS map of the same name already exists in the specified library.

Target release (TGTRLS)

specifies the release level of the CICS/400 system on which you intend to use the object being created.

You can specify an exact release level in the format VxRxMx, where Vx is the version, Rx is the release, and Mx is the modification level. For example, V5R2M0 would be version 5, release 2, modification 0.

The possible values are:

***CURRENT:** Specifies that the object is to be used on the release of CICS/400 currently running on your system. For example, if V5R2M0 is running on the system, *CURRENT means that you intend to use the object on a system with V5R2M0 installed.

***PRV:** The object is to be used on the previous release of CICS/400. If V5R2M0 is running on your system, *PRV means that you intend to use the object on a system with V5R1M0 installed.

release-level: Specify the release level in the format VxRxMx. The object can be used on a system with the specified release.

Valid values depend on the current version, release, and modification level, and they change with each new release.

Text (TEXT)

specifies the text that briefly describes the new CICS definition.

***SRCMBRTXT:** The text is taken from the source file member being used to create the CICS map. Text for a database source member can be added or changed using the Source Entry Utility (STRSEU) command, or by using either the Add Physical File Member (ADDPFM) or the Change Physical File Member (CHGPFM) command. If the source file is an inline file or a device file, the text is blank.

***BLANK:** No text is specified.

description: Specify no more than 50 characters of text, enclosed by apostrophes.

Examples

```
CRTCICSMAP CICS MAP(MYLIB/MYMAP) +
SRCFILE(MYLIB/QMAPSRC) +
SRCMBR(*MAP) LMAPSRC(MYLIB/QLBLSRC) +
LMAPMBR(*MAP) TEXT('My test BMS map.')
```

This command generates a 3270 or 5250 CICS BMS physical map in library MYLIB with name MYMAP, and a symbolic map in member MYMAP, in file QLBLSRC, on library MYLIB.

Chapter 31. Programming reference

Introduction to EXEC CICS commands

This part of the book shows the syntax of each command, describes the purpose and format of each command and its options, and gives a list of the conditions that can arise during the execution of each command. There are two groups of command:

- Application programming commands.
- System programming commands that are used for monitoring and changing certain parameters within the CICS system, and for resource administration.

Both types of command are executed within application programs. The main difference between them is that system programming commands affect system-wide resources. Because system programming commands deal with CICS resources, you will not want everyone to be able to change parameters within the system. Use of the system programming commands is normally limited to the system administrator.

Command format

The general format of a CICS command is **EXECUTE CICS** (or **EXEC CICS**) followed by the name of the required function, and possibly by one or more **options**, as follows:
where:

```
EXEC CICS function  
      [option[(argument)]]...
```

- function** describes the operation required (for example, READ).
- option** describes any of the many optional facilities available with each command. Some options are followed by an argument in parentheses. You can write options (including those that require arguments) in any order.
- argument** is a value such as “data-value” or “data-area”. A “data-value” can be a constant. This means that an argument that provides data to CICS is generally a “data-value”. However, an argument that receives data from CICS is generally either a “data-area” or a “ptr-ref”. For further information, see “Argument values” on page 307..

An example of a CICS command is as follows:

```
EXEC CICS READ  
      FILE('FILEA')  
      INTO(RECORD)  
      RIDFLD(KEYNUM)  
      UPDATE
```

You must add the appropriate end-of-command delimiter: see “CICS syntax notation used” for further information.

CICS syntax notation used

In the CICS documentation, the syntax of application programming commands is shown in a standard way. The “EXEC CICS” that always precedes each command’s function keyword is not included; nor is the delimiter that you must code at the end of each command. For COBOL programs, this is “END-EXEC”. For ILE C programs, this is a semicolon (;).

In examples, EXEC CICS is included, but the command delimiter is omitted.

Table 12 explains the command syntax conventions. You interpret the syntax by following the arrows from left to right, and from top to bottom, along the main path line.

Table 12. Command syntax conventions

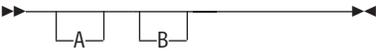
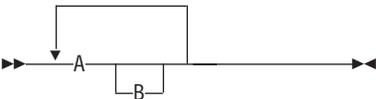
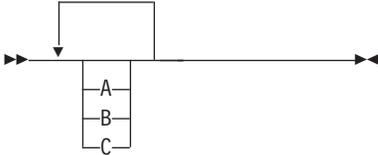
Symbol	Meaning
	Required items appear on the main path line.
	If there is more than one required item to choose from, the items are stacked vertically. This is a set of alternatives—one of which you <i>must</i> code.
	Optional items appear below the main path line.
	If there is more than one optional item to choose from, the items are stacked vertically below the main path line. This is a set of alternatives—one of which you <i>may</i> code.
	If one item in a set of alternatives is the default, this item appears above the main path line and all other items are stacked vertically below the line.
	An arrow returning to the left above items on the main path line means that the items can be repeated. Such items may be either required or optional.

Table 12. Command syntax conventions (continued)

Symbol	Meaning
	An arrow returning to the left above a set of items means that more than one item can be selected.
	Use with the named section in place of its name.
<p>Name:</p> 	
Punctuation and uppercase characters	Code exactly as shown.
Lowercase characters appearing like <i>this</i>	Code your own text, as appropriate.

For example, with FILE(*name*) you must code FILE and () unchanged, but are free to code any valid text string for the name of your file.

Argument values

The parenthesized argument values that follow options in a CICS command are specified as follows:

- data-value
- data-area
- CICS-value data area (or cvda)
- pointer-value (or ptr-value)
- pointer-ref (or ptr-ref)
- name
- label
- hhmmss

Arguments can be used as either of the following:

Sender

Data is passed from the application to CICS. All argument types can be used as senders, but data-values and pointer-values are preferred to data-areas and pointer-references as sending arguments.

Receiver

Data is passed from CICS to the application. Only data-areas, CVDAs, and pointer-references can be receivers.

Note: An argument used to send and receive data should be defined as a receiver.

COBOL argument values

Conversion note: OS/VS COBOL and COBOL II treat COMP fields as if the BINARY attribute had been specified, whereas COBOL/400 treats these fields as if the COMP-3 attribute had been specified. This is because COMP fields default to the numeric representation that is most efficient for the system concerned. If you are porting applications from other CICS platforms, you must check for references to COMP and USAGE IS COMPUTATIONAL because these may be interpreted differently on the source and target systems; then change all COMP fields to BINARY.

The argument values can be replaced as follows:

- “data-value” can be replaced by any COBOL data name of the correct data type for the argument, or by a constant that can be converted to the correct type for the argument. The data type can be specified as one of the following:
 - Halfword binary—PIC S9(4) COMP
 - Fullword binary—PIC S9(8) COMP
 - Character string—PIC X(n), where “n” is the number of bytes

“data-value” includes “data-area” as a subset.

- “data-area” can be replaced by any COBOL data name of the correct data type for the argument. The data type can be specified as one of the following:
 - Halfword binary—PIC S9(4) COMP
 - Fullword binary—PIC S9(8) COMP
 - Character string—PIC X(n), where “n” is the number of bytes

If the data type is unspecified, the data-area can refer to an elementary or group item.

- “cvda” is described in “CICS-value data areas (CVDAs)” on page 309.
- “pointer-value” or “ptr-value” can be replaced by any COBOL/400 data name declared as a POINTER variable. Consult the *COBOL/400 Reference* for details.
- “pointer-ref” or “ptr-ref” can be replaced by any COBOL/400 data name declared as a POINTER variable. Consult the *COBOL/400 Reference* for details. Fields used as pointers in mainframe CICS are usually defined as PIC S9(8) COMP. Remember to change this to USAGE IS POINTER for COBOL/400.
- “name” can be replaced by either of the following:
 - A character string in single quotation marks (that is, a nonnumeric literal). If this is shorter than the required length, it is padded on the right with blanks.
 - A COBOL data-area with a length equal to the length required for the name. The value in the data-area is the name to be used by the argument. If the data-area is shorter than the required length, the excess characters are undefined.
- “label” can be replaced by any COBOL paragraph name or section name.
- “hhmmss” can be replaced by a decimal constant or by any COBOL data name of the data type PIC S9(7) COMP-3. The value must be of the form 0HHMMSS+, where:

HH represents hours from 00 through 99

MM represents minutes from 00 through 59

SS represents seconds from 00 through 59

ILE C argument values

The argument values can be replaced as follows:

- “data-value” can be replaced by any ILE C expression that can be converted to the correct data type for the argument. The data type can be specified as one of the following:
 - Halfword binary—short int
 - Fullword binary—long int
 - Character string—char[n], where “n” is the number of bytes

“data-value” includes “data-area” as a subset.

- “data-area” can be replaced by any ILE C data reference that has the correct data type for the argument. The data type can be specified as one of the following:
 - Halfword binary—short int
 - Fullword binary—long int
 - Character string—char[n], where “n” is the number of bytes

If the data type is unspecified, the data-area can refer to a scalar data type, array, or structure. The reference must be to contiguous storage.

- “cvda” is described in “CICS-value data areas (CVDAs).”
- “pointer-value” or “ptr-value” (which includes “pointer-ref” as a subset) can be replaced by any ILE C expression that can be converted to an address.
- “pointer-ref” or “ptr-ref” can be replaced by any ILE C pointer type reference.
- “name” can be replaced by either of the following:
 - A character string in double quotation marks (that is, a literal constant).
 - Any ILE C expression or data reference whose value can be converted to a character array with a length equal to the maximum length allowed for the name. The value of the character array is the name to be used by the argument.
- “label” is not supported for the ILE C language.
- “hhmmss” can be replaced by a decimal constant or an expression that can be converted to a long int value. The value must be of the form HHMMSS, where:

HH represents hours from 00 through 99

MM represents minutes from 00 through 59

SS represents seconds from 00 through 59

For example, the int value 145359 represents 14 hours, 53 minutes, and 59 seconds.

CICS-value data areas (CVDAs)

There are options on some CICS commands that give the status or definition of a resource. For example, the STATE option on the CONNECT PROCESS command returns the state of the current conversation. Also, there are many options on INQUIRE and SET commands that refer to resource status or definition. These options all have values that are CICS-supplied and are known as **CICS-value data areas**. They are shown in the syntax of commands with “cvda” in parentheses after the option name. In other contexts, uppercase letters (CVDA) are used.

You *pass* a CVDA value to CICS in one of two ways, depending on the circumstances:

Flexible form

You can use the DFHVALUE translator built-in function to assign a CVDA value to the fullword binary data-area that is specified on the command. In COBOL, this might be:

```
MOVE DFHVALUE(RELEASED) TO AREA-A.  
EXEC CICS SET TERMINAL(terminal_id)  
ACQSTATUS(AREA-A)
```

This form allows you to change a CVDA value in your program as the result of other runtime factors.

The ILE C equivalent is:

```
symb_name = DFHVALUE(RELEASED);  
EXEC CICS SET TERMINAL(terminal_id)  
ACQSTATUS(symb_name)
```

You cannot use DFHVALUE as a keyword argument. CICS/400 does not support the following:

```
EXEC CICS SET TERMINAL(terminal_id)  
ACQSTATUS(DFHVALUE(RELEASED))  
END-EXEC.
```

Short form

If the required CVDA value is always the same for a particular command, you can specify this value directly as an option on the command. For example:

```
EXEC CICS ENQ RESOURCE(RESNAME) LUW  
  
or  
  
EXEC CICS ENQ RESOURCE(RESNAME) TASK  
  
or  
  
EXEC CICS SET TERMINAL(terminal_id)  
RELEASED
```

You *receive* a CVDA value by specifying a fullword binary data-area into which it can be received. You can then use the DFHVALUE translator built-in function to test the returned value, as in the following COBOL example:

```
EXEC CICS CONNECT PROCESS ...
      STATE(AREA-A)
END-EXEC.
IF AREA-A = DFHVALUE(ALLOCATED) THEN ...
IF AREA-A = DFHVALUE(CONFFREE) THEN ...
```

The ILE C equivalent is:

```
EXEC CICS CONNECT PROCESS ...
      STATE(symb_name);
if (symb_name == DFHVALUE(ALLOCATED)) ...
if (symb_name == DFHVALUE(CONFFREE)) ...
;
```

Any of the following CVDA values can be returned by CICS/400 as the state of the current conversation on APPC commands such as `CONNECT PROCESS`, although not all values are applicable to each command:

- ALLOCATED
- CONFFREE
- CONFRECEIVE
- CONFSEND
- FREE
- PENDFREE
- PENDRECEIVE
- RECEIVE
- SEND

For a list of the CVDA symbolic names that are supported by CICS/400, together with their associated numeric values, see Appendix E, “CICS-value data areas supported by CICS/400,” on page 579.

Note: You should be aware that, when using the command-level interpreter (CECI) or the execution diagnostic facility (EDF), CVDAs are displayed on the screen as numeric values rather than their associated symbolic names.

DATASET option

On file control commands, the `DATASET` option is accepted for compatibility with CICS application programs that have been ported to CICS/400 from earlier releases on other CICS platforms, but `FILE` is the preferred option and should be used for all new CICS application programs.

Similarly, `DSIDERR` is supported as a synonym for the `FILENOTFOUND` condition. This applies to its use both as an option of the `HANDLE CONDITION` command and also as an argument to the `DFHRESP` built-in function.

INTO and SET options

The `INTO` and `SET` options provide alternative ways to specify the address of data to be received by an application program.

If an application expects to receive data by issuing a RECEIVE command, you must specify either the INTO or the SET option.

If an application issues a RECEIVE command simply to test whether a terminal user has pressed an attention identifier key (AID), you can omit both the INTO and the SET option. If an application uses a RECEIVE command to accept an AID, ensure that a HANDLE AID command is issued before the RECEIVE command. See the HANDLE AID command for more information about the attention identifier keys.

LENGTH options

Many commands involve the transfer of data between the application program and CICS. In most cases, the length of the data to be transferred must be provided by the application program; the syntax of each command and its associated options shows whether this rule applies.

For COBOL programs, if a data-area is specified as the source or target on a command and the translator is able to generate a default length, there is no need for your program to provide the length explicitly unless you want CICS to use a length different from that of the variable referred to.

When a CICS command offers the LENGTH option, it is generally expressed as a signed halfword binary value. This puts a theoretical upper limit of 32 767 bytes on LENGTH, or 32 763 bytes allowing for the possibility of four bytes of control information. In practice (depending on issues of recoverability, function shipping, and so on) the achievable upper limit varies from command to command, but is somewhat less than this theoretical maximum.

Whatever the CICS command, you are recommended not to use a LENGTH value greater than 24KB. This recommendation is for consistency with other CICS platforms, and hence for application portability.

For *journaling* commands, the record length may be further restricted by the buffer size of the journal. The journal record length is the sum of the LENGTH and PFXLENG values.

For *temporary storage*, *transient data*, and *file control* commands, the file definitions may themselves impose further restrictions.

NOHANDLE option

You can use the NOHANDLE option on any command to specify that you want no action to be taken for any condition or attention identifier (AID) resulting from the execution of the command. The NOHANDLE option is available on all EXEC CICS commands and is not shown on the syntax diagrams in this book.

Note that using the C language implies NOHANDLE on all commands.

The NOHANDLE option covers all conditions that can occur for the commands on which it is specified; the IGNORE CONDITION command covers specified conditions for all commands (until its effect is changed by a HANDLE CONDITION command that names one or more of these conditions).

You must be careful when using NOHANDLE with the RECEIVE command, because NOHANDLE overrides the HANDLE AID command as well as the HANDLE CONDITION command, with the result that PF key responses are ignored.

RESP and RESP2 options

You can use the RESP option with any command to test whether an exception condition was raised during its execution. With some commands where a condition can occur for more than one reason, you can use the RESP2 option to determine more exactly why the condition occurred.

The RESP and RESP2 options are implicit on all EXEC CICS commands and are not shown on the syntax diagrams in this book.

After every EXEC CICS command CICS sets the RESP field in the Execution Interface Block (EIB) to reflect the condition raised. This includes the special condition NORMAL.

Some commands provide more detailed information about why an exception condition was raised. These commands are typically ones where the same exception condition can occur for more than one reason. For these commands CICS sets the RESP2 field in the EIB to further qualify the exception condition value returned in the RESP field.

Where RESP2 values are defined to further qualify the RESP value, they are included in the "Exception conditions" section of the command descriptions that follow. In all other cases the RESP2 field is reserved and the returned value is undefined.

For more general information on exception conditions see Chapter 6, "Dealing with exception conditions," on page 87.

RESP(xxx)

"xxx" is a user-defined fullword binary data-area. On return from the command, it contains a value corresponding to a condition that may have been raised or to a normal response, that is, xxx=DFHRESP(NORMAL). You can use the DFHRESP translator built-in function to test the returned value.

In COBOL you can test the returned RESP value as follows:

```
EXEC CICS WRITEQ TS FROM(abc)
                        QUEUE(qname)
                        RESP(xxx)
END-EXEC.

IF xxx = DFHRESP(NORMAL) THEN ...
IF xxx = DFHRESP(NOSPACE) THEN ...
```

In ILE C, a similar test would be:

```

EXEC CICS WRITEQ TS FROM(abc)
                QUEUE(qname)
                RESP(xxx);

switch (xxx) {
  case DFHRESP(NORMAL) : break;
  case DFHRESP(NOSPACE) : Nospace_Cond();
                        break;
  default              : Errors();
}

```

In ILE C programs you must have issued an EXEC CICS ADDRESS EIB command to obtain addressability to the EIB prior to using the RESP option.

As the use of RESP implies NOHANDLE, you must be careful when using RESP with the RECEIVE command in COBOL programs. See “NOHANDLE option” on page 312 for further information.

RESP2(yyy)

“yyy” is a user-defined fullword binary data-area. On return from the command, it contains a value that further qualifies the response to certain commands. Unlike the RESP values, RESP2 values have no associated symbolic names and there is no translator built-in function corresponding to DFHRESP, so you must test the fullword binary value itself.

System programming commands

The system programming commands provide a command-level equivalent to most of the functions of the master terminal (CEMT). (See the *CICS for iSeries Administration and Operations Guide*.) CICS command-level applications can be written and invoked as CICS transactions to administer the running of the CICS system. This could, for example, provide a subset of the master terminal function, for example, for a particular person or group of people.

The system programming commands have all the advantages of the other EXEC CICS commands. In particular, they are supported by the CICS command interpreter (CECI), the CICS execution diagnostic facility (CEDF), and the CICS translator.

Note: There is a difference between the system programming commands and basic application programming commands in that none of the system programming commands can be function shipped. If a CICS resource is defined as remote, its remote definition cannot be retrieved or updated. An exception to this rule is when a CICS TCT is available in a remote system in either model or surrogate form. A change can then be made to the remote definition. The change is not shipped back to the terminal owning region (TOR). This allows the CICS user to make a change that applies only to the remote TCT.

The system programming commands are:

- INQUIRE and SET commands
- the PERFORM command
- the DISCARD command

INQUIRE and SET commands

The EXEC CICS INQUIRE and EXEC CICS SET commands allow user-written system programs to look at the information that defines a named CICS resource (including the installed definition and runtime values) and to change some of the values.

On CICS, these commands give access to the definitions of the following CICS resources:

- Autoinstall terminal models
- CICS/400 control region parameters
- Connections
- Files
- Journals
- Programs
- Tasks
- Terminals
- Trace facilities
- Transactions
- Transient data queues

Changes to the resource attributes apply only for the duration of the current CICS session and are supported only within the local CICS system.

Although the INQUIRE and SET commands have functions that can be used in application programs, they are intended primarily for user-written system programs and not for general application programs.

The INQUIRE command also has special forms that enable you to browse all of the runtime definitions for a particular resource. For more information, see “Browsing resource definitions.”

The following general points should be noted:

- You do not have to issue an INQUIRE command before issuing a SET command.
- In almost all cases, the items that you can change with a SET command are a subset of those that you can retrieve with an INQUIRE command. If a CICS resource is defined as remote, normally you cannot retrieve or alter its remote definition. This rule applies to the definitions of CICS files, transient data queues, and transactions.
- CICS does not maintain exclusive control of the information that is returned on an INQUIRE command. This means that it can be changed at any time, for example, by issuing a SET command.
- If a condition occurs on an INQUIRE command, the command did not execute correctly and the validity of the returned values cannot be guaranteed.
- If a condition occurs on a SET command, as few as possible of the requested changes have been made on return. To establish which, if any, of the changes have been made, you can issue an INQUIRE command.

Browsing resource definitions

The INQUIRE command allows you to retrieve information about a single named resource. The command also has special forms that allow you to browse all of the runtime definitions for a particular resource.

You can browse the definitions for the following resources:

- Autoinstall terminal models
- Connections
- Files
- Journals
- Programs
- Terminals
- Transactions
- Transient data queues

There is no locking of the retrieved information, which means that a definition can be changed at any time during the browse.

Note that there is no facility for selective browsing; that is, for returning entries only if they meet certain conditions.

The general command format of a browse operation has the following standard pattern, where *resource* represents the resource type (for example, PROGRAM):

▶▶—INQUIRE—*resource*—START—▶▶

Condition: ILLOGIC

▶▶—INQUIRE—*resource*—(—*data-area*—)—NEXT—▶▶

▶—Other options as for INQUIRE—*resource*—▶▶

Conditions: END, ILLOGIC

▶▶—INQUIRE—*resource*—END—▶▶

Condition: ILLOGIC

Descriptions follow of each of the command forms. The information given applies to all browse operations.

INQUIRE *resource* START

This command positions an internal pointer at the first definition in the relevant CICS resource definitions. It does not retrieve any information, nor does it allow you to specify a start point.

Only one browse of a particular resource type is allowed at any one time in a given task. The ILLOGIC condition occurs if a browse is already in progress when an INQUIRE START command for the same resource type is issued.

INQUIRE *resource*(*data-area*) NEXT

The first time this command is issued, it retrieves the name and any requested attributes of the first resource definition. On each successive occasion, it retrieves the name and attributes of the next resource definition, if there is one.

The END condition occurs when you have already retrieved all the entries in the relevant CICS resource definitions. All data-areas specified on the command are left unchanged. Note that the default action for the END condition is to terminate the task abnormally, so you need to detect its occurrence.

You cannot request a selective browse of the resource definitions, and you can only browse forward. All optional attributes that you specify are output fields only and are ignored on input.

CICS maintains a pointer to the current browse position at transaction level, and does not reset it when LINK or XCTL commands are issued.

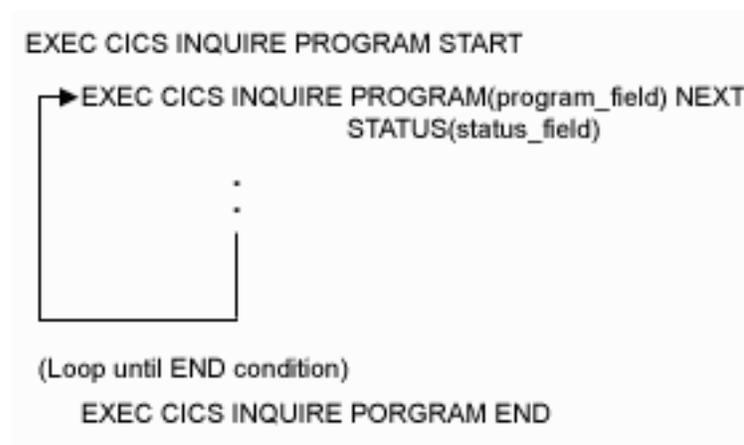
The ILLOGIC condition occurs if an INQUIRE NEXT command is issued for a resource type that has had no previous INQUIRE START command successfully issued.

INQUIRE *resource* END

This command ends the browse and frees any held resources. You can end a browse at any time after issuing the INQUIRE START command. If you do not issue an INQUIRE END command, the browse terminates at the end of the transaction. A browse operation is not terminated by user syncpoints.

The ILLOGIC condition occurs if an INQUIRE END command is issued for a resource type that has had no previous INQUIRE START command successfully issued.

The following example shows a browse of the processing program table (PPT) that retrieves the name and status of each program defined in the PPT. The entire table is being browsed, so the application issues the INQUIRE PROGRAM NEXT command until the END condition arises, and then issues the INQUIRE PROGRAM END command.



Null values

If you issue an INQUIRE command to find out the value of an attribute that is not applicable to the named resource, a “null” value is returned in the data-area that you have defined. You also get a null value if the information you have requested is not available at the time of the inquiry.

Null values depend on the format of the user-defined data-area, and are defined as follows:

- Character fields are blanks.

- Binary fields are -1.
- Pointer fields are X'FF000000'.
- CVDA fields are DFHVALUE(NOTAPPLIC). For further information about the use of CVDA, see "CICS-value data areas (CVDA)" on page 309.

If you issue a SET command that includes one or more null argument values, the corresponding attributes are ignored. This allows the possibility of coding general SET commands in which some attributes may be left as they are, without having to issue an INQUIRE command first to establish what the current values are. You are most likely to want to do this when the field is defined as a CVDA. However, in all cases, if you simply omit an optional attribute from a SET command, its value remains unchanged.

For CVDA on SET commands, the null value can be coded as DFHVALUE(IGNORE), where IGNORE is equivalent to (and has the same numeric value as) NOTAPPLIC on INQUIRE commands.

The SET command is coded as in the following COBOL example:

```
MOVE DFHVALUE(IGNORE) TO AREA-A.

EXEC CICS SET TDQUEUE(queue_name)
        ENABLED OPENSTATUS(AREA-A)
END-EXEC.
```

This causes the named transient data queue to be enabled, and its open status remains the same as it was before the SET TDQUEUE command was issued. Simply omitting the OPENSTATUS option from the command would achieve the same effect, but this approach lacks flexibility for some coding situations.

For further information about the use of CVDA, see "CICS-value data areas (CVDA)" on page 309.

PERFORM command

The EXEC CICS PERFORM SHUTDOWN command is used for shutting down the CICS system.

DISCARD commands

The following CICS resource types can be discarded:

- Autoinstall terminal models
- Files
- Programs
- Transactions

The EXEC CICS DISCARD command removes installed CICS resource definitions from an active CICS/400 control region. The EXEC CICS DISCARD command does not affect the OS/400 user space used to define the CICS resource, therefore the resource can be reinstated using the CEDA Install group function. See *CICS for iSeries Administration and Operations Guide* for further information about using CEDA to install resource definitions.

EXEC CICS DISCARD commands have the same security attached to them as EXEC CICS SET commands. The EXEC CICS DISCARD command takes effect

immediately following the issue of the command. You cannot discard CICS resources that are locked to a user or are resources owned by CICS. These have a name beginning either with "AEG" or with "C" for CICS-supplied transactions.

Commands by function

The following is a list of commands categorized according to the function they perform. Commands that are not supported for ILE C programs are shown by the superscript "1" after the command name in this list. Where a command has one or more options that are not supported for ILE C programs, this is indicated in the description of the command.

Abend support

ABEND
HANDLE ABEND

APPC mapped conversation

ALLOCATE
CONNECT PROCESS
CONVERSE
EXTRACT ATTRIBUTES
EXTRACT PROCESS
FREE
ISSUE ABEND
ISSUE CONFIRMATION
ISSUE ERROR
ISSUE PREPARE
ISSUE SIGNAL
RECEIVE
SEND
WAIT CONVID

BMS

RECEIVE MAP
SEND CONTROL
SEND MAP
SEND TEXT

Built-in function

BIF DEEDIT

Diagnostic services

DUMP TRANSACTION
ENTER TRACENUM

Environment services

ADDRESS
ASSIGN

DISCARD AUTINSTMODEL
DISCARD FILE
DISCARD PROGRAM
DISCARD TRANSACTION
INQUIRE AUTINSTMODEL
INQUIRE CONNECTION
INQUIRE FILE
INQUIRE JOURNALNUM
INQUIRE PROGRAM
INQUIRE SYSTEM
INQUIRE TASK
INQUIRE TDQUEUE
INQUIRE TERMINAL/NETNAME
INQUIRE TRACEDEST
INQUIRE TRANSACTION
PERFORM SHUTDOWN
SET CONNECTION
SET FILE
SET JOURNALNUM
SET PROGRAM
SET SYSTEM
SET TASK
SET TDQUEUE
SET TERMINAL
SET TRACEDEST
SET TRANSACTION

Exception support

HANDLE CONDITION¹
IGNORE CONDITION¹
POP HANDLE¹
PUSH HANDLE¹

File control

DELETE
ENDBR
READ
READNEXT
READPREV
RESETBR
REWRITE
STARTBR
UNLOCK
WRITE

Interval control

ASKTIME
CANCEL
DELAY
FORMATTIME
POST
RETRIEVE
START
WAIT EVENT

Journaling

WAIT JOURNALNUM
WRITE JOURNALNUM

Printer spooling

SPOOLCLOSE
SPOOLOPEN OUTPUT
SPOOLWRITE

Program control

LINK
LOAD
RELEASE
RETURN
XCTL

Storage control

FREEMAIN
GETMAIN

Syncpoint

SYNCPOINT
SYNCPOINT ROLLBACK

Task control

DEQ
ENQ
SUSPEND

Temporary storage control

DELETEQ TS
READQ TS
WRITEQ TS

Terminal control

CONVERSE (3270 logical)
HANDLE AID¹
ISSUE ERASEAUP
RECEIVE (3270 logical)
SEND (SCS)
SEND (3270 logical)

Transient data control

DELETEQ TD
READQ TD
WRITEQ TD

Chapter 32. Application programming commands - reference

This chapter shows the syntax for each EXEC CICS command, in alphabetic order, and describes the options and exception conditions applicable to each command. For a general introduction to EXEC CICS commands and an explanation of the syntax notation used, refer to Chapter 31, "Programming reference," on page 305.

ABEND

Terminate a task abnormally.

ABEND

►► ABEND [ABCODE(—name—)] [CANCEL] [NODUMP] ◀◀

Description

ABEND terminates a task abnormally.

The main storage associated with the terminated task is released; optionally, a transaction dump of this storage can be obtained.

Refer to Chapter 9, "Abnormal termination recovery," on page 109 for more information about this command.

Options

ABCODE(*name*)

specifies that main storage related to the task that is being terminated is to be dumped. The ABCODE is used as a transaction dumpcode to identify the dump. The name should have up to four characters and should not contain any leading or imbedded blanks. If ABCODE is not coded, the dump is identified by ????.

Do not start the name with the letter A, because this is reserved for CICS itself.

CANCEL

specifies that exits established by HANDLE ABEND commands are to be ignored. An ABEND CANCEL command cancels all exits at any level in the task (and terminates the task abnormally).

NODUMP

allows you to request an abend without causing a dump to be taken.

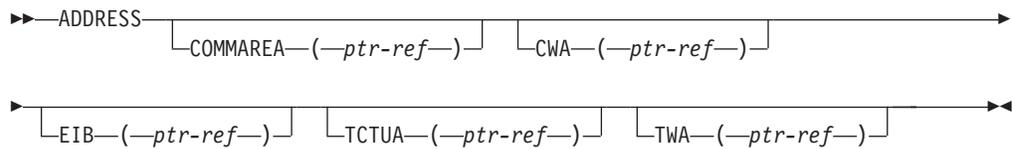
Examples

The following example shows how to terminate a task abnormally:

```
EXEC CICS ABEND ABCODE('BCDE')
```

ADDRESS

Obtain access to CICS storage areas.



Condition: INVREQ

Description

ADDRESS accesses the following areas:

- the communication area available to the invoked program (COMMAREA)
- the common work area (CWA)
- the EXEC interface block (EIB)
- the terminal control table user area (TCTUA)
- the transaction work area (TWA).

See pages 68 through 71 for more information about this command.

Options

COMMAREA(*ptr-ref*)

returns a pointer reference, set to the address of the communication area (COMMAREA) available to the currently executing program. COMMAREA is used to pass information between application programs. If the COMMAREA does not exist, the pointer reference is set to the null value, X'FF000000'.

CWA(*ptr-ref*)

returns a pointer reference, set to the address of the common work area (CWA). This area makes information available to applications running in a single CICS system. If a CWA does not exist, CICS sets the pointer reference to the null value, X'FF000000'.

EIB(*ptr-ref*)

returns a pointer reference set to the address of the EXEC interface block (EIB). You must use this option to get addressability to the EIB when it is not already available.

TCTUA(*ptr-ref*)

returns a pointer reference, set to the address of the terminal control table user area (TCTUA) for the principal facility, not that for any alternate facility that may have been allocated. This area is used for passing information between application programs, but only if the same terminal is associated with the application programs involved. If a TCTUA does not exist, the pointer reference is set to the null value, X'FF000000'.

TWA(*ptr-ref*)

returns a pointer reference, set to the address of the transaction work area (TWA). This area is used for passing information between application programs, but only if they are in the same task. If a TWA does not exist, the pointer reference is set to the null value, X'FF000000'.

If TASKDATALOC(ANY) is defined on the transaction definition, the address of the data may be above or below the 16MB line.

If TASKDATALOC(BELOW) is defined on the transaction definition, and the data resides above the 16MB line, the data is copied below the 16MB line, and the address of this copy is returned.

Exception Conditions

INVREQ

RESP2 values:

200 The TCTUA option is specified on an ADDRESS command issued in a DPL server program.

Default action: terminate the task abnormally.

ALLOCATE

Acquire a session to a remote APPC logical unit for use by an APPC mapped conversation.

```
▶▶—ALLOCATE—SYSID—(—name—)——┬──PROFILE—(—name—)──┬──NOQUEUE──┬──┬──NOSUSPEND──┬──┬──▶▶
                                └──┬──STATE—(—cvda—)──┘
```

Conditions: INVREQ, SYSBUSY, SYSIDERR

Description

ALLOCATE acquires the session and optionally selects a set of session-processing options. CICS makes one of the sessions associated with the named system available to the application program.

CICS returns, in the EIBRSRCE field of the EIB, the 4-byte conversion identifier (CONVID) for use by the application program in all subsequent commands that relate to the conversation.

See *CICS for iSeries Intercommunication* for more help on using this command.

Options

NOQUEUE

requests CICS to pass control to the CONNECT command that follows this ALLOCATE command, if a session is not immediately available from the requested SYSID. This option is overridden by a previously issued HANDLE CONDITION(SYSBUSY) command. If NOQUEUE is not specified and a HANDLE CONDITION(SYSBUSY) command has not been issued, CICS suspends application execution until a session is available.

NOSUSPEND

is an alternative keyword for NOQUEUE. It means the same.

PROFILE(name)

specifies the name of an OS/400 mode description used during the execution

of mapped commands for the session specified in the SYSID option. The name can be up to 8 characters long. If this option is omitted, a default profile, specified during resource definition, is selected.

STATE(*cvda*)

gets the state of the current conversation. The following CVDA value is returned by CICS if the request is successful:

ALLOCATED

For a complete list of the CVDA values that can be returned on APPC commands and for information about receiving and testing these values, see “CICS-value data areas (CVDAs)” on page 309.

SYSID(*name*)

specifies the name of an APPC connection to the required system. The name can be up to 4 characters long. This option requests that one of the sessions to the named system is to be allocated.

Exception Conditions

INVREQ

occurs if the ALLOCATE command is not valid for the device to which it is directed.

Default action: Terminate the task abnormally.

SYSBUSY

occurs if no session is immediately available. A previously issued HANDLE CONDITION(SYSBUSY) command overrides the NOQUEUE option.

Default action: CICS suspends application execution until a session is available.

SYSIDERR

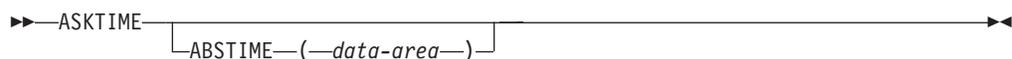
occurs if CICS is unable to provide the application program with a suitable session. That is, if:

- The SYSID option specifies a name that is not defined in the terminal control system table (TCS).
- The mode name derived from the PROFILE option is not one of the mode names defined for the APPC system entry.
- All the sessions in the group specified by the SYSID option and the mode name are out of service.
- All sessions to the system specified by the SYSID option are out of service.

Default action: Terminate the task abnormally.

ASKTIME

Request the current date and time of day.



Description

ASKTIME updates the date (EIBDATE) and CICS time-of-day clock (EIBTIME) fields in the EIB. These two fields initially contain the date and time when the task started. Refer to Appendix A, “EXEC interface block,” on page 529 for details of the EIB.

Options

ABSTIME(*data-area*)

specifies a user data area to receive the time, in milliseconds since 00:00 on 1 January 1900, as a packed decimal value. For example, after execution of:

```
EXEC CICS ASKTIME ABSTIME(utime)
```

“utime” might contain a value like 002837962864828 in milliseconds.

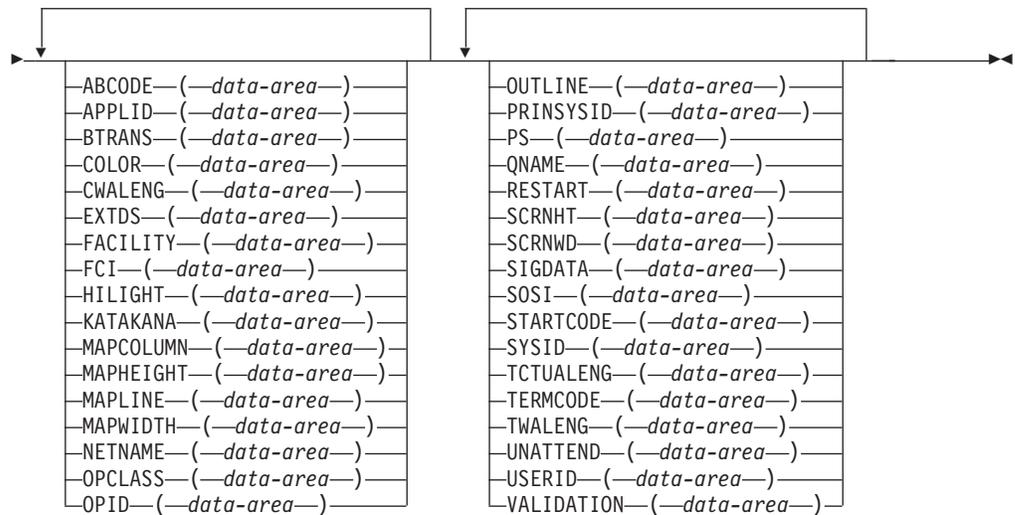
The format of the ABSTIME argument is:

```
COBOL: PIC S9(15) COMP-3  
ILE C: char abs_time[8];
```

ASSIGN

Request values from outside the application program’s local environment.

►►—ASSIGN—►►



Condition: INVREQ

Description

ASSIGN gets values from outside the local environment of the application program. The data obtained depends on the specified options. Up to sixteen options can be specified in one ASSIGN command.

Where any of the following options apply to terminals or terminal-related data, the reference is always to the principal facility.

If the principal facility is a remote terminal, the data returned is obtained from the local copy of the information; the request is not routed to the system to which the remote terminal is attached.

Transaction routing is as far as possible transparent to the ASSIGN command. In general, the values returned are the same whether the transaction is local or remote.

Many of the ASSIGN options cannot be used in a server program invoked by a distributed program link (DPL), and result in the INVREQ condition if specified. For further details of the restricted API in the DPL server environment, see the LINK command on page 380.

The following ASSIGN options are restricted in DPL server programs: BTRANS, COLOR, EXTDS, FACILITY, FCI, HIGHLIGHT, KATAKANA MAPCOLUMN, MAPHEIGHT, MAPLINE, MAPWIDTH, OPCLASS, OUTLINE, PS, QNAME, SCRINH, SCRINH, SIGDATA, SOSI TCTUALENG, TERMCODE, UNATTEND, and VALIDATION.

Options

ABCODE(*data-area*)

returns a 4-character current abend code. (Abend codes are documented in *CICS/400 Problem Determination*.) If an abend has not occurred, the variable is set to blanks.

APPLID(*data-area*)

returns an 8-character applid of the CICS system owning the transaction.

BTRANS(*data-area*)

returns a 1-byte indicator showing whether the terminal is defined as having the background transparency capability (X'FF') or not (X'00'). If the task is not initiated from a terminal, INVREQ occurs.

COLOR(*data-area*)

returns a 1-byte indicator showing whether the terminal is defined as having the extended color capability (X'FF') or not (X'00'). If the task is not initiated from a terminal, INVREQ occurs.

CWALENG(*data-area*)

returns a halfword binary field indicating the length of the CWA. If no CWA exists, a zero length is returned.

EXTDS(*data-area*)

returns a 1-byte indicator showing whether the terminal accepts the 3270 extended data stream, (X'FF') or not (X'00'). Extended data stream capability is required for a terminal that supports the query feature, color, extended highlighting, programmed symbols or validation.

If the task is not initiated from a terminal, INVREQ occurs.

FACILITY(*data-area*)

returns a 4-byte identifier of the facility that initiated the transaction; If this option is specified, and there is no allocated facility, INVREQ occurs.

Note: You should always use the QNAME option (described on page 330) to get the name of the transient data intrapartition queue whose trigger level caused the transaction to be initiated.

FCI(*data-area*)

returns a 1-byte field giving the facility control indicator code. This indicates the type of facility associated with the transaction; for example, X'01' indicates a terminal or logical unit. The obtained value is always returned. This code indicates the type of facility associated with the transaction.

The codes are listed here as both bit patterns and hexadecimal values.

Code		Meaning
.... ...1	X'01'	TERMINAL OR LOGICAL UNIT
.... ..1.	X'02'	K C P MACRO FILE MASK
.... .1..	X'04'	NONTERMINAL FACILITY MASK
.... 1...	X'08'	Triggered task
...1	X'10'	START with data
111.	X'E0'	Reserved

HIGHLIGHT(*data-area*)

returns a 1-byte indicator showing whether the terminal is defined as having the extended highlight capability (X'FF') or not (X'00'). If the task is not initiated from a terminal, INVREQ occurs.

KATAKANA(*data-area*)

returns a 1-byte indicator showing whether the principal facility supports Katakana (X'FF') or not (X'00'). If the task is not initiated from a terminal, INVREQ occurs.

MAPCOLUMN(*data-area*)

returns a halfword binary number of the column on the display containing the origin of the most recently positioned map. If no map has yet been positioned, or if the task is not initiated from a terminal, INVREQ occurs.

MAPHEIGHT(*data-area*)

returns a halfword binary height of the most recently positioned map. If no map has yet been positioned, or if the task is not initiated from a terminal, INVREQ occurs.

MAPLINE(*data-area*)

returns a halfword binary number of the line on the display containing the origin of the most recently positioned map. If no map has yet been positioned, or if the task is not initiated from a terminal, INVREQ occurs.

MAPWIDTH(*data-area*)

returns a halfword binary width of the most recently positioned map. If no map has yet been positioned, or if the task is not initiated from a terminal, INVREQ occurs.

NETNAME(*data-area*)

returns the 8-character name of the logical unit in the network. If the task is not initiated from a terminal, INVREQ occurs. If the principal facility is not a local terminal, CICS no longer returns a null string but the netname of the remote terminal.

OPCLASS(*data-area*)

returns a 3-character string indicating the operator class. Operator class codes are in the range 1 through 24. One bit represents each of the 24 possible operator classes, but in reverse order (that is, byte 0 corresponds to codes 24 through 17, byte 1 to codes 16 through 9, and byte 2 to codes 8 through 1). If the task is not initiated from a terminal, INVREQ occurs. CICS/400 accepts this option for compatibility only.

OPID(*data-area*)

returns a 3-character string indicating the operator identification. If the task is not initiated from a terminal, INVREQ occurs. CICS/400 accepts this option for compatibility only.

OUTLINE(*data-area*)

returns a 1-byte indicator showing whether the terminal is defined as having the field outlining capability (X'FF') or not (X'00'). If the task is not initiated from a terminal, INVREQ occurs.

PRINSYSID(*data-area*)

returns a 4-character string that applies when the principal facility is an APPC session to another CICS system, or to another APPC system or device.

PRINSYSID is the name by which the other system is known in the local system; that is, the CONNECTION definition that defines the other system.

If the task is not initiated from a terminal, or if the principal facility is not an APPC session, INVREQ occurs.

Note: Special considerations apply generally when transaction routing. In particular, an EXEC CICS ASSIGN PRINSYSID command cannot be used in a routed transaction to find the name of the terminal-owning region. See the *CICS/400 Intercommunication* book for more information about transaction routing.

PS(*data-area*)

returns a 1-byte indicator showing whether the terminal is defined as having the programmed symbols capability (X'FF') or not (X'00'). If the task is not initiated from a terminal, INVREQ occurs.

QNAME(*data-area*)

returns a 4-character name of the transient data intrapartition queue that caused this task to be initiated by reaching its trigger level. If the task is not initiated by automatic transaction initiation (ATI), INVREQ occurs.

RESTART(*data-area*)

returns a 1-byte indicator showing whether a restart of the task (X'FF'), or a normal start of the task (X'00'), has occurred.

SCRNHT(*data-area*)

returns a halfword binary variable that contains the height of the 3270 or 5250 screen defined for the current task. If the task is not initiated from a terminal, INVREQ occurs.

SCRNWD(*data-area*)

returns a halfword binary variable that contains the width of the 3270 or 5250 screen defined for the current task. If the task is not initiated from a terminal, INVREQ occurs.

SIGDATA(*data-area*)

returns a 4-byte character string containing the inbound signal data received from a logical unit. If the task is not initiated from a terminal, INVREQ occurs.

SOSI(*data-area*)

returns a 1-byte indicator showing whether the terminal is defined as having the mixed SBCS/DBCS fields capability (X'FF') or not (X'00'). The DBCS subfields within an SBCS field are delimited by SO (shift-out) and SI (shift-in) characters. If the task is not initiated from a terminal, INVREQ occurs.

STARTCODE(*data-area*)

returns a 2-byte indicator showing how the transaction issuing the request was started. It can have the following values:

Code **Transaction started by**

D A distributed program link (DPL) request that did not specify the

SYNCONRETURN option. The task cannot issue I/O requests against its principal facility, nor can it issue any syncpoint requests. (For details, see the LINK command on page 380.)

- DS** A distributed program link (DPL) request, as in code D, that did specify the SYNCONRETURN option. The task can issue syncpoint requests.
- QD** Transient data trigger level.
- S** START command without data.
- SD** START command with data.
- TD** Terminal input or permanent transid.
- TP** CICS transaction.
- U** User-attached task.

SYSID(*data-area*)

returns the 4-character name given to the local CICS control region. This value may be specified in the SYSID option of a file control, interval control, temporary storage, or transient data command, in which case the resource to be accessed is assumed to be on the local system.

TCTUALENG(*data-area*)

returns a halfword binary length of the terminal control table user area (TCTUA). If no TCTUA exists, a zero length is returned.

TERMCODE(*data-area*)

returns a 2-character string giving the type and model number of the terminal associated with the task.

The first byte is a code identifying the terminal type; the second byte is a single-character model number. The terminal type is derived from the DEVTYPE parameter of the ADDCICSTCT CL command. The model number is set by the DEVD parameter of the same command. See the *CICS/400 Administration and Operations Guide* for details of ADDCICSTCT.

The terminal type codes are listed here as both bit patterns and hexadecimal values.

Note: The list of possible values is shorter than that for CICS for MVS/ESA because there are fewer device types supported by CICS/400.

Code		Meaning
1..1 ...1	X'91'	T3277R: 5250, 3270, 3270J, ASCII
1..1 ..11	X'93'	T3284L: 3270P, 3270JP
1.11 .11.	X'B6'	T3790SCSP: SCS

If the task is not initiated from a terminal, INVREQ occurs.

TWALENG(*data-area*)

returns a halfword binary length of the transaction work area (TWA). If no TWA exists, a zero length is returned.

UNATTEND(*data-area*)

returns a 1-byte indicator showing whether the mode of operation of the terminal is unattended, that is to say no person is actually attending the

terminal. These indicators are X'FF' for unattended and X'00' for attended. If the task is not initiated from a terminal, INVREQ occurs.

USERID(*data-area*)

returns an 10-character string indicating the user identifier of whoever is signed on. This option returns a blank string when there is no user identifier.

If the task is not initiated from a terminal, INVREQ occurs.

VALIDATION(*data-area*)

returns a 1-byte indicator showing whether the terminal is defined as having the validation capability (X'FF') or not (X'00'). Validation capability consists of the mandatory fill, mandatory enter, and trigger attributes. If the task is not initiated from a terminal, INVREQ occurs.

Exception Conditions

INVREQ

RESP2 values:

- 2 No BMS command has yet been issued, or no map has yet been positioned.
- 4 The task is not initiated by automatic transaction initiation (ATI).
- 5 The task is not associated with a terminal; or the task has no principal facility; or the principal facility is not an APPC session.
- 200 An ASSIGN command in a DPL server program includes one or more terminal-related options that are not allowed in a server environment.

Default action: terminate the task abnormally.

BIF DEEDIT

Deediting (built-in function).



Description

BIF DEEDIT provides the built-in function DEEDIT. It specifies that alphabetic and special characters are removed from an EBCDIC data field, and the remaining digits right-aligned and padded to the left with zeros as necessary.

If the field ends with a minus sign or a carriage-return (CR), a negative zone (X'D') is placed in the rightmost (low-order) byte.

If the zone portion of the rightmost byte contains one of the characters X'A' through X'F', and the numeric portion contains one of the hexadecimal digits X'0' through X'9', the rightmost byte is returned unaltered (see the example). This permits the application program to operate on a zoned numeric field. The returned value is in the field that initially contained the unedited data.

Note that a 1-byte field is returned unaltered, no matter what the field contains.

Options

FIELD(*data-area*)
specifies the field to be edited.

LENGTH(*data-value*)
specifies the field length in bytes.

Examples

```
EXEC CICS BIF DEEDIT  
      FIELD(CONTG)  
      LENGTH(9)
```

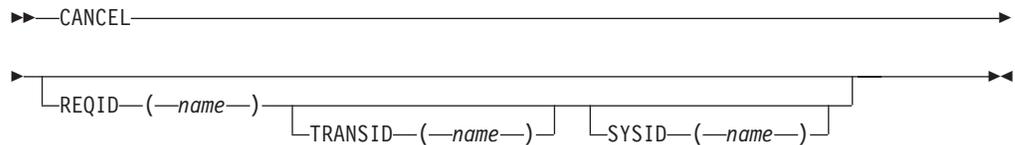
This removes all characters other than digits from CONTG, a 9-byte field, and returns the edited result in that field to the application program. Two examples of the contents of CONTG before and after execution of the command are:

Original value	Returned value
14-6704/B	00146704B
\$25.68	000002568

Note that a decimal point is an EBCDIC special character and as such is removed.

CANCEL

Cancel an interval control request.



Conditions: INVREQ, ISCINVREQ, NOTAUTH, NOTFND, SYSIDERR

Description

CANCEL cancels a previously issued DELAY, POST, or START command. If you include the SYSID option, the command is shipped to a remote system. If you omit SYSID, the TRANSID option, if present, determines where the command is to be executed. The effect of the cancellation varies depending on the type of command being canceled, as follows:

- A DELAY command can be canceled only before it has expired, and only by a task other than the task that issued the DELAY command (which is suspended for the duration of the request). The REQID used by the suspended task must be specified. The effect of the cancellation is the same as an early expiration of the original DELAY. That is, the suspended task becomes dispatchable as though the original expiration time has been reached.
- When a POST command issued by the same task is to be canceled, no REQID need be specified. Cancellation can be requested either before or after expiration of the original request. The effect of the cancellation is as if the original request had never been made.
- When a POST command issued by another task is to be canceled, the REQID of that command must be specified. The effect of the cancellation is the same as an

early expiration of the original POST request. That is, the timer event control area for the other task is posted as though the original expiration time had been reached.

- When a START command is to be canceled, the REQID of the original command must be specified. The effect of the cancellation is as if the original command had never been made. The cancellation is effective only before the original command has expired.

See Chapter 18, “Interval control,” on page 193 for more information.

Options

REQID(*name*)

specifies a name that uniquely identifies the command to be canceled. The name can be up to 8 characters long. This name is used as a temporary storage identifier. The temporary storage queue thus identified must be defined as a local queue on the CICS system where the CANCEL command is processed.

This option cannot be used to cancel a POST command issued by the same task (for which the REQID option is ignored if it is specified).

SYSID(*name*)

specifies the name of the system to which the request is directed. The name can be up to 4 characters long.

TRANSID(*name*)

specifies the symbolic identifier that determines where the CANCEL command is to be executed. The name can be up to 4 characters long and must have been defined in the program control table (PCT) unless the SYSID option specifies a remote system.

If a nonlocal SYSID is specified, the transaction is assumed to be on a remote system irrespective of whether the name is defined in the local PCT. Otherwise, the PCT entry is used to determine whether the transaction is on a local or a remote system.

Exception Conditions

INVREQ

occurs if the CANCEL command is not valid for processing by CICS.

Default action: Terminate the task abnormally.

ISCINVREQ

occurs if the remote system indicates a failure that does not correspond to a known condition.

Default action: Terminate the task abnormally.

NOTAUTH

occurs if the user requesting the CANCEL command is not authorized to the CICS/400 temporary storage files for the control region in which the command is being run.

Default action: Terminate the task abnormally.

NOTFND

occurs if the request identifier specified fails to match an unexpired interval control command.

Default action: Terminate the task abnormally.

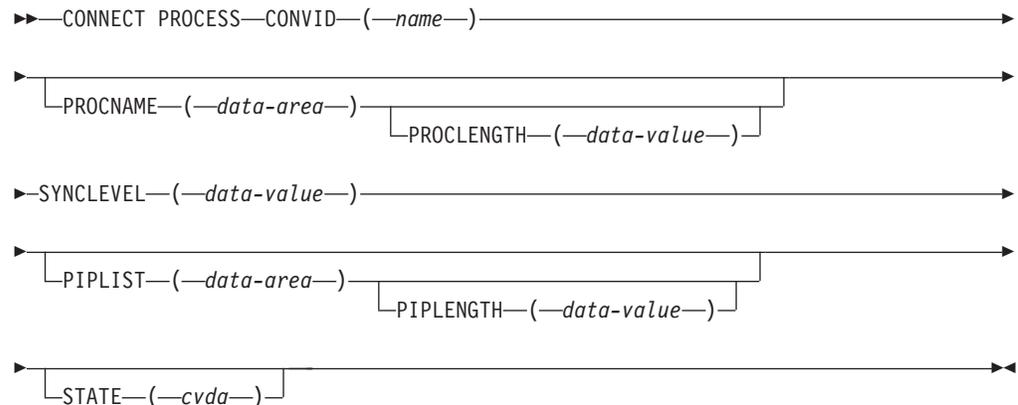
SYSIDERR

occurs if the SYSID option specifies either a name that is not defined in the terminal control system table (TCS), or a system to which the link is closed.

Default action: Terminate the task abnormally.

CONNECT PROCESS

Initiate APPC mapped conversation.



Conditions: INVREQ, LENGERR, NOTALLOC

Description

CONNECT PROCESS allows an application to specify a process name and synchronization level to be passed to CICS and used when the remote partner is attached.

See part 3 of *CICS/400 Intercommunication* for more information.

Options

CONVID(*name*)

identifies the conversation to which the command relates. The 4-character name specifies the token returned by a previously executed ALLOCATE command in EIBRSRCE in the EIB.

PIPLENGTH(*data-value*)

specifies the total length (halfword binary value) of the specified process initialization parameter (PIP) list.

PIPLIST(*data-area*)

specifies the PIP data to be sent to the remote system. The PIP list consists of variable-length records, each containing a single PIP. A PIP starts with a 2-byte inclusive length field (LL), followed by a 2-byte reserved field, and then the parameter data.

PROCLENGTH(*data-value*)

specifies the length (as a halfword binary value) of the process name specified by the PROCNAME option.

PROCNAME(*data-area*)

specifies the partner process (that is, the transaction) to be attached in the remote system.

One byte is sufficient to identify a CICS transaction. The APPC architecture allows a range of 1–64 bytes but leaves each product free to set its own maximum. CICS complies by allowing a range of 1–32 bytes. If the remote system expects ASCII characters, there are certain restrictions; for full details, refer to the “AE” character set defined in the SNA publication *SNA LU6.2 reference: Verb descriptions*, GC30-3084.

STATE(*cvda*)

gets the state of the current conversation.

For a complete list of the CVDA values that can be returned on APPC commands and for information about receiving and testing these values, see “CICS-value data areas (CVDAs)” on page 309.

SYNCLEVEL(*data-value*)

specifies the synchronization level (halfword binary value) for the current conversation. The possible values are:

- 0 None
- 1 Confirm
- 2 Syncpoint

Exception Conditions

INVREQ

RESP2 values:

200 A distributed program link server application specified the function-shipping session (its principal facility) on the CONVID option.

also occurs (RESP2 not set) in any of the following situations:

- A synchronization level other than 0, 1, or 2, has been requested in the SYNCLEVEL option.
- The command is not valid for the terminal or LU in use.
- The CONVID value was obtained by an ASSIGN FACILITY command. However, the principal facility is *not* an APPC conversation.

Default action: terminate the task abnormally.

LENGERR

occurs in any of the following situations:

- An out-of-range value is supplied in the PROCLENGTH option.
- The value specified in the PIPLength option is less than 0.
- The value specified in the PIPLength option exceeds the CICS implementation limit of 32 763.
- A PIPLIST length element (LL) has a value less than 4.
- The sum of the length elements (LLs) in the PIPLIST does not equal the value specified by PIPLength.

Default action: terminate the task abnormally.

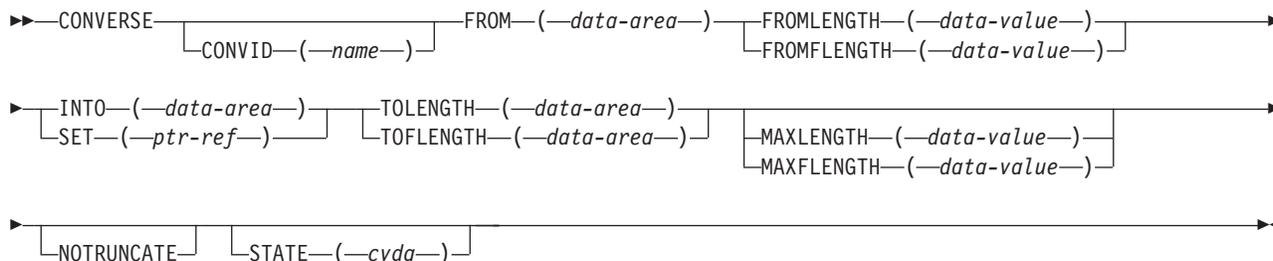
NOTALLOC

occurs if the specified CONVID value does not relate to a conversation owned by the application.

Default action: terminate the task abnormally.

CONVERSE (APPC)

Send, then receive, data on an APPC mapped conversation.



Conditions: EOC, INVREQ, LENGERR, NOTALLOC, SIGNAL, TERMERR

Description

CONVERSE sends, then receives, data on an APPC mapped conversation. See *CICS for iSeries Intercommunication* for more information.

Options

CONVID(*name*)

identifies the conversation to which the command relates. The 4-character name identifies the token returned by a previously executed ALLOCATE command in the EIBRSRCE field of the EIB. If this option is omitted, the principal facility for the task is used by default.

FROM(*data-area*)

specifies the data to be sent to the partner transaction.

FROMFLENGTH(*data-value*)

is a fullword alternative to FROMLENGTH(*data-value*).

FROMLENGTH(*data-value*)

specifies as a halfword binary value the length of the data to be sent.

INTO(*data-area*)

specifies the application target data area into which data is to be received from the application program connected to the other end of the current conversation. The length of this area must be greater than or equal to the maximum receive length specified in the TOLENGTH, TOFLENGTH, MAXLENGTH, and MAXFLENGTH options.

MAXFLENGTH(*data-value*)

is a fullword alternative to MAXLENGTH(*data-value*).

MAXLENGTH(*data-value*)

specifies as a halfword binary value the maximum amount of data that CICS is to recover in response to a CONVERSE command. If INTO is specified, MAXLENGTH overrides the use of TOLENGTH as an input to CICS. If SET is specified, MAXLENGTH provides a way for the program to limit the amount of data that it receives at one time.

If the length of data exceeds the value specified and the NOTRUNCATE option is not present, the data is truncated to that value and the LENGERR condition occurs. The data area specified in the TOLENGTH option is set to the original length of data (before any truncation).

If the length of data exceeds the value specified and the NOTRUNCATE option is present, CICS retains the remaining data and uses it to satisfy subsequent RECEIVE commands. To determine whether all the data has been received, a program can use the EIBCOMPL field in the EIB. The data area specified in the TOLENGTH option is set to the length of data returned.

If MAXLENGTH is omitted, CICS uses the value specified in the TOLENGTH option as the maximum length that the program accepts.

NOTRUNCATE

specifies that, when the data available exceeds the length requested, the remaining data is not to be discarded but is to be retained for retrieval by subsequent RECEIVE commands.

SET(*ptr-ref*)

specifies a pointer reference to be set to the address of the data received from the partner transaction. The pointer reference, unless changed by other commands or statements, is valid until the next CONVERSE (APPC) command or the end of task.

STATE(*cvda*)

gets the state of the current conversation. For a complete list of the CVDA values that can be returned on APPC commands and for information about receiving and testing these values, see “CICS-value data areas (CVDAs)” on page 309.

TOLENGTH(*data-area*)

is a fullword alternative to TOLENGTH(*data-area*).

TOLENGTH(*data-area*)

specifies as a halfword binary value the length of the data to be received. If you specify INTO, but omit MAXLENGTH, this option specifies the maximum length that the program accepts.

Exception Conditions

EOC

occurs when no other condition is raised. The EIBEOC field also contains this indicator.

Default action: Ignore the condition.

INVREQ

RESP2 values:

- The CONVID value was obtained by an ASSIGN FACILITY command. However, the principal facility is *not* an APPC conversation.
- 200** The command is issued in a DPL server program and refers to the principal facility.

Default action: Terminate the task abnormally.

LENGERR

occurs in any of the following situations:

- Received data is discarded by CICS because its length exceeds the maximum that the program accepts (see the TOLENGTH and MAXLENGTH options), and the NOTRUNCATE option is not specified.
- An out-of-range value is supplied in one of the options, TOLENGTH, FROMLENGTH, MAXLENGTH, TOFLENGTH, FROMFLENGTH, or MAXFLENGTH.

Default action: Terminate the task abnormally.

NOTALLOC

occurs if the specified CONVID value does not relate to a conversation owned by the application.

Default action: Terminate the task abnormally.

SIGNAL

occurs if an inbound SIGNAL data-flow control command is received from the partner transaction. EIBSIG is always set when an inbound signal is received.

Default action: Ignore the condition.

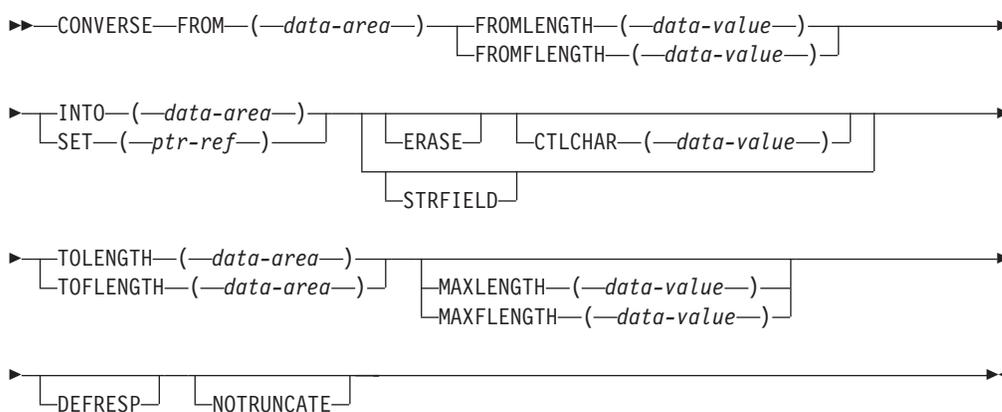
TERMERR

occurs if there is a session-related error. Any action on that conversation other than a FREE command causes an ATCV abend.

Default action: Terminate the task abnormally (with abend code ATNI).

CONVERSE (5250 or 3270 logical)

Communicate on a 5250-display or 3270-display logical unit, or a 3270-printer logical unit.



Conditions: EOC, INVREQ, LENGERR, TERMERR

Description

The CONVERSE (5250 or 3270 logical) command communicates on a 5250-display or 3270-display logical unit, or a 3270-printer logical unit. See Chapter 14, "Terminal control," on page 169 for more information.

Options

CTLCHAR(data-value)

specifies a 1-byte write control character (WCC) that controls the CONVERSE command. A COBOL user must specify a data area containing this character. If the option is omitted, all modified data tags are reset to zero and the keyboard is restored.

DEFRESP (ignored by CICS/400)

indicates that a definite response is required when the output operation has been completed.

ERASE

specifies that the buffer is to be erased or the display image is to be erased and the cursor returned to the upper left corner of the screen before writing occurs.

Normally, ERASE should be specified in the first output command of a transaction. This clears the screen ready for the new output data.

However, when switching from one screen size to another on a transaction basis, if ERASE is not specified in the first output command of the transaction, the screen size is unchanged from its previous setting (that is, the previous transaction setting, or the default screen size if the CLEAR key has been pressed).

FROM(*data-area*)

specifies the data to be written to the logical unit.

FROMLENGTH(*data-value*)

is a fullword alternative to FROMLENGTH(*data-value*).

FROMLENGTH(*data-value*)

specifies as a halfword binary value the length of the data to be written.

For a description of a safe upper limit, see "LENGTH options" on page 312.

INTO(*data-area*)

specifies the receiving field for the data read from the logical unit.

MAXLENGTH(*data-value*)

is a fullword alternative to MAXLENGTH(*data-value*).

MAXLENGTH(*data-value*)

specifies as a halfword binary value the maximum amount of data that CICS is to recover in response to a CONVERSE command. If INTO is specified, MAXLENGTH overrides the use of TOLENGTH as an input to CICS. If SET is specified, MAXLENGTH provides a way for the program to limit the amount of data that it receives at one time.

If the length of data exceeds the value specified and the NOTRUNCATE option is not present, the data is truncated to that value and the LENGERR condition occurs. The data area specified in the TOLENGTH option is set to the original length of data (before any truncation).

If the length of data exceeds the value specified and the NOTRUNCATE option is present, CICS retains the remaining data and uses it to satisfy subsequent RECEIVE commands. The data area specified in the LENGTH option is set to the length of data returned.

If MAXLENGTH is omitted, CICS uses the value specified in the TOLENGTH option as the maximum length that the program accepts.

NOTRUNCATE

specifies that when the data available exceeds the length requested, the remaining data is not to be discarded but is to be retained for retrieval by subsequent RECEIVE commands.

SET(*ptr-ref*)

specifies a pointer reference to be set to the address of the data read from the terminal or logical unit. The pointer reference, unless changed by other commands or statements, is valid until the next terminal I/O command or the end of task.

STRFIELD

specifies that the data area specified in the FROM option contains structured

fields. This option applies to 3270 devices only. When this option is specified, the contents of all structured fields must be handled by the application program. (Structured fields are described in the *3270 Data Stream Programmer's Reference*.)

CTLCHAR and ERASE are mutually exclusive with STRFIELD, and their use with STRFIELD generates an error message.

TOFLENGTH(*data-area*)

is a fullword alternative to TOLENGTH(*data-area*).

TOLENGTH(*data-area*)

specifies as a halfword binary value the length of the data to be received. If you specify INTO, but omit MAXLENGTH, this specifies the maximum length that the program accepts. If the value is less than zero, zero is assumed. If the length of data exceeds the value specified, but NOTRUNCATE is omitted, the data is truncated to that value, and the LENGERR condition occurs. When the data is received, the data area is set to the length of the data. If you specify the SET option, the argument must be a data area. When the data has been received, the data area is set to the length of the data.

For a description of a safe upper limit, see "LENGTH options" on page 312.

Exception Conditions

EOC

occurs when no other condition is raised. The EIBEOC field also contains this indicator.

Default action: Ignore the condition.

INVREQ

RESP2 values:

200 The command is issued in a DPL server program.

Default action: Terminate the task abnormally.

LENGERR

occurs in any of the following situations:

- Received data is discarded by CICS because its length exceeds the maximum that the program accepts (see the TOLENGTH, TOFLENGTH, MAXLENGTH, and MAXFLENGTH options), and the NOTRUNCATE option is not specified.
- An out-of-range value is supplied in the FROMLENGTH, FROMFLENGTH, MAXLENGTH, MAXFLENGTH, TOLENGTH, or TOFLENGTH option.

Default action: Terminate the task abnormally.

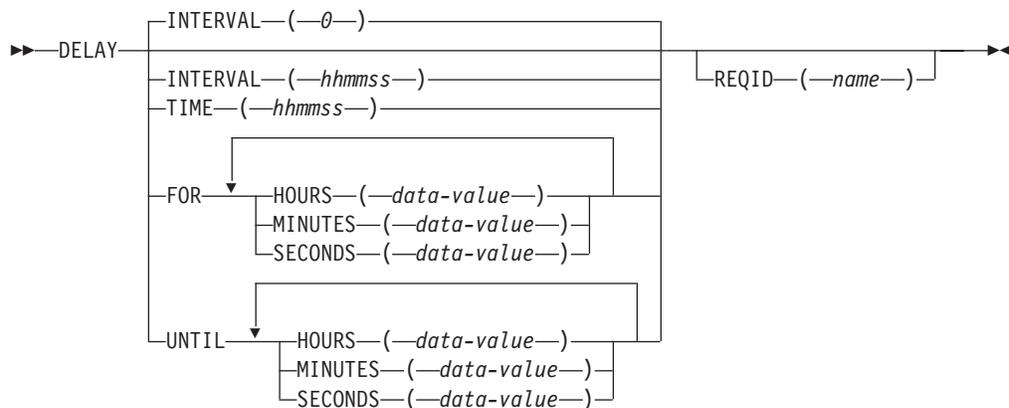
TERMERR

occurs if there is a terminal-related error.

Default action: Terminate the task abnormally (with abend code ATNI).

DELAY

Delay the processing of a task.



Conditions: EXPIRED, INVREQ

Description

DELAY suspends the processing of the issuing task for a specified interval of time or until a specified time of day. It supersedes any previously initiated POST command for the task.

See Chapter 18, "Interval control," on page 193 for more information.

Options

FOR

together with one or more of the HOURS, MINUTES, and SECONDS options, specifies the interval of time for which the task processing is to be suspended.

HOURS(*data-value*)

specifies as a fullword binary value the number of hours for use in conjunction with FOR or UNTIL. The value must be in the range 0 through 99.

INTERVAL(*hhmmss*)

specifies the interval of time that is to elapse from the time at which the DELAY command is issued until processing is resumed. The specified interval is added to the current clock time by CICS to calculate the expiration time. See Chapter 18, "Interval control," on page 193 for an explanation of how expiration times are used with interval control.

INTERVAL(0) is the default. The maximum permitted INTERVAL value is 995959.

For compatibility between CICS platforms, it is recommended that the INTERVAL option is not used in ILE C programs. You should use the FOR option instead.

MINUTES(*data-value*)

specifies as a fullword binary value the number of minutes for use in conjunction with FOR or UNTIL. The value must be in the range 0 through 59 if HOURS or SECONDS is also specified, or in the range 0 through 5999 otherwise.

REQID(*name*)

specifies a name that uniquely identifies the DELAY command. The name can be up to 8 characters long. This name is used as a temporary storage identifier.

This option can be used when another task is to be provided with the capability of canceling an unexpired DELAY command.

SECONDS(*data-value*)

specifies as a fullword binary value the number of seconds for use in conjunction with FOR or UNTIL. The value must be in the range 0 through 59 if HOURS or MINUTES is also specified, or in the range 0 through 359999 otherwise.

TIME(*hhmmss*)

specifies the time at which the task is to resume processing. See Chapter 18, "Interval control," on page 193 for an explanation of how expiration times are used within interval control.

For compatibility between CICS platforms, it is recommended that the INTERVAL option is not used in ILE C programs. You should use the UNTIL option instead.

UNTIL

together with one or more of the HOURS, MINUTES, and SECONDS options, specifies the time at which the task is to resume processing.

Exception Conditions

EXPIRED

occurs if the time specified has already expired when the command is issued.

Default action: Ignore the condition.

INVREQ

RESP2 values:

- The DELAY command is not valid for processing by CICS.
- 4 Hours are out of range.
- 5 Minutes are out of range.
- 6 Seconds are out of range.
- The DELAY command specifies a REQID name that already exists.

Default action: Terminate the task abnormally.

Examples

The following example shows how to suspend the processing of a task for five minutes, using the INTERVAL option:

```
EXEC CICS DELAY
      INTERVAL(500)
      REQID('GXLBZQMR')
      :
```

The following example shows how to suspend the processing of a task until 12:45, using the TIME option:

```
EXEC CICS DELAY
      TIME(124500)
      REQID('UNICODE')
      :
```

ILE C supports the packed decimal argument type and, consequently, the use of the INTERVAL and TIME options. It is recommended, however, that you use the FOR and UNTIL options for portability of applications between CICS platforms. There are two ways to enter the interval or time value using FOR or UNTIL:

1. A combination of at least two of HOURS(0-99), MINUTES(0-59), and SECONDS(0-59). For example, HOURS(1) SECONDS(3) means one hour and three seconds (the minutes default to zero).
2. Any one of HOURS(0-99), MINUTES(0-5999), or SECONDS(0-359999). For example, HOURS(1) means one hour; MINUTES(62) means one hour and two minutes; and SECONDS(3723) means one hour, two minutes, and three seconds.

The following example shows how to suspend the processing of a task for five minutes, using the FOR option:

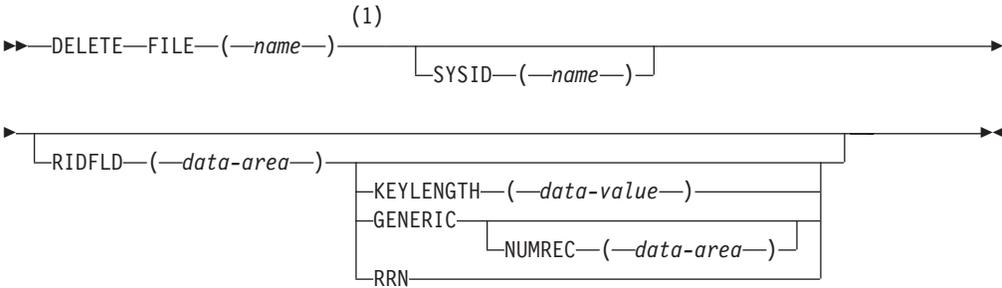
```
EXEC CICS DELAY
      FOR MINUTES(5)
      REQID('GXLBZQMR')
      :
```

The following example shows how to suspend the processing of a task until 12:45, using the UNTIL option:

```
EXEC CICS DELAY
      UNTIL HOURS(12) MINUTES(45)
      REQID('UNICODE')
      :
```

DELETE

Delete a record.



Notes:

- 1 DATASET is also accepted, but FILE is the preferred term (see “DATASET option” on page 311).

Conditions: DISABLED, DUPKEY, FILENOTFOUND, ILLOGIC, INVREQ, IOERR, ISCINVREQ, NOTAUTH, NOTFND, NOTOPEN, SYSIDERR

Description

DELETE deletes a record from a file that refers to a KSDS, a path over a KSDS, or an RRDS. (Records cannot be deleted from an ESDS.) The file may be on a local or a remote system. You identify, in the RIDFLD option, the specific record to be deleted.

You can delete a group of records in a similar way with a single invocation of this command, except that you identify the group by the GENERIC option.

You can also use a different form of this command to delete a single record that has previously been retrieved for update (by a READ UPDATE command). The record is deleted, instead of being rewritten, by this command. In this case, you must not specify the RIDFLD option.

Refer to Chapter 10, "File control," on page 115 for more information.

Options

FILE(*name*)

specifies the name of the file to be accessed. The name must be alphanumeric, up to 8 characters long, and must have been defined in the file control table (FCT) unless the SYSID option specifies a remote system.

If a nonlocal SYSID is specified, the underlying file is assumed to be on a remote system irrespective of whether the name is defined in the local FCT. Otherwise, the FCT entry is used to determine whether the underlying file is on a local or a remote system.

GENERIC

specifies that the search key is a generic key with a length specified in the KEYLENGTH option. The GENERIC option may only be used with a KSDS or a path over a KSDS. The search for a record is satisfied when a record is found with a key that has the same starting characters (generic key) as those specified.

KEYLENGTH(*data-value*)

specifies as a halfword binary value the length of the key supplied in the RIDFLD option. If a specified KEYLENGTH value differs from the length defined for the underlying file and the operation is not generic, the INVREQ condition occurs.

INVREQ also occurs if you specify GENERIC, and the KEYLENGTH value is not less than that defined for the file.

Note: If you specify GENERIC and KEYLENGTH(0), CICS/400 deletes all the records from the file.

NUMREC(*data-area*)

specifies a halfword binary data area that CICS sets to the number of records that have been deleted.

RIDFLD(*data-area*)

specifies the record identification field. The contents can be a key or a relative record number. For a relative record number, the format of this field must be fullword binary and the RIDFLD can be greater than or equal to 1.

RRN

specifies that the record identification field specified in the RIDFLD option contains a relative record number. Use this option only when deleting records from an RRDS.

SYSID(*name*)

specifies the name of the system to which the request is directed. The name can be up to 4 characters long.

If you specify SYSID, and omit RRN, you must also specify KEYLENGTH; it cannot be found in the FCT.

Exception Conditions

Note: RESP2 values are not set for files that are on remote systems.

DISABLED

RESP2 values:

50 A file is disabled.

A file may be disabled because:

- It was initially defined as disabled and has not since been enabled.
- It has been disabled by an EXEC CICS SET FILE command.
- It has been disabled by the CEMT transaction.

This condition cannot occur when deleting a record just read for update.

Default action: Terminate the task abnormally.

DUPKEY

RESP2 values:

140 A record is deleted from a VSAM emulated file that allows duplicate keys, and another record with the same key exists.

Default action: Terminate the task abnormally.

FILENOTFOUND

RESP2 values:

1 The name specified in the FILE option cannot be found in the FCT.

Default action: Terminate the task abnormally.

ILLOGIC

RESP2 values:

110 There is an error that does not fall within one of the other CICS response categories. (Further information is available in the EIBRCODE field; refer to Appendix A, "EXEC interface block," on page 529 for details.)

Default action: Terminate the task abnormally.

INVREQ

RESP2 values:

20 Delete operations are not allowed according to the file entry specification in the FCT.

21 A DELETE command is issued for a file referring to an ESDS.

- 22 A generic delete is issued for a file that is not a KSDS.
- 25 The KEYLENGTH and GENERIC options are specified, and the length specified in the KEYLENGTH option is greater than or equal to the length of a full key.
- 26 The KEYLENGTH option is specified (but the GENERIC option is not specified), and the specified length differs from the length defined for the file.
- 31 A DELETE command without the RIDFLD option is issued for a file for which no previous READ UPDATE command has been issued.
- 32 A DELETE command with the RIDFLD option specified is issued for a file when a READ UPDATE command is outstanding.
- 42 The KEYLENGTH and GENERIC options are specified, and the length specified in the KEYLENGTH option is less than zero.

Default action: Terminate the task abnormally.

IOERR

RESP2 values:

- 120 There is an I/O error during the file control operation. An I/O error is any unusual event that is not covered by a CICS exception condition.

(Further information is available in the EIBRCODE field; refer to Appendix A, "EXEC interface block," on page 529 for details.)

Default action: Terminate the task abnormally.

ISCINVREQ

RESP2 values:

- 70 The remote system indicates a failure that does not correspond to a known condition.

Default action: Terminate the task abnormally.

NOTAUTH

RESP2 values:

- 101 A resource security check has failed on FILE(*name*).

Default action: Terminate the task abnormally.

NOTFND

RESP2 values:

- 80 An attempt to delete a record based on the search argument provided is unsuccessful.

Default action: Terminate the task abnormally.

NOTOPEN

RESP2 values:

- 60 One of the following has occurred:
 - The requested file is CLOSED and UNENABLED. The CLOSED, UNENABLED state is reached after a close request has been received against an OPEN ENABLED file and the file is no longer in use.

- The requested file is OPEN and UNENABLED and in use by other transactions, but a close request against the file has been received.

This condition does not occur if the request is made to either a CLOSED, ENABLED file or a CLOSED, DISABLED file. In the first case, the file is opened as part of executing the request. In the second case, the DISABLED condition occurs. It also cannot occur when deleting a record just read for update.

Default action: Terminate the task abnormally.

SYSIDERR

RESP2 values:

- 130** The SYSID option specifies either a name that is not defined in the terminal control system table (TCS), or a system to which the link is closed.

Default action: Terminate the task abnormally.

Examples

The following example shows how to delete a group of records in a file:

```
EXEC CICS DELETE
      FILE('MASTVSAM')
      RIDFLD(ACCTNO)
      KEYLENGTH(LEN)
      GENERIC
      NUMREC(NUMDEL)
      :
```

DELETEQ TD

Delete all the transient data associated with a particular intrapartition destination (queue).

```
▶▶DELETEQ TD=QUEUE—(—name—)——[SYSID—(—name—)]————▶▶
```

Conditions: DISABLED, INVREQ, ISCVREQ, NOTAUTH, QIDERR, SYSIDERR

Description

Note: You cannot use this command to delete an *extrapartition* transient data queue. An attempt to do so results in the INVREQ condition.

See Chapter 23, “Transient data control,” on page 215 for more information.

Options

QUEUE(*name*)

specifies the symbolic name of the queue to be deleted. The name must be alphanumeric, up to 4 characters long, and must have been defined in the destination control table (DCT) unless the SYSID option specifies a remote system.

If a nonlocal SYSID is specified, the queue is assumed to be on a remote system irrespective of whether the name is defined in the local DCT. Otherwise, the DCT entry is used to determine whether the queue is on a local or a remote system.

SYSID(*name*)

specifies the name of the system to which the request is directed. The name can be up to 4 characters long.

Exception Conditions

DISABLED

occurs if the queue has been disabled.

Default action: Terminate the task abnormally.

INVREQ

occurs if DELETEQ names an extrapartition queue.

Default action: Terminate the task abnormally.

ISCINVREQ

occurs if the remote system indicates a failure that does not correspond to a known condition.

Default action: Terminate the task abnormally.

NOTAUTH

occurs if a resource security check has failed on QUEUE(*name*).

Default action: Terminate the task abnormally.

QIDERR

occurs if the symbolic destination to be used with DELETEQ TD cannot be found.

Default action: Terminate the task abnormally.

SYSIDERR

occurs if the SYSID option specifies either a name that is not defined in the terminal control system table (TCS), or a system to which the link is closed.

Default action: Terminate the task abnormally.

DELETEQ TS

Delete a temporary storage queue.

►► DELETEQ TS—QUEUE—(*name*)—SYSID—(*name*)—►►

Conditions: INVREQ, ISCINVREQ, NOTAUTH, QIDERR, SYSIDERR

Description

DELETEQ TS deletes all the temporary data associated with a temporary storage queue. All storage associated with the queue is freed.

You should delete temporary data as soon as possible to avoid using excessive amounts of storage.



Conditions: INVREQ, LENGERR

Description

DEQ causes a resource currently enqueued on by the task to be released for use by other tasks.

If a task enqueues on, but does not dequeue from, a resource, CICS automatically releases the resource during syncpoint processing or when the task is terminated. A resource in the context of this command is any string of 1–255 bytes, established by in-house standards, to protect against conflicting actions between tasks, or to cause single-threading within a program.

When issuing the DEQ command, the resource to be dequeued from must be identified by the method used when enqueueing on the resource. (See “ENQ” on page 355.) If no enqueue has been issued for the resource, the dequeue is ignored.

If more than one ENQ command is issued for a resource by a task, that resource remains owned by the task until the task issues a matching number of DEQ commands.

See Chapter 19, “Task control,” on page 197 for more information.

Options

LENGTH(*data-value*)

specifies as a halfword binary value the length of the resource to be dequeued from. The value must be in the range 1 through 255; otherwise, the LENGERR condition occurs. If the LENGTH option is specified in an ENQ command, it must also be specified in the corresponding DEQ command for that resource, and the values of these options must be the same.

MAXLIFETIME(*cvda*)

specifies the duration of the ENQ being released. CVDA values are:

LUW ENQ was acquired with a duration of a logical unit of work. This is the default value.

TASK ENQ was acquired with a duration of a task.

For examples of how to code the MAXLIFETIME option, see “CICS-value data areas (CVDAs)” on page 309.

RESOURCE(*data-area*)

specifies either an area whose address represents the resource to be dequeued from, or a variable that contains the resource (an employee name, for example). In the latter case, you must use the LENGTH option.

Exception Conditions

INVREQ

occurs in any of the following situations:

- The MAXLIFETIME option is set with an incorrect CVDA.

Default action: terminate the task abnormally.

LENGERR

RESP2 values:

- 1 The value you specified for the LENGTH option is outside the range 1 through 255.

Default action: terminate the task abnormally.

Examples

The following examples show how to dequeue from a resource:

```
EXEC CICS DEQ
      RESOURCE(RESNAME)

EXEC CICS DEQ
      RESOURCE(SOCSECNO)
      LENGTH(8)
```

DUMP TRANSACTION

Request a transaction dump.

DUMP TRANSACTION

►►—DUMP TRANSACTION—DUMPCODE—(—*name*—)—————►

Condition: INVREQ

Description

DUMP or DUMP TRANSACTION dumps all of the main storage areas related to a task.

Note: The TRANSACTION keyword is not necessary on this command. For compatibility purposes both EXEC CICS DUMP and EXEC CICS DUMP TRANSACTION are accepted.

See “Dump” on page 112 for more information.

Options

DUMPCODE(*name*)

specifies a name (1–4 characters) that identifies the dump. The name can be up to 4 characters long, must not contain any leading or embedded blanks, and must not start with “A”, which is reserved for CICS.

Exception Conditions

INVREQ

RESP2 values:

- 13 An incorrect DUMPCODE is specified. DUMPCODE contains unprintable characters, or leading or imbedded blanks.

Default action: terminate the task abnormally.

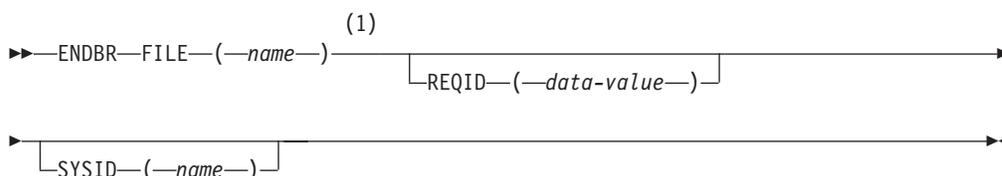
Examples

The following example shows how to request a dump of all the task-related storage areas:

```
EXEC CICS DUMP TRANSACTION
      DUMPCODE('name')
      :
```

ENDBR

End a browse on a file on a local or a remote system.



Notes:

- 1 DATASET is also accepted, but FILE is the preferred term (see “DATASET option” on page 311).

Conditions: FILENOTFOUND, ILLOGIC, INVREQ, ISCINVREQ, NOTAUTH, SYSIDERR

Description

You must always issue an ENDBR command before performing any update operations on the same file (READ UPDATE, DELETE with RIDFLD, or WRITE), and before a syncpoint. You need to issue ENDBR only after a successful STARTBR command.

Refer to Chapter 10, “File control,” on page 115 for more information.

Options

FILE(*name*)

specifies the name of the file being browsed. The name must be alphanumeric, up to 8 characters long, and must have been defined in the file control table (FCT) unless the SYSID option specifies a remote system.

If a nonlocal SYSID is specified, the underlying file is assumed to be on a remote system irrespective of whether the name is defined in the local FCT. Otherwise, the FCT entry is used to determine whether the underlying file is on a local or a remote system.

REQID(*data-value*)

specifies as a halfword binary value a unique request identifier for the browse; it is used to control multiple browse operations on a file. If this option is not specified, a default value of zero is assumed.

SYSID(*name*)

specifies the name of the system to which the request is directed. The name can be up to 4 characters long.

Exception Conditions

Note: RESP2 values are not set for files that are on remote systems.

FILENOTFOUND

RESP2 values:

- 1** The name referred to in the FILE option cannot be found in the FCT.

Default action: Terminate the task abnormally.

ILLOGIC

RESP2 values:

- 110** There is an error that does not fall within one of the other CICS response categories. (Further information is available in the EIBRCODE field; refer to Appendix A, "EXEC interface block," on page 529 for details.)

Default action: Terminate the task abnormally.

INVREQ

RESP2 values:

- 35** The ENDBR command is issued for a file that has not had a successful STARTBR command previously issued for it.

Default action: Terminate the task abnormally.

ISCINVREQ

RESP2 values:

- 70** The remote system indicates a failure that does not correspond to a known condition.

Default action: Terminate the task abnormally.

NOTAUTH

RESP2 values:

- 101** A resource security check has failed on FILE(*name*).

Default action: Terminate the task abnormally.

SYSIDERR

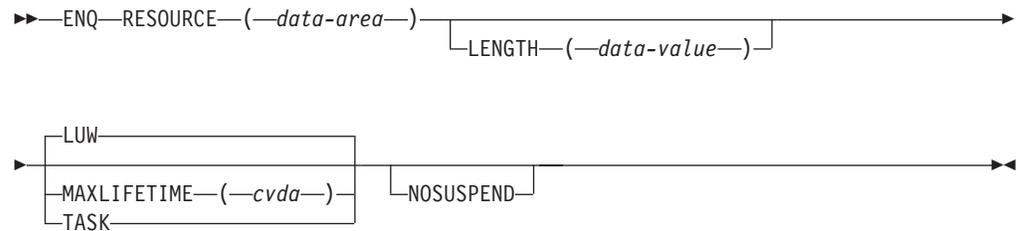
RESP2 values:

- 130** The SYSID option specifies either a name that is not defined in the terminal control system table (TCS), or a system to which the link is closed.

Default action: Terminate the task abnormally.

ENQ

Schedule use of a resource by a task (enqueue).



Conditions: ENQBUSY, LENGERR, INVREQ

Description

ENQ causes further execution of the task issuing the ENQ command to be synchronized with the availability of the specified resource; control is returned to the task when the resource is available.

A resource in the context of this command is any string of 1–255 bytes, established by in-house standards, to protect against conflicting actions between tasks, or to cause single threading within a program.

If a task enqueues on a resource but does not dequeue from it, CICS automatically releases the resource during syncpoint processing or when the task is terminated. Option LUW on MAXLIFETIME forces the dequeue at the end of a logical unit of work (LUW). Option TASK on MAXLIFETIME forces the dequeue at the end of a task. If there are several LUWs in a task, the enqueue carries over the LUWs.

If more than one ENQ command is issued for the same resource by a given task, the resource remains owned by that task until the task issues a matching number of DEQ commands.

The resource to be enqueued on must be identified by one of the following methods:

- Specifying a data area that is the resource. It is the location (address) of the data area in storage that matters, not its contents.
- Specifying a data area that contains a unique argument (for example, an employee name) that represents the resource. It is the contents of the data value that matters, not its location.

If a resource is not available when ENQUEUED, the application program is suspended until it becomes available. However, if the NOSUSPEND option has been specified and the resource is unavailable, the ENQBUSY condition is raised, as it is also raised if you have an active HANDLE condition. This allows the application program to handle the case of resource unavailability (by HANDLE CONDITION ENQBUSY) without waiting for the resource to become available.

See Chapter 19, “Task control,” on page 197 for more information.

Options

LENGTH(*data-value*)

specifies as a halfword binary value the length of the resource to be enqueued on. The value must be in the range 1 through 255; otherwise, the LENGERR

condition occurs. If the LENGTH option is specified in an ENQ command, it must also be specified in the DEQ command for that resource, and the values of these options must be the same. You must specify LENGTH when using the method that specifies a data value containing a unique argument, but not for the method that specifies a data area as the resource. It is the presence or absence of LENGTH that tells CICS which method you are using.

MAXLIFETIME(*cvda*)

specifies the duration of the ENQ to be automatically released by CICS. CVDA values are:

- LUW** The duration of the ENQ is a logical unit of work. Examples are a syncpoint rollback or syncpoint, if the application does not issue a DEQ before the LUW ends. This is the default value.
- TASK** The duration of the ENQ is a task. The enqueue carries over the LUWs within the task. Use MAXLIFETIME(TASK) with great care because other tasks issuing ENQ commands on the same resource could be suspended until the end of this task.

There are two ways to code this option.

- You can assign a *cvda* value with the translator routine DFHVALUE. This allows you to change a *cvda* value in the program. For example:

```
MOVE DFHVALUE(LUW) TO AREA-A
EXEC CICS ENQ RESOURCE(RESNAME)
        MAXLIFETIME(AREA-A)
```

- If the required action is always the same, you can declare the value directly. For example:

```
EXEC CICS ENQ RESOURCE(RESNAME) LUW
or
EXEC CICS ENQ RESOURCE(RESNAME) TASK
```

NOSUSPEND

specifies that the application program is not to be suspended if the ENQBUSY condition occurs.

RESOURCE(*data-area*)

identifies the resource to be enqueued on by:

- Specifying an area whose address represents the resource.
- Specifying a variable that contains the resource (an employee name, for example). In this case, you must use the LENGTH option.

Exception Conditions

ENQBUSY

occurs when an ENQ command specifies a resource that is unavailable and the NOSUSPEND option is specified, or there is an active HANDLE CONDITION ENQBUSY.

Default action: wait for the resource to become available.

INVREQ

RESP2 values: CVDA values are:

- 2 The MAXLIFETIME option is set with an incorrect CVDA.

Default action: terminate the task abnormally.

LENGERR

RESP2 values:

- 1 The value specified for the LENGTH option is outside the range 1 through 255.

Default action: terminate the task abnormally.

Examples

Two tasks, enqueueing on the same resource and specifying a data area that is the resource, must refer to the same location in storage. They could both, for example, refer to the same location in the CWA.

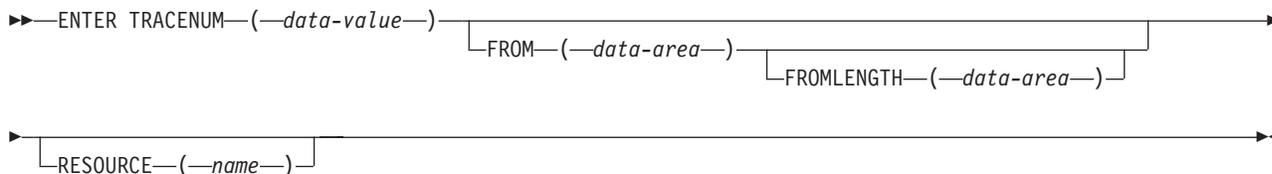
```
EXEC CICS ENQ
      RESOURCE(RESNAME)
```

Two tasks, enqueueing on the same resource and specifying a data area that contains a unique argument, can refer to the same location or to different locations, but the contents of the locations must be the same. The length must be supplied in the LENGTH option. For example, if you define the resource EMPNAME to be the name of an employee, you could use the ENQ command as follows:

```
EXEC CICS ENQ
      RESOURCE(EMPNAME)
      LENGTH(8)
      :
```

ENTER TRACENUM

Write a trace entry.



Conditions: INVREQ, LENGERR

Description

ENTER TRACENUM writes a trace entry.

For information about the trace entry format, see *CICS/400 Problem Determination*.

See “Trace” on page 111 for further information about this command.

Options

FROM(*data-area*)

specifies a data area whose contents are to be entered into the data field of the trace entry. If you omit the FROM option, two fullwords of binary zeros are supplied.

FROMLENGTH(*data-area*)

specifies a halfword binary data area containing the length of the trace data. The value must be in the range 0 through 4000 bytes. If FROMLENGTH is not specified, a length of 8 bytes is assumed.

RESOURCE(*name*)

specifies an 8-character name to be entered into the resource field of the trace entry.

TRACENUM(*data-value*)

specifies as a halfword binary value the trace identifier for a user trace entry. The value must be in the range 0 through 199.

Exception Conditions

INVREQ

RESP2 values:

- 1 TRACENUM is outside the range 0 through 199.
- 2 There is no valid trace destination.
- 3 The user trace master flag is set OFF.

Default action: Terminate the task abnormally.

LENGERR

occurs if FROMLENGTH is outside the range 0 through 4000.

Default action: Terminate the task abnormally.

Examples

The following example shows how to specify that a user trace entry is to be produced:

```
EXEC CICS ENTER TRACENUM(123)
      FROM(MSG)
      :
```

►►—EXTRACT ATTRIBUTES—┌──────────────────┐ STATE—(—cvda—)——►

 └─CONVID—(—name—)─┘

Conditions: INVREQ, NOTALLOC

EXTRACT ATTRIBUTES (APPC)

Obtain the state of the APPC conversation.

```
▶▶ EXTRACT ATTRIBUTES [CONVID(—name—)] STATE(—cvda—) ▶▶
```

Conditions: INVREQ, NOTALLOC

Description

EXTRACT ATTRIBUTES extracts conversation state information for APPC mapped conversations.

Options

CONVID(*name*)

identifies the conversation to which the command relates. The 4-character name identifies the token returned by a previously executed ALLOCATE command in EIBRSRCE in the EIB.

By default, the principal facility is assumed.

STATE(*cvda*)

gets the state of the transaction program.

For a complete list of the CVDA values that can be returned on APPC commands and for information about receiving and testing these values, see “CICS-value data areas (CVDAs)” on page 309.

Exception Conditions

INVREQ

RESP2 values:

200 A distributed program link server application explicitly, or implicitly by default, specified the function-shipping session (its principal facility) on the CONVID option.

also occurs (RESP2 not set) in any of the following situations:

- The CONVID value was obtained by an ASSIGN FACILITY command. However, the principal facility is *not* an APPC conversation.

Default action: terminate the task abnormally.

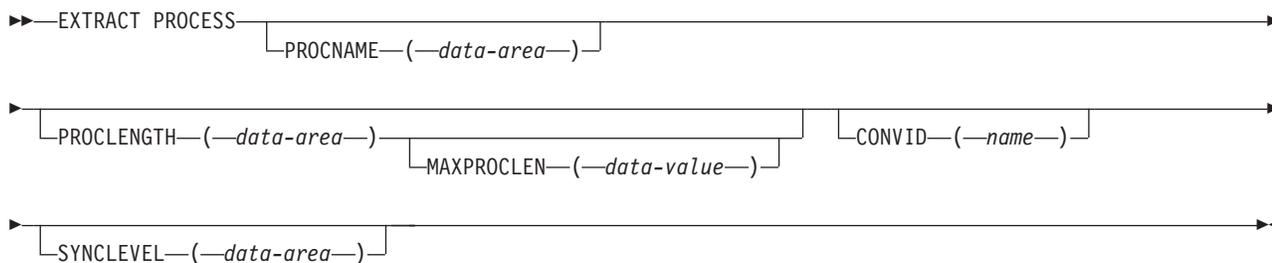
NOTALLOC

occurs if the specified CONVID value does not relate to a conversation owned by the application.

Default action: terminate the task abnormally.

EXTRACT PROCESS

Retrieve values from APPC conversation attach header.



Conditions: INVREQ, LENGERR, NOTALLOC

Description

EXTRACT PROCESS lets an application program access conversation-related data, specified to CICS when the program is attached. The attach receiver does not have to execute an EXTRACT PROCESS command unless it requires this information.

The EXTRACT PROCESS command is valid only on an APPC conversation that is the principal facility for the task. See part 3 of *CICS for iSeries Intercommunication* for more information.

Options

CONVID(*name*)

identifies the conversation to which the command relates. The 4-character name identifies the token representing the principal session (EIBTRMID).

If this option is omitted, the principal facility for the task is used by default.

MAXPROCLEN(*data-value*)

specifies the buffer length of PROCNAME. If MAXPROCLEN is not specified, the buffer is assumed to have 32 bytes.

PROCLENGTH(*data-area*)

specifies a halfword data area that is set by CICS to the length of the process name. If PROCNAME is specified, this option must be specified.

PROCNAME(*data-area*)

specifies the data area to receive the process name specified by the remote system that caused the task to start. The data area can be 1–64 bytes long. The process name is padded on the right with blanks if it is too short. The PROCNAME data area should not be shorter than the MAXPROCLEN value.

SYNCLEVEL(*data-area*)

specifies a halfword data area that is set by CICS to the SYNCLEVEL value. For further information about synchronization levels, see the *CICS Family: Interproduct Communication* book and the *CICS for iSeries Intercommunication* book.

Exception Conditions

INVREQ

RESP2 values:

200 A distributed program link server application specified the function-shipping session (its principal facility) on the CONVID option.

also occurs (RESP2 not set) in any of the following situations:

- EXTRACT PROCESS has been used on a conversation other than APPC mapped.
- EXTRACT PROCESS has been used on a conversation that was not started by input from the network and whose principal facility is not an APPC session.
- The CONVID value was obtained by an ASSIGN FACILITY command. However, the principal facility is *not* an APPC session.

Default action: terminate the task abnormally.

LENGERR

occurs if the actual length of PROCNAME is greater than MAXPROCLLEN, or greater than 32 bytes if MAXPROCLLEN is not specified.

Default action: terminate the task abnormally.

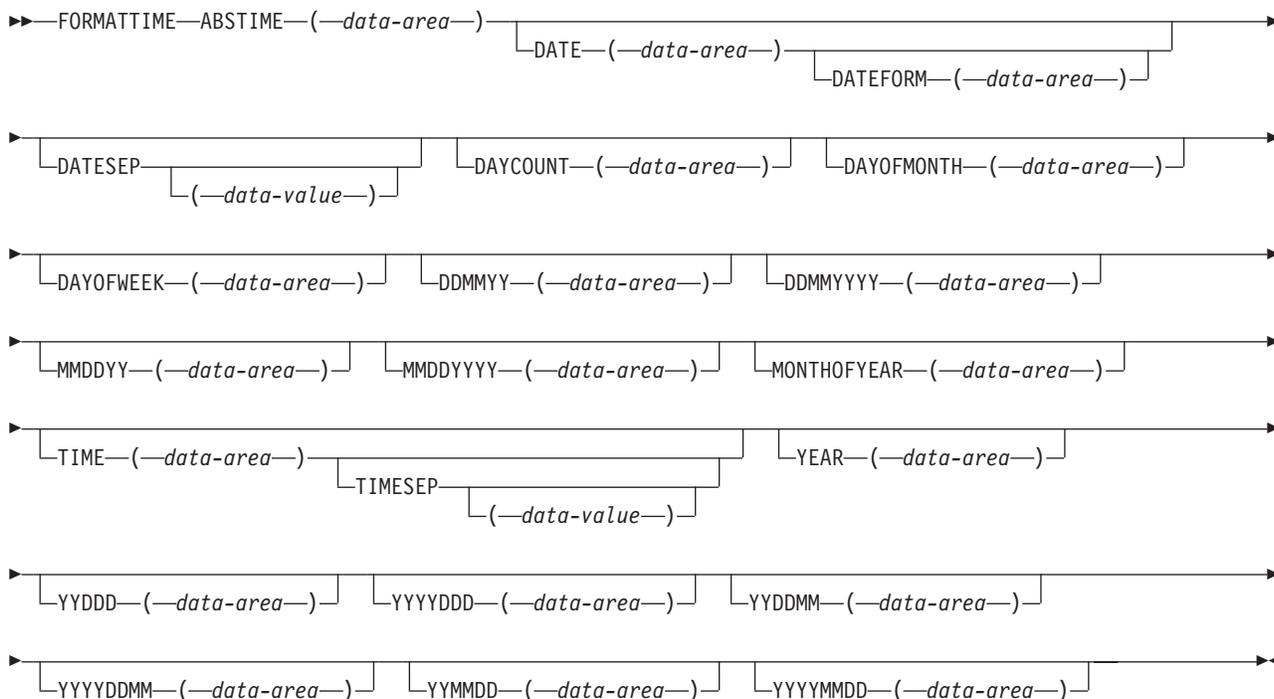
NOTALLOC

occurs if the specified CONVID value specified does not relate to a conversation owned by the application.

Default action: terminate the task abnormally.

FORMATTIME

Transform absolute date and time into a specified format.



Condition: INVREQ

Description

FORMATTIME transforms the absolute date and time into any of a variety of formats. Normally, the ABSTIME argument is the value returned by an ASKTIME ABSTIME command.

To obtain an elapsed time in a particular format, the ABSTIME data value can be the difference between two values returned by ASKTIME, and options such as DAYCOUNT(d) and TIME(t) can be specified.

Options

ABSTIME(*data-area*)

specifies the data value for the time, in packed decimal, since 00:00 hours on 1 January 1900 (in milliseconds rounded to the nearest hundredth of a second) that is to be converted to an alternative format.

The format of the parameter is:

```
COBOL: PIC S9(15) COMP-3
ILE C: char abs_time[8];
```

DATE(*data-area*)

specifies the variable that is to receive the date in the format specified in the DATFORM system initialization parameter. The returned value is in 8-character format, and a separator is present or not depending on the DATESEP option. You should normally use this option only when a date is needed for output purposes. Where a date is needed for analysis, you should request the date in explicit form, for example, using the MMDDYY option.

DATEFORM(*data-area*)

specifies the format of the installation-defined date. CICS returns YYMMDD, DDMMYY, or MMDDYY (six characters) according to the DATFORM system initialization parameter.

DATESEP(*data-value*)

specifies the character to be inserted as the separator between the year and the month, and between the day and the month; or between the year and the day if form YYDDD or YYYYDDD is specified.

If you omit this option, no separator is supplied. If you omit "data-value", a slash (/) is assumed as the separator.

DAYCOUNT(*data-area*)

returns the number of days since 1 January 1900 (day 1), as a fullword binary number. This is useful if you need to compare the current date with a previous date that has, for example, been stored in a file.

DAYOFMONTH(*data-area*)

returns the number of the day in the month as a fullword binary number.

DAYOFWEEK(*data-area*)

returns the relative day number of the week as a fullword binary number: Sunday=0, Saturday=6. This number can be converted to a textual form of day in any language.

DDMMYY(*data-area*)

specifies the 8-character user field where CICS is to return the date, in day/month/year format (for example, 21/10/95).

DDMMYYYY(*data-area*)

specifies the 10-character user field where CICS is to return the date, in day/month/year format (for example, 21/10/1995).

MMDDYY(*data-area*)

specifies the 8-character user field in which CICS is to return the date, in month/day/year format (for example, 10/21/95).

MMDDYYYY(*data-area*)

specifies the 10-character user field in which CICS is to return the date, in month/day/year format (for example, 10/21/1995).

MONTHOFYEAR(*data-area*)

"data-area" is set to the relative month number of the year as a fullword binary number (January=1, December=12). You can convert this number, in your application program, to the name of the month in any language.

TIME(*data-area*)

"data-area" is set as an 8-character field to the current 24-hour clock time in the form hh:mm:ss, where the separator is specified by the TIMESEP option.

TIMESEP(*data-value*)

specifies the character to be used as the separator in the returned time. If you omit this option, no separator is assumed and six bytes are returned in an 8-character field. If you omit the "data-value", a colon (:) is used as a separator.

YEAR(*data-area*)

specifies the full 4-figure number of the year as a fullword binary number (for example, 1995, 2001).

YYDDD(*data-area*)

specifies the 6-character user field where CICS is to return the date, in year/day format (for example, 95/301).

YYYYDDD(*data-area*)

specifies the 8-character user field where CICS is to return the date, in year/day format (for example, 1995/301).

YYDDMM(*data-area*)

specifies the 8-character user field where CICS is to return the date, in year/day/month format (for example, 95/30/10).

YYYYDDMM(*data-area*)

specifies the 10-character user field where CICS is to return the date, in year/day/month format (for example, 1995/30/10).

YYMMDD(*data-area*)

specifies the 8-character user field where CICS is to return the date, in year/month/day format (for example, 95/10/21).

YYYYMMDD(*data-area*)

specifies the 10-character user field where CICS is to return the date, in year/month/day format (for example, 1995/10/21).

Exception Conditions

INVREQ

RESP2 values:

- 1 The ABSTIME option is in an incorrect form.

also occurs (RESP2 not set) in any of the following situations:

Exception Conditions

INVREQ

RESP2 values:

- 200** A distributed program link server application specified the function-shipping session (its principal facility) on the CONVID option.

also occurs (RESP2 not set) in any of the following situations:

- The CONVID option is omitted and the principal facility is not an APPC session.

Default action: terminate the task abnormally.

NOTALLOC

occurs if the specified CONVID value does not relate to a conversation owned by the application.

Default action: terminate the task abnormally.

FREEMAIN

Release main storage acquired by a GETMAIN command.

►►—FREEMAIN—DATA—(*—data-area—*)—◄◄

Condition:
INVREQ

Description

FREEMAIN releases main storage previously acquired by a GETMAIN command issued by the application. If the task that acquired the storage does not release it, CICS releases it at task end, unless the GETMAIN command specified the SHARED option. In this case, the storage remains allocated until another task issues a FREEMAIN to release it.

Refer to Chapter 22, “Storage control,” on page 213 for more information.

Options

DATA(*data-area*)

specifies the location of the main storage to be released.

This storage must have been acquired by a previous GETMAIN command.

The length of storage released is the length obtained by the GETMAIN and not necessarily the length of the data area.

Note that, in COBOL, when using the DATA option, you must specify a data area located at the start of the storage that was acquired by the GETMAIN command; you do not specify the pointer reference that was set to the address of the acquired storage. So in COBOL/400 it must be a data name.

Exception Conditions

INVREQ

RESP2 values:

- 1 The storage specified in the DATA option is not addressed by a pointer returned by a GETMAIN command.

Default action: Terminate the task abnormally.

Examples

Here are some examples of the use of the FREEMAIN command:

COBOL

```
DATA DIVISION.
WORKING-STORAGE SECTION.
  01 WORKING-STORAGE.
    02 AREA-POINTER  USAGE IS POINTER.

LINKAGE SECTION.
  01 AREA           PIC X(100).

PROCEDURE DIVISION.

  EXEC CICS GETMAIN SET(AREA-POINTER)
    LENGTH(100)
  END-EXEC.
  SET ADDRESS OF AREA TO AREA-POINTER.
  :
  :
  EXEC CICS FREEMAIN DATA(AREA)
  END-EXEC.
  :
  :
  EXEC CICS RETURN
  END-EXEC.
```

C

```
main()
{
  char    *buffer;

  EXEC CICS GETMAIN
    SET(buffer)
    LENGTH(100);

  buffer[2] = 'a';
  :
  :
  EXEC CICS FREEMAIN DATA(buffer);

  EXEC CICS RETURN;
}
```


The maximum length that you can specify is the size of the storage space from which the request is satisfied or 65 488 bytes, whichever is smaller.

If the requested length is greater than the storage space, the LENGERR condition occurs. If it is not greater than the storage space but is more than is available, the NOSTG condition occurs.

INITIMG(*data-value*)

specifies an optional 1-byte initialization value. If you specify INITIMG, CICS sets every byte of the acquired storage to the bit string you provide. Otherwise, CICS does not initialize the storage. In COBOL programs, you must use a data area rather than a data value to define the initialization bit string.

LENGTH(*data-value*)

specifies as a signed halfword binary value the number of bytes of storage required. The maximum length you can specify is 32KB. If you need more than 32KB of storage, use the FLENGTH option.

If the requested length is zero, the LENGERR condition occurs. If it is greater than the amount of storage available, the NOSTG condition occurs.

NOSUSPEND

changes the CICS default action for the NOSTG condition, causing CICS to return control to the task without the requested storage instead of suspending the task until the storage becomes available.

SET(*ptr-ref*)

specifies a pointer reference to be set to the address of the acquired main storage. The pointer is set to the first usable byte of the free space (not to the CICS control information that precedes it).

SHARED

prevents the automatic release of storage obtained by a GETMAIN command at the end of the task that requested it. This enables task-to-task communication. An area obtained with SHARED is not released until a corresponding FREEMAIN is issued, by the requesting task or another task.

Exception Conditions

LENGERR

RESP2 values:

- 1 The FLENGTH value is less than 1 or greater than the length of the free spaces from which the request can be satisfied.
- The LENGTH value is zero.

Default action: Terminate the task abnormally.

NOSTG

RESP2 values:

- 2 More storage is requested than is currently available.

Default action: When SHARED storage is requested, either suspend the task until enough main storage becomes free to satisfy the request or, if the NOSUSPEND option is specified, return control to the task without the requested storage. Suspended tasks may subsequently terminate abnormally if purged (abend code ASCP) or under deadlock timeout control (abend code AKCS).

When non-shared storage is requested, the task is terminated abnormally. This is because non-shared storage is a private resource and none would become available even if the task were suspended.

Examples

The following example shows how to get a 1024-byte area of main storage and initialize it to spaces:

```
EXEC CICS GETMAIN SET(PTR)
      LENGTH(1024) INITIMG(BLANK)
      :
```

You must define BLANK in your program as the character representing a space.

HANDLE ABEND

Handle an abnormal termination exit



Conditions: NOTAUTH, PGMIDERR

Description

The HANDLE ABEND command is used to activate, cancel, or reactivate an exit for abnormal termination processing. You can suspend (and later restore) the command by means of the PUSH HANDLE and POP HANDLE commands as described in “How to use EXEC CICS PUSH HANDLE and POP HANDLE commands” on page 95.

The HANDLE ABEND command cannot intercept abends resulting from AS/400 machine event errors.

For COBOL programs, when the label specified in a HANDLE ABEND LABEL command is to receive control, control is returned to the HANDLE ABEND command and a branch occurs to the specified label.

The effect of a HANDLE ABEND command is ended when the issuing program terminates by issuing an EXEC CICS RETURN command.

The effect of a HANDLE ABEND command is suspended when the issuing program transfers control by issuing an EXEC CICS LINK command, and is restored when control is returned from the linked program.

A HANDLE ABEND PROGRAM command remains in effect when the issuing program transfers control by issuing an EXEC CICS XCTL command. The scope of the HANDLE command has the same relation to the new program as it formerly had to the issuing program. Note that this does not apply to HANDLE ABEND LABEL commands.

Refer to Chapter 9, “Abnormal termination recovery,” on page 109 for more information.

Options

CANCEL

specifies that a previously established exit at the logical level of the application program in control is to be canceled.

LABEL(*label*)

specifies the program label of an abnormal termination exit routine to receive control if the task is terminated abnormally.

The LABEL option can be used in COBOL programs only.

PROGRAM(*name*)

specifies the name of an abnormal termination exit program to receive control if the task is terminated abnormally. The name must be alphanumeric, up to 8 characters long, and must have been defined in the processing program table (PPT) as a local program.

The program named in this option should *always* terminate with an abend, except when handling abends generated as a result of application program logic.

RESET

specifies that an exit canceled by a HANDLE ABEND CANCEL command, or by CICS, is to be reactivated.

This option is usually issued by an abnormal termination exit routine.

Exception Conditions

NOTAUTH

occurs if a resource security check has failed on PROGRAM(*name*).

Default action: Terminate the task abnormally.

PGMIDERR

RESP2 values:

- 1 The program does not have an installed resource definition.
- 2 The program is disabled.
- 9 The program is defined as remote.

Default action: Terminate the task abnormally.

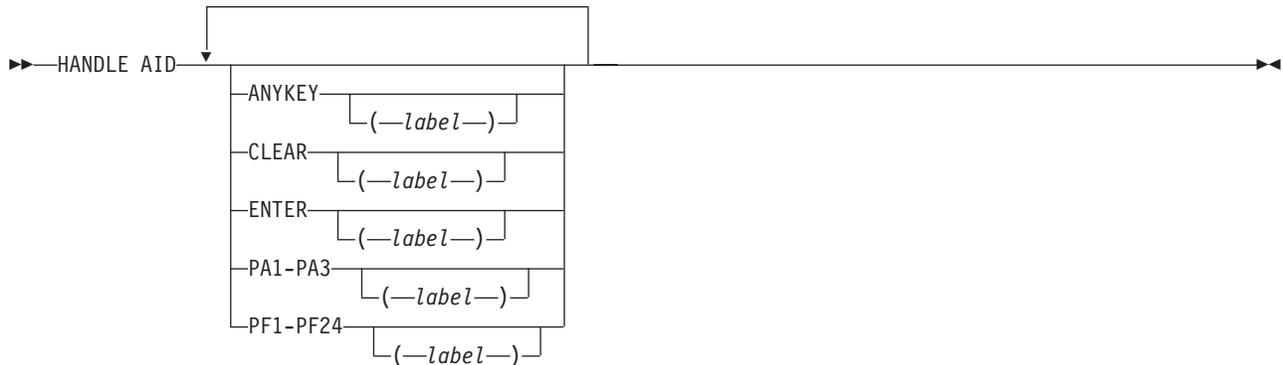
Examples

The following example shows how to establish a program as an exit:

```
EXEC CICS HANDLE ABEND  
PROGRAM('EXITPGM')  
END-EXEC.
```

HANDLE AID

Handle attention identifiers (AIDs).



Condition: INVREQ

Description

HANDLE AID is used to specify the label to which control is to be passed when an AID is received from a display device. Control is passed after the input command is completed; that is, after any data received in addition to the AID has been passed to the application program.

To cause an AID to be ignored, issue a HANDLE AID command that specifies the associated option *without* a label. This deactivates the effect of that option in any previously-issued HANDLE AID command.

No more than 16 options are allowed in the same command.

The ILE C language does not support HANDLE AID.

The options that can be specified are:

- ANYKEY (any PA key, any PF key, or the CLEAR key, but not ENTER)
- CLEAR (for the key of that name)
- ENTER (for the key of that name)
- PA1, PA2, or PA3 (any of the program access keys)
- PF1 through PF24 (any of the program function keys)

If a task is initiated from a terminal by means of an AID, the first RECEIVE command in the task does not read from the terminal but copies only the input buffer (even if the length of the data is zero) so that control may be passed by means of a HANDLE AID command for that AID.

For the standard attention identifier list (DFHAID), and the standard attribute and printer control character list (DFHBMSCA), see Appendix B, “BMS-related constants,” on page 545.

See “Handling attention identifiers (EXEC CICS HANDLE AID)” on page 173 for more information.

Exception Conditions

INVREQ

RESP2 values:

200 The command was issued by a DPL server program.

Default action: terminate the task abnormally.

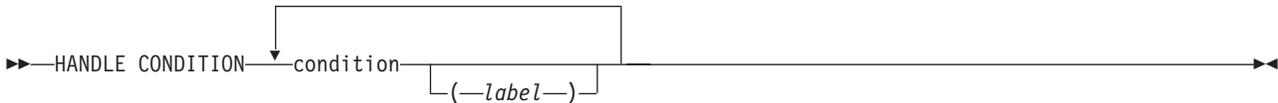
Examples

The following example shows a HANDLE AID command that specifies one label for the PA1 key; and a second label for CLEAR, PA2, PA3, and all the PF keys except PF10. If a PF10 AID is received or ENTER is pressed, control returns to the application program at the instruction immediately following the input command.

```
EXEC CICS HANDLE AID PA1(LAB1)
      ANYKEY(LAB2) PF10
```

HANDLE CONDITION

Handle conditions.



Description

HANDLE CONDITION is not supported for C programs.

For information about the conditions, see Appendix A, "EXEC interface block," on page 529.

You use this HANDLE CONDITION to specify the label to which control is to be passed if a condition occurs. You must include the name of the condition and, optionally, a label to which control is to be passed if the condition occurs. You must ensure that the HANDLE CONDITION command is executed before the command that may give rise to the associated condition.

You cannot include more than sixteen conditions in the same command; the conditions must be separated by at least one space. You may specify additional conditions in further HANDLE CONDITION commands.

The HANDLE CONDITION command for a given condition applies only to the program in which it is specified. The HANDLE CONDITION command:

- Remains active while the program is running, or until:
 - An IGNORE CONDITION command for the same condition is encountered, in which case the HANDLE CONDITION command is overridden
 - Another HANDLE CONDITION command for the same condition is encountered, in which case the new command overrides the previous one
- Is temporarily deactivated by the NOHANDLE, RESP, or RESP2 option on a command

If a condition occurs that was not specified in a HANDLE CONDITION or IGNORE CONDITION command, the default action for the condition is taken unless that action is to terminate the task abnormally, in which case the ERROR condition occurs. If the ERROR condition was specified in a HANDLE CONDITION or IGNORE CONDITION command, the action (possibly none) for ERROR is taken. You can use the ERROR condition in a HANDLE CONDITION list to specify that all other errors pass control to the same label.

Refer to Chapter 6, “Dealing with exception conditions,” on page 87 for more information.

Options

condition(*label*)

specifies the name of the condition; “label” specifies the program label to which control is to be passed if the condition occurs.

If you omit “label”, any HANDLE CONDITION command in effect for that condition is deactivated, and the default action for the condition is taken if the condition occurs.

Examples

The following example shows how to handle the conditions, including DUPREC and LENGERR, that can occur when you use a WRITE command to add a record to a file. Suppose that you want DUPREC to be handled as a special case; that you want standard system action (that is, to terminate the task abnormally) to be taken for LENGERR; and that you want all other errors to be handled by the error routine ERRHANDL. You would code:

```
EXEC CICS HANDLE CONDITION
          ERROR(ERRHANDL)
          DUPREC(DUPRTN) LENGERR
END-EXEC.
```

IGNORE CONDITION

Ignore conditions.



Description

IGNORE CONDITION is not supported for ILE C programs.

For information about the conditions, see Appendix A, “EXEC interface block,” on page 529.

You use IGNORE CONDITION to specify that no action is to be taken if a condition occurs (that is, control returns to the instruction following the command that has failed to execute, and the EIB is set). Execution of a command could result in several conditions being raised. CICS checks these in a predetermined order and only the first one that is not ignored (by your IGNORE CONDITION command) is passed to your application program.

The IGNORE CONDITION command for a given condition applies only to the program in which it is specified, and it remains active while the program is being executed, or until a HANDLE CONDITION command for the same condition is encountered, in which case the IGNORE CONDITION command is overridden.

You cannot include more than sixteen conditions in the same command; the conditions must be separated by at least one space. You may specify additional conditions in further IGNORE CONDITION commands.

Refer to Chapter 6, “Dealing with exception conditions,” on page 87 for more information.

Options

condition

specifies the name of the condition to be ignored.

ISSUE ABEND

Abend the mapped conversation with an APPC partner.



Conditions: INVREQ, NOTALLOC, TERMERR

Description

ISSUE ABEND abnormally ends the conversation. The partner transaction sees the TERMERR condition. See part 3 of *CICS for iSeries Intercommunication* for more information.

Options

CONVID(name)

identifies the conversation to be abended. The 4-character name identifies the token returned by a previously executed ALLOCATE command in the EIBRSRCE field of the EIB.

If this option is omitted, the principal facility for the task is used by default.

STATE(cvda)

gets the state of the current conversation.

For a complete list of the CVDA values that can be returned on APPC commands and for information about receiving and testing these values, see “CICS-value data areas (CVDAs)” on page 309.

Exception Conditions

INVREQ

RESP2 values:

200 A distributed program link server application specified the function-shipping session (its principal facility) on the CONVID option.

also occurs (RESP2 not set) in any of the following situations:

- The CONVID value was obtained by an ASSIGN FACILITY command. However, the principal facility is *not* an APPC conversation.

Default action: terminate the task abnormally.

NOTALLOC

occurs if the specified CONVID value relates to a conversation that is not owned by the application.

Default action: terminate the task abnormally.

TERMERR

occurs for a session-related error. Any action on that conversation other than a FREE command causes an ATCV abend.

Default action: terminate the task abnormally with abend code ATNI.

ISSUE CONFIRMATION

Issue a positive response to a SEND CONFIRM on an APPC mapped conversation.



Conditions: INVREQ, NOTALLOC, TERMERR

Description

ISSUE CONFIRMATION allows an application program to respond positively when the CONFIRM option has been specified on a SEND command executed by a partner transaction. See part 3 of *CICS for iSeries Intercommunication* for more information.

Options

CONVID(*name*)

identifies the conversation in which to send the response. The 4-character name identifies the token returned by a previously executed ALLOCATE command in the EIBRSRCE field of the EIB. If this option is omitted, the principal facility for the task is used by default.

STATE(*cvda*)

gets the state of the current conversation.

For a complete list of the CVDA values that can be returned on APPC commands and for information about receiving and testing these values, see “CICS-value data areas (CVDAs)” on page 309.

Exception Conditions

INVREQ

RESP2 values:

- 200** A distributed program link server application specified the function-shipping session on the CONVID option.

also occurs (RESP2 not set) in any of the following situations:

- ISSUE CONFIRMATION is used on a conversation that is sync level 0.

- The CONVID value was obtained by an ASSIGN FACILITY command. However, the principal facility is *not* an APPC conversation.

Default action: terminate the task abnormally.

NOTALLOC

occurs if the specified CONVID value relates to a conversation that is not owned by the application.

Default action: terminate the task abnormally.

TERMERR

occurs for a session-related error. Any action on that conversation other than a FREE causes an ATCV abend.

Default action: terminate the task abnormally with abend code ATNI.

ISSUE ERASEAUP

Erase all unprotected fields of a 3270 buffer.

ISSUE ERASEAUP

▶—ISSUE ERASEAUP—▶

Conditions: INVREQ, TERMERR

Description

ISSUE ERASEAUP erases unprotected fields by:

1. Clearing all unprotected fields to nulls (X'00')
2. Resetting modified data tags in each unprotected field to zero
3. Positioning the cursor to the first unprotected field
4. Restoring the keyboard.

You can use the ISSUE ERASEAUP command for the following types of 3270 logical units:

- 3270-display logical unit
- 3270-printer logical unit

If you issue this command for a 5250 device, results are unpredictable.

Exception Conditions

INVREQ

RESP2 values:

200 The command is issued in a DPL server program.

Default action: terminate the task abnormally.

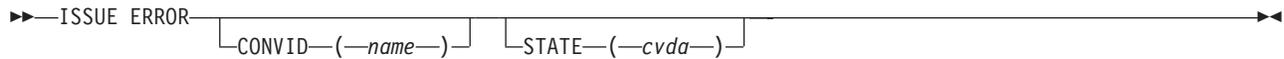
TERMERR

occurs if there is a terminal-related error.

Default action: terminate the task abnormally with abend code ATNI.

ISSUE ERROR

Inform APPC mapped conversation partner of error.



Conditions: INVREQ, NOTALLOC, SIGNAL, TERMERR

Description

ISSUE ERROR allows an application program to inform a process in a connected APPC system that some program-detected error has occurred. For example, a remote CICS application is notified by having EIBERR set, with EIBERRCD=X'0889'. The actions required to recover from the error are the responsibility of logic contained in both application programs. The application program can use this command to respond negatively when the CONFIRM option has been specified on a SEND command executed by a process in a connected APPC system.

Options

CONVID(*name*)

identifies the conversation to which the command relates. The 4-character name identifies either the token returned by a previously executed ALLOCATE command in EIBRSRCE in the EIB.

If this option is omitted, the principal facility for the task is used by default.

STATE(*cvda*)

gets the state of the current conversation. For a complete list of the CVDA values that can be returned on APPC commands and for information about receiving and testing these values, see "CICS-value data areas (CVDA)" on page 309.

Exception Conditions

INVREQ

RESP2 values:

- 200** A distributed program link server application specified the function-shipping session on the CONVID option.

also occurs (RESP2 not set) in any of the following situations:

- The CONVID value was obtained by an ASSIGN FACILITY command. However, the principal facility is *not* an APPC conversation.

Default action: terminate the task abnormally.

NOTALLOC

occurs if the specified CONVID value does not relate to a conversation owned by the application.

Default action: terminate the task abnormally.

SIGNAL

occurs when an inbound SIGNAL data-flow control command is received from a partner transaction. EIBSIG is always set when an inbound signal is received.

Default action: ignore the condition.

TERMERR

occurs for a session-related error. Any action on that conversation other than a FREE command causes an ATCV abend.

Default action: terminate the task abnormally with abend code ATNI.

ISSUE PREPARE

Issue the first flow of a syncpoint request on an APPC mapped conversation.

```
▶▶—ISSUE PREPARE—┌CONVID—(—name—)—┐┌STATE—(—cvda—)—┐▶▶
```

Conditions: INVREQ, NOTALLOC, TERMERR

Description

ISSUE PREPARE applies only to distributed transaction processing over APPC links. It enables a syncpoint initiator to prepare a syncpoint slave for syncpointing by sending only the first flow (prepare-to-commit) of the syncpoint exchange. Depending on the reply from the syncpoint slave, the initiator can proceed with the syncpoint by issuing a SYNCPOINT command, or initiate back-out by issuing a SYNCPOINT ROLLBACK command.

Options

CONVID(*name*)

identifies the conversation to which the command relates. The 4-character name identifies either the token returned by a previously executed ALLOCATE command in EIBRSRCE in the EIB, or the token representing the principal facility (returned by a previously executed ASSIGN command).

If this option is omitted, the principal facility is assumed.

STATE(*cvda*)

gets the state of the current conversation. The cvda values returned by CICS are:

```
ALLOCATED
CONFREE
CONFRECEIVE
CONFSEND
FREE
PENDFREE
PENDRECEIVE
RECEIVE
ROLLBACK
SEND
SYNCFREE
SYNCRECEIVE
SYNCSSEND
```


For a complete list of the CVDA values that can be returned on APPC commands and for information about receiving and testing these values, see “CICS-value data areas (CVDA)” on page 309.

Exception Conditions

INVREQ

RESP2 values:

200 A distributed program link server application specified the function-shipping session (its principal facility) on the CONVID option.

also occurs (RESP2 not set) in any of the following situations:

- The CONVID value was obtained by an ASSIGN FACILITY command. However, the principal facility is *not* an APPC conversation.

Default action: terminate the task abnormally.

NOTALLOC

occurs if the specified CONVID value does not relate to a conversation that is owned by the application.

Default action: terminate the task abnormally.

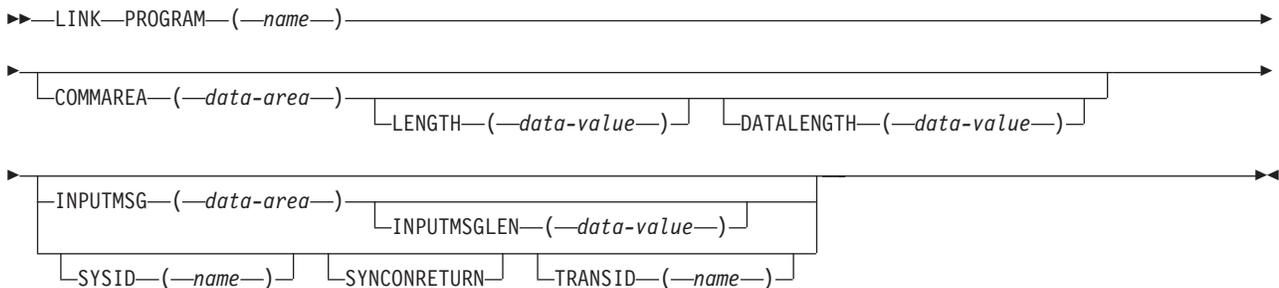
TERMERR

occurs for a session-related error. Any action on that conversation other than a FREE causes an ATCV abend.

Default action: terminate the task abnormally with abend code ATNI.

LINK

Link to another program expecting return.



Conditions: INVREQ, LENGERR, NOTAUTH, PGMIDERR, ROLLEDBACK, SYSIDERR, TERMERR

Description

LINK passes control from an application program at one logical level to an application program at the next lower logical level.

If the linked-to program is not already in main storage, it is loaded. When the RETURN command is executed in the linked-to program, control is returned to the program issuing the LINK command at the next sequential executable instruction.

The linked-to program operates independently of the program that issues the LINK command with regard to handling exception conditions and attention identifiers. For example, the effects of HANDLE CONDITION commands in a

COBOL linking program are not inherited by the linked-to program, but the original HANDLE CONDITION commands are restored on return to the linking program. See Figure 39 on page 201 for an illustration of the concept of logical levels.

You can use the HANDLE ABEND command in COBOL programs to deal with abnormal terminations in other link levels. See Chapter 9, “Abnormal termination recovery,” on page 109 for further details about the relationship between the LINK and HANDLE ABEND commands.

See Chapter 20, “Program control,” on page 199 for more information about Program Control.

If you specify a remote CICS system name on the SYSID option (with or without the associated TRANSID and SYNCONRETURN options), the link is known as a **distributed program link** (DPL). DPL is also used if the PPT entry specifies a remote program. In response to a distributed program link, the local CICS/400 system (the **client system**) ships the link request to the remote system (the **server system**). The server system executes the linked-to program (the **server program**) on behalf of the program issuing the link request (the **client program**).

A server program running in the server system is restricted to a DPL subset of the CICS API. These restrictions relate mainly to terminals and logical units. Briefly, the server program cannot issue:

- Commands that address the TCTUA (that is, ADDRESS with the TCTUA option).
- ASSIGN commands with terminal-related options. (See the ASSIGN command on page 327 for details of the restricted options for CICS/400.)
- Authentication commands (that is, SIGNON and SIGNOFF). These are not supported by CICS/400.
- Terminal control commands, and APPC commands referring to the principal facility (see below for further details).
- Batch data interchange commands (not supported by CICS/400).
- BMS commands.
- Syncpoint commands (that is, SYNCPOINT with or without the ROLLBACK option) unless SYNCONRETURN was specified on the LINK command.

For lists of the APPC, terminal control, and BMS commands supported by CICS/400, see “Commands by function” on page 319.

APPC commands are restricted only when referring to the principal facility; that is, when either the CONVID option is omitted on commands that include this option (all except ALLOCATE) or the CONVID option states explicitly the conversation identifier of the principal facility (EIBTRMID value). This allows a server program to initiate a conversation with another system (possibly even back to the client system, although this requires a separate session).

If a restricted command is issued in a DPL server program, the INVREQ condition occurs and a RESP2 value of 200 is set. If this condition is not catered for by a HANDLE CONDITION command, an IGNORE CONDITION command, or the NOHANDLE option, the server program task terminates abnormally with abend code ADPL.

For details of the restricted DPL subset of the API, see Figure 38 on page 185.

Abends in the server program If a DPL server program abends, the abend code is returned to the client program. If the client program is not written to handle the abend returned by the server program, the client program abends with the same abend code returned by the server program.

For more information about DPL, see the Chapter 15, “Intercommunication considerations,” on page 175.

Options

COMMAREA(*data-area*)

specifies a communication area that is to be made available to the invoked program. In this option, a pointer to the data area is passed. In a COBOL invoked program, you must give this data area the name DFHCOMMAREA. (See “Passing data to other programs” on page 201.) In a C receiving program, this data area must be referenced by an EXEC CICS ADDRESS COMMAREA command.

DATALENGTH(*data-value*)

specifies as a halfword binary value the length in bytes of a contiguous area of storage, from the start of the COMMAREA, to be passed to the invoked program. If the amount of data being passed in a COMMAREA is small but the COMMAREA itself is large, in the interest of performance you should specify DATALENGTH so that the linked-to program need receive only the requested data.

INPUTMSG(*data-area*)

specifies data to be supplied to the invoked program when it first issues an EXEC CICS RECEIVE command. This data remains available until the execution of an EXEC CICS RECEIVE or EXEC CICS RETURN command.

An invoked program can invoke a further program and so on, creating a chain of linked programs.

If a linked-to chain exists, CICS supplies the INPUTMSG data to the first EXEC CICS RECEIVE command executed in the chain. If control returns to the program that issued the EXEC CICS LINK command with the INPUTMSG option before the INPUTMSG data has been accepted by an EXEC CICS RECEIVE command, CICS assumes that an EXEC CICS RECEIVE command has been issued. This means that the original INPUTMSG data is no longer available.

INPUTMSG cannot be used at the same time as DATALENGTH.

Note: You cannot specify the INPUTMSG option for a remote program. That is, you cannot specify the INPUTMSG and the SYSID options on the same EXEC CICS LINK command, and you cannot specify the INPUTMSG option if the SYSID specified on the program resource definition of the program to which you are linking, is anything other than *NONE. In either case, the INVREQ condition is raised with a RESP2 value of 19.

See Chapter 20, “Program control,” on page 199 for more information about the INPUTMSG option.

INPUTMSGLEN(*data-value*)

specifies as a halfword binary value the length of the INPUTMSG data. If the value is negative, zero is assumed.

LENGTH(*data-value*)

specifies as a halfword binary value the length in bytes of the communication area. For a description of a safe upper limit, see “LENGTH options” on page 312..

PROGRAM(*name*)

specifies the identifier of the program to which control is to be passed unconditionally, and from which return is expected. The name must be alphanumeric, up to 8 characters long, and must have been defined as a program to CICS.

If the SYSID option specifies a remote system, the linked-to program is the server program in the server system.

SYNCONRETURN

specifies that the server system named on the SYSID option is to take a syncpoint on successful completion of the server program. This option is independent of any syncpoint flows across the link.

If you omit SYNCONRETURN, the default synchronization level is the same as that of the link.

SYSID(*name*)

specifies the system name of a CICS server system to which the program link request is to be routed. The name can be up to four characters. Although this option is primarily for a distributed program link, you can specify the name of the local CICS system, in which case CICS treats the LINK request as a normal local link request.

The SYSID specified on the LINK command takes priority over any remote system specified on the program resource definition.

TRANSID(*name*)

specifies the name of the transaction that the remote system is to attach, and under which it is to run the server program. If you omit the TRANSID option, the server system attaches either the CSMI or the CVMI transaction by default.

The transaction name you specify on the LINK command takes priority over any transaction specified on the program resource definition.

Exception Conditions

INVREQ

RESP2 values:

- The INPUTMSG option is issued for a program that is not associated with a terminal.
- The INPUTMSG option is not supplied with an address.
- 8 The INPUTMSG option is issued for a program that is associated with an APPC logical unit.
- 14 The SYNCONRETURN option is specified, but the program issuing the link request (the client program) is already in conversation with a mirror task (that is, a logical unit of work (LUW) is in progress) in the remote system specified on the SYSID option. In this case, the client program is in an incorrect state to support the SYNCONRETURN option.
- 15 The program issuing the link request is already in conversation with a mirror task and the specified TRANSID value differs from the transaction identifier of the active mirror.

- 16 The specified TRANSID is all blanks.
- 19 The INPUTMSG option is issued for a program that is the subject of a DPL request; that is, SYSID is also specified, or REMOTESYSTEM(name) is specified on the PROGRAM resource definition.
- 200 The INPUTMSG option is issued for a program that is executing as a DPL server program.

Default action: Terminate the task abnormally.

LENGERR

RESP2 values:

- The INPUTMSGLEN value is outside the range 1 through 32 767.
 - The length specified on the LENGTH option is greater than the length of the data area specified in the COMMAREA option, and while that data was being copied a destructive overlap occurred because of the incorrect length.
- 11 A negative LENGTH value is supplied.
 - 12 A negative DATALENGTH value is supplied.
 - 13 The length specified on the DATALENGTH option is greater than the length specified on the LENGTH option.

Default action: Terminate the task abnormally.

NOTAUTH

occurs if a resource security check has failed on PROGRAM(name).

Default action: Terminate the task abnormally.

PGMIDERR

RESP2 values:

- 1 The program does not have an installed resource definition.
- 2 The program is disabled.
- 3 The program cannot be loaded.

Default action: Terminate the task abnormally.

ROLLEDBACK

occurs if the SYNCONRETURN option is specified and the server program is unable to take a syncpoint successfully. The server program has taken a rollback, and all changes made to recoverable resources in the remote system, within the current LUW, are backed out.

Default action: Terminate the task abnormally.

SYSIDERR

RESP2 values:

- 18 The SYSID option specifies either a name that is not defined in the terminal control system table (TCS), or a system to which the link is closed.

Note: There is no local queuing in the event of a SYSIDERR.

Default action: Terminate the task abnormally.

TERMERR

RESP2 values:

- 17 There is an unrecoverable error during the conversation with the mirror (for example, if the session fails, or if the server system fails).

Default action: Terminate the task abnormally.

Examples

The following example shows how to request a link to an application program called PROG1:

```
EXEC CICS LINK PROGRAM('PROG1')
:
```

LOAD

Provide addressability to resources defined in the PPT.

```
▶▶—LOAD—PROGRAM—(—name—)———|—SET—(—ptr-ref—)|——|—HOLD——|▶▶
```

Conditions: NOTAUTH, PGMIDERR

Description

Provide addressability to resources defined in the PPT. When issued with the SET option, the LOAD command provides pointer addressability to OS/400 *USRSPC objects defined as tables or map sets.

On other CICS platforms, the LOAD command can be used to reduce system overhead by fetching an application program, table, or map set from the library where it resides and loading it into main storage. CICS/400 program management is provided by OS/400 and, therefore, CICS/400 does not require the use of the LOAD command for this purpose.

Each time a LOAD command is issued for the same resource, a use count is incremented by 1.

Refer to Chapter 20, "Program control," on page 199 for more information.

Options

HOLD

specifies, on other CICS platforms, that the loaded program, table, or map set is not to be removed from main storage when the task issuing the LOAD command terminates, but is to be removed only in response to a RELEASE command issued by this task or another task. CICS/400 supports use of the HOLD option and insists on correct usage of the HOLD option and the RELEASE command.

PROGRAM(*name*)

specifies the identifier of the table or map set to be loaded. The name must be alphanumeric and up to 8 characters long.

SET(*ptr-ref*)

specifies a pointer reference to be set to the address of the table or map set.

Exception Conditions

NOTAUTH

occurs if a resource security check has failed on PROGRAM(*name*).

Default action: Terminate the task abnormally.

PGMIDERR

RESP2 values:

- 1 The resource specified in the PROGRAM option does not have an installed resource definition.
- 2 The resource specified in the PROGRAM option is disabled.
- 3 The resource specified in the PROGRAM option refers to an OS/400 object that could not be located.
- 9 The resource specified in the PROGRAM option is defined as remote.

Default action: Terminate the task abnormally.

POP HANDLE

Restore the stack.

►►—POP HANDLE—◄◄

Condition: INVREQ

Description

POP HANDLE is not supported for C programs.

POP HANDLE restores the effect of HANDLE ABEND, HANDLE AID, HANDLE CONDITION, and IGNORE CONDITION commands to the state they were in before a PUSH HANDLE command was executed at the current link level. This can be useful, for example, during a branch to a subroutine embedded in a main program.

Normally, when a CICS program calls a subroutine, the program or routine that receives control inherits the current HANDLE commands. These commands may not be appropriate within the called program. The called program can use PUSH HANDLE to suspend existing HANDLE commands. Before returning control to the caller, the called program can restore the original commands using the POP HANDLE command.

You can nest PUSH HANDLE...POP HANDLE command sequences within a task. Each POP HANDLE command restores a set of specifications that were previously stacked by a PUSH HANDLE command at the same link level.

Refer to Chapter 6, "Dealing with exception conditions," on page 87 for more information.

Exception Conditions

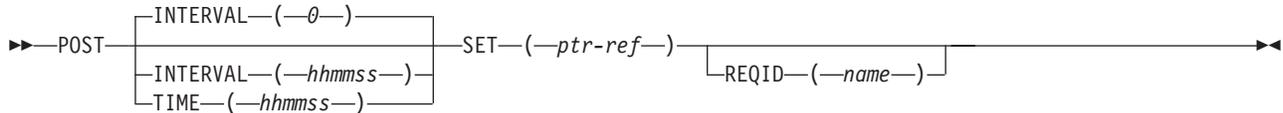
INVREQ

occurs if no matching PUSH HANDLE command has been executed at the current link level.

Default action: Terminate the task abnormally.

POST

Request notification when a specified time has expired.



Conditions: EXPIRED, INVREQ

Description

In response to this command, CICS makes a timer event control area available for testing. This 4-byte control area is initialized to binary zeros, and the pointer reference specified in the SET option is set to its address.

When the time you specify has expired, the timer event control area is posted; that is, its first byte is set to X'40' and its third byte to X'80'. You can test posting in either of the following ways:

- By checking the timer event control area at intervals. You must give CICS the opportunity to post the area; that is, the task must relinquish control to CICS before you test the area. Normally, this condition is satisfied as a result of other commands being issued; if a task is performing a long internal function, you can force control to be relinquished by issuing a SUSPEND command.
- By suspending task activity with a WAIT EVENT command until the timer event control area is posted. This action is similar to issuing a DELAY command, the difference being that with a POST and WAIT EVENT command sequence, you can do some processing after issuing the POST command, whereas a DELAY command suspends task activity at once. No other task should attempt to wait on the event set up by a POST command.

The timer event control area can be released for a variety of reasons (see below). If this happens, the result of any other task issuing a wait on the event set up by the POST command is unpredictable.

However, other tasks can cancel the event if they have access to the REQID associated with the POST command. (See the CANCEL command and the description of the REQID option.) A timer event control area provided for a task is not released or altered (except as described above) until one of the following events occurs:

- The task issues a subsequent DELAY, POST, or START command.
- The task issues a CANCEL command to cancel the POST command.
- The task is terminated, normally or abnormally.
- Any other task issues a CANCEL command for the event set up by the POST command.

A task can have only one POST command active at any given time. Any DELAY, POST, or START command supersedes a POST command previously issued by the task.

Refer to Chapter 18, “Interval control,” on page 193 for more details about Interval Control.

Options

INTERVAL(*hhmmss*)

specifies the interval of time that is to elapse from the time at which the POST command is issued until notification occurs, that is, until the timer event control area is posted. The specified interval is added to the current clock time by CICS to calculate the expiration time. See Chapter 18, “Interval control,” on page 193 for an explanation of how expiration times are used within interval control.

The maximum permitted INTERVAL value is 995959.

REQID(*name*)

specifies a name that uniquely identifies the POST command. The name can be up to 8 characters long. This name is used as a temporary storage identifier.

This option can be used when another task is to be provided with the capability of canceling an unexpired POST command.

If this option is not specified, the POST command can only be canceled from the same task, and CICS generates a unique request identifier in the EIBREQID field of the EXEC interface block.

SET(*ptr-ref*)

specifies a pointer reference to be set to the address of the 4-byte timer event control area generated by CICS. This area is initialized to binary zeros; on expiration of the specified time, the first byte is set to X'40', and the third byte to X'80'.

TIME(*hhmmss*)

specifies the time at which notification is to occur, that is, the time at which the timer event control area is to be posted. See Chapter 18, “Interval control,” on page 193 for an explanation of how expiration times are used within interval control.

Exception Conditions

EXPIRED

occurs if the time specified has already expired when the command is issued.

Default action: Ignore the condition.

INVREQ

RESP2 values:

- The POST command is not valid for processing by CICS.
- 4 Hours are out of range.
- 5 Minutes are out of range.
- 6 Seconds are out of range.
- The specified REQID is not unique within the system.

Default action: Terminate the task abnormally.

Examples

The following example shows how to request a timer event control area for a task, to be posted after 30 seconds:

```
EXEC CICS POST
      INTERVAL(30)
      REQID('RBL3D')
      SET(PREF)
      :
```

The following example shows how to provide a timer event control area for the task, to be posted when the specified time of day is reached. Because no request identifier is specified in the command, CICS automatically assigns one and returns it to the application program in the EIBREQID field in the EIB.

```
EXEC CICS POST
      TIME(PACKTIME)
      SET(PREF)
      :
```

PUSH HANDLE

Suspend the stack.

►►—PUSH HANDLE—◄◄

Description

PUSH HANDLE is not supported for ILE C programs.

PUSH HANDLE suspends the current effect of HANDLE ABEND, HANDLE AID, HANDLE CONDITION, and IGNORE CONDITION commands. This can be useful, for example, during a branch to a subroutine embedded in a main program.

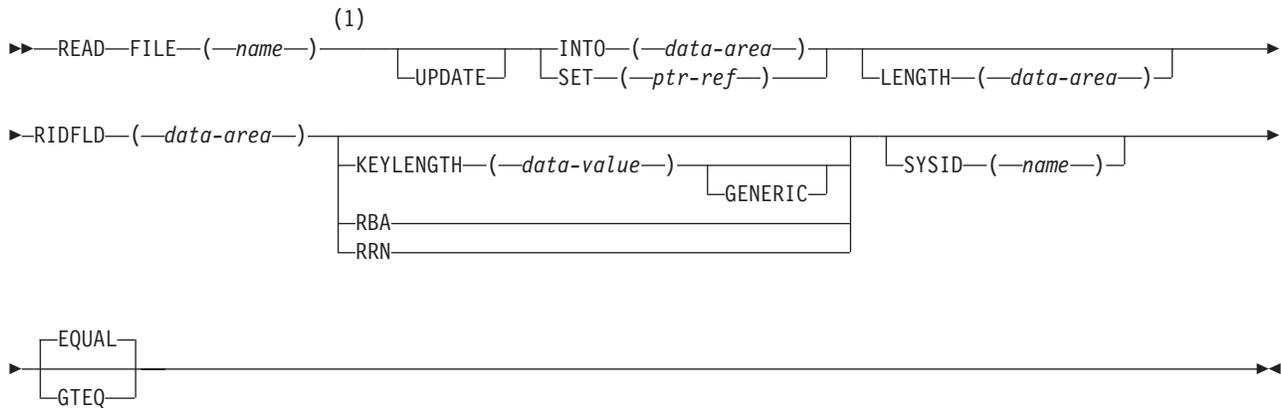
Normally, when a CICS program calls a subroutine, the program or routine that receives control inherits the current HANDLE commands. These commands may not be appropriate within the called program. The called program can use PUSH HANDLE to suspend existing HANDLE commands. Before returning control to the caller, the called program can restore the original commands using the POP HANDLE command.

You can nest PUSH HANDLE...POP HANDLE command sequences within a task. Each PUSH HANDLE command stacks a set of specifications for restoration later by POP HANDLE.

Refer to Chapter 6, “Dealing with exception conditions,” on page 87 for more information.

READ

Read a record from a file on a local or a remote system.



Notes:

1 DATASET is also accepted, but FILE is the preferred term (see “DATASET option” on page 311).

Conditions: DISABLED, DUPKEY, FILENOTFOUND, ILLOGIC, INVREQ, IOERR, ISCINVREQ, LENGERR, NOTAUTH, NOTFND, NOTOPEN, SYSIDERR

Description

For both UPDATE and nonupdate commands, you must identify the record to be retrieved by the record identification field specified in the RIDFLD option. Immediately on completion of a READ UPDATE command, the RIDFLD data area is available for reuse by the application program.

You can specify only one update operation for each file within a transaction at any given time. To avoid deadlock when accessing a file, your next command to the file must be REWRITE, DELETE without RIDFLD, or UNLOCK.

Refer to Chapter 10, “File control,” on page 115 for more information.

Options

EQUAL

specifies that the search is satisfied only by a record having the same key (complete or generic) as that specified in the RIDFLD option.

FILE(name)

specifies the name of the file to be accessed. The name must be alphanumeric, up to 8 characters long, and must have been defined in the file control table (FCT) unless the SYSID option specifies a remote system.

If a nonlocal SYSID is specified, the underlying file is assumed to be on a remote system irrespective of whether the name is defined in the local FCT. Otherwise, the FCT entry is used to determine whether the underlying file is on a local or a remote system.

GENERIC

specifies that the search key is a generic key whose length is specified in the KEYLENGTH option. This option can be used only with a KSDS or a path over a KSDS or ESDS. The search for a record is satisfied when a record is found that has the same starting characters (generic key) as those specified.

GTEQ

specifies that, if the search for a record having the same key (complete or generic) as that specified in the RIDFLD option is unsuccessful, the first record having a greater key satisfies the search. Use this option only with a KSDS or a path over a KSDS or ESDS.

INTO*(data-area)*

specifies the data area into which the record retrieved from the file is to be written.

KEYLENGTH*(data-value)*

specifies as a halfword binary value the length of the key supplied in the RIDFLD option. If a specified KEYLENGTH value differs from the length defined for the underlying file and the operation is not generic, the INVREQ condition occurs.

INVREQ also occurs if you specify GENERIC, and the KEYLENGTH value is not less than that defined for the file.

If KEYLENGTH(0) is used with the object of reading the first record in the file, the GTEQ option must also be specified. If EQUAL is specified either explicitly or by default with KEYLENGTH(0), the results of the READ are unpredictable.

LENGTH*(data-area)*

specifies as a halfword binary value the length of the data area where the retrieved record is to be placed. On completion of the READ command, the LENGTH argument (which must be a data area) contains the actual length of the record.

This option must be specified with the INTO option on READ commands involving variable-length records. It need not be specified for fixed-length records, but its inclusion is recommended because:

- It causes a check to be made that the record being read is not too long for the available data area.
- When reading fixed-length records into an area longer or shorter than the record being accessed, the LENGERR condition occurs if the LENGTH option is not specified.

When reading into a target data area longer than the record being read, the contents of the target data area, from the end of the retrieved record to the end of the target data area, are unpredictable.

If you specify the INTO option, the LENGTH argument must specify the largest record that the program accepts. If the retrieved record is longer than the value specified in the LENGTH option, the record is truncated to the specified value and the LENGERR condition occurs. In this case, the LENGTH argument is set to the length of the record before truncation.

If you specify the SET option, the LENGTH option need not be specified. If the SET option is specified, the argument must be a data area.

RBA

specifies that the record identification field specified in the RIDFLD option contains a relative byte address. This option can be used only when reading records directly from an ESDS. The RIDFLD value can be from 0 upward.

RIDFLD*(data-area)*

specifies the record identification field. The contents can be a key, a relative

byte address, or a relative record number. For a relative byte address or a relative record number, the format of this field must be fullword binary.

Make sure that the RIDFLD data area is not shorter than the KEYLENGTH value specified in this command or, if KEYLENGTH is not specified, the key length of the file you are reading; otherwise, the results are unpredictable.

RRN

specifies that the record identification field specified in the RIDFLD option contains a relative record number. Use this option only when reading records from an RRDS. The RIDFLD value can be from 1 upward.

SET(*ptr-ref*)

specifies a pointer reference to be set to the address of the retrieved record.

The pointer reference is valid until the next READ command for the same file or until completion of a corresponding REWRITE, DELETE, or UNLOCK command, or a SYNCPOINT in the case of READ UPDATE SET. If you want to retain the data within the field addressed by the pointer, you should move it to your own area.

SYSID(*name*)

specifies the name of the system to which the request is directed. The name can be up to 4 characters long.

If you specify SYSID, and omit both RBA and RRN, you must also specify LENGTH and KEYLENGTH; they cannot be found in the FCT.

UPDATE

specifies that the record is to be obtained for updating or deletion. If this option is omitted, a read-only operation is assumed.

Exception Conditions

Note: RESP2 values are not set for files that are on remote systems.

DISABLED

RESP2 values:

50 A file is disabled.

A file may be disabled because:

- It was initially defined as disabled and has not since been enabled.
- It has been disabled by an EXEC CICS SET FILE command.
- It has been disabled by the CEMT transaction.

Default action: Terminate the task abnormally.

DUPKEY

RESP2 values:

140 occurs if a record is retrieved from a VSAM emulated file that allows duplicate keys, and another record with the same key follows.

Default action: Terminate the task abnormally.

FILENOTFOUND

RESP2 values:

1 The name specified in the FILE option cannot be found in the FCT.

Default action: Terminate the task abnormally.

ILLOGIC

RESP2 values:

- 110 There is an error that does not fall within one of the other CICS response categories. (Further information is available in the EIBRCODE field; refer to Appendix A, "EXEC interface block," on page 529 for details.)

Default action: Terminate the task abnormally.

INVREQ

RESP2 values:

- 20 READ is not allowed according to the file entry specification in the FCT.
- 20 A READ command with the UPDATE option is issued to a file where update operations are not allowed according to the file entry specification in the FCT.
- 25 The KEYLENGTH and GENERIC options are specified, and the length specified in the KEYLENGTH option is greater than or equal to the length of a full key.
- 26 The KEYLENGTH option is specified (but the GENERIC option is not specified), and the specified length differs from the length defined for the underlying file.
- 28 Following a READ UPDATE command for a file, another READ UPDATE command is issued for the same file before exclusive control is released by a REWRITE, UNLOCK, or DELETE command.
- 42 The KEYLENGTH and GENERIC options are specified, and the length specified in the KEYLENGTH option is less than zero.

Default action: Terminate the task abnormally.

IOERR

RESP2 values:

- 120 There is an I/O error during the READ. An I/O error is any unusual event that is not covered by a CICS exception condition.
- (Further information is available in the EIBRCODE field; refer to Appendix A, "EXEC interface block," on page 529 for details.)

Default action: Terminate the task abnormally.

ISCINVREQ

RESP2 values:

- 70 The remote system indicates a failure that does not correspond to a known condition.

Default action: Terminate the task abnormally.

LENGERR

RESP2 values:

- 10 Neither the LENGTH option nor the SET option is specified on a READ command for a file with variable-length records.
- 11 The length of a record read with the INTO option specified exceeds the

value specified in the LENGTH option; the record is truncated, and the data area supplied in the LENGTH option is set to the actual length of the record.

13 An incorrect length is specified for a file with fixed-length records.

Default action: Terminate the task abnormally.

NOTAUTH

RESP2 values:

101 A resource security check has failed on FILE(*name*).

Default action: Terminate the task abnormally.

NOTFND

RESP2 values:

80 An attempt to retrieve a record based on the search argument provided is unsuccessful.

Default action: Terminate the task abnormally.

NOTOPEN

does not arise in CICS/400 but is mentioned here for compatibility with other versions of CICS. If the file is CLOSED and ENABLED it is opened as part of the READ command. If the file is CLOSED and DISABLED, the DISABLED condition is raised. In CICS/400 there is no file UNENABLED state as there is in other versions of CICS.

SYSIDERR

RESP2 values:

130 The SYSID option specifies either a name that is not defined in the terminal control system table (TCS), or a system to which the link is closed.

Default action: Terminate the task abnormally.

Examples

The following example shows how to read a record from a file named MASTER into a specified data area:

```
EXEC CICS READ
      INTO(RECORD)
      FILE('MASTER')
      RIDFLD(ACCTNO)
      LENGTH(RLENGTH)
      :
```

The following example shows how to read a record for update from a file, using a generic key and specifying a greater-than-or-equal key search:

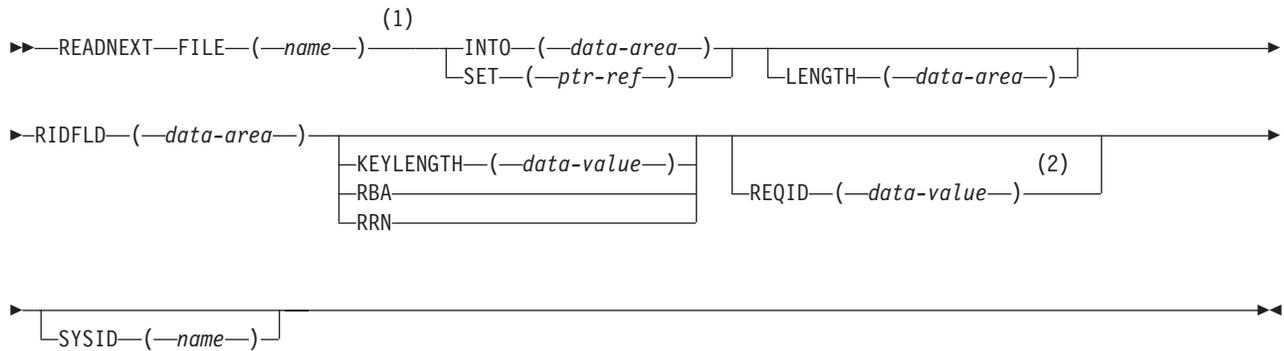
```

EXEC CICS READ
      INTO(RECORD)
      LENGTH(RLENGTH)
      FILE('MASTER')
      RIDFLD(ACCTNO)
      KEYLENGTH(4)
      GENERIC
      GTEQ
      UPDATE
      :

```

READNEXT

Read the next record during a browse.



Notes:

- 1 DATASET is also accepted, but FILE is the preferred term (see “DATASET option” on page 311).
- 2 Only supported when function shipping.

Conditions: DUPKEY, ENDFILE, FILENOTFOUND, ILLOGIC, INVREQ, IOERR, ISCINVREQ, LENGERR, NOTAUTH, NOTFND, SYSIDERR

Description

READNEXT can be used repeatedly to read records in sequential order from a file on a local or a remote system. Such a series of sequential read commands is known as a **browse** of the file. A browse can also consist of a sequence of READNEXT and READPREV commands in any order. A browse must be initiated with the STARTBR command, to identify the starting point of the browse, and terminated with the ENDBR command.

You must provide, in the RIDFLD option, a data area that is sufficiently large to contain a complete identifier (full key, RBA, or RRN) of records in the file.

The first READNEXT command reads the record at the position specified in the STARTBR command. On completion of the READNEXT command, CICS places the complete identifier of the record just retrieved into the RIDFLD data area. When the next READNEXT command is issued, CICS reads the record following the position identified by the contents of the RIDFLD field and updates the RIDFLD field with the identifier of the record just read.

You may modify the RIDFLD data area contents before issuing subsequent READNEXT commands. This has the effect of starting a new browse, which means the record identified by the RIDFLD field is read on the next READNEXT command, rather than the next record, in the same way as if a STARTBR command had been issued. CICS updates the RIDFLD contents with the complete identifier of the record just read, and the next READNEXT command causes the next record to be read, as before.

If the browse was started with the GENERIC option, the modified RIDFLD must be generic. If the browse was started with the GTEQ option, the next record returned is the first record in the file with a key greater than or equal to the modified RIDFLD.

A READNEXT command following a READPREV command has the same effect as issuing a STARTBR command or repositioning the browse. The record identified by the RIDFLD field is read; that is CICS reads the same record as that read by the READPREV command, not the next record.

See Chapter 10, "File control," on page 115 for further information.

Options

FILE(*name*)

specifies the name of the file to be browsed. The name must be alphanumeric, up to 8 characters long, and must have been defined in the file control table (FCT) unless the SYSID option specifies a remote system.

If a nonlocal SYSID is specified, the underlying file is assumed to be on a remote system irrespective of whether the name is defined in the local FCT. Otherwise, the FCT entry is used to determine whether the underlying file is on a local or a remote system.

INTO(*data-area*)

specifies the data area into which the record retrieved from the file is to be written.

KEYLENGTH(*data-value*)

specifies as a halfword binary value the length of the key supplied in the RIDFLD option.

If the browse was started without the GENERIC option (that is, a full key browse) and a specified KEYLENGTH value differs from the length defined for the underlying file, the INVREQ condition occurs.

In a generic browse, if KEYLENGTH is omitted or the KEYLENGTH value is unchanged, the browse continues using the last specified key length.

KEYLENGTH(0) may be used in a generic browse with the object of reading the first record in the file, provided that GTEQ was specified on the STARTBR or RESETBR command.

LENGTH(*data-area*)

specifies as a halfword binary value the length of the data area where the retrieved record is to be placed. On completion of the retrieval operation, the LENGTH argument (which must be a data area) contains the actual length of the retrieved record.

This option must be specified with SYSID. It must also be specified with the INTO option on commands involving variable-length records. It need not be specified for fixed-length records, but its inclusion is recommended because:

- It causes a check to be made that the record being read is not too long for the available data area.
- When browsing fixed-length records into an area longer than the record being accessed, the LENGERR condition occurs if the LENGTH option is not specified. If the length specified exceeds the file record length, CICS uses the longer length for the move. If the target area in the application program is not large enough, storage is overlaid beyond the target area.

When browsing into a target data area longer than the record being read, the contents of the target data area, from the end of the retrieved record to the end of the target data area, are unpredictable.

If you specify the INTO option, the LENGTH argument must specify the largest record that the program accepts. If the retrieved record is longer than the value specified in the LENGTH option, the record is truncated to the specified value and the LENGERR condition occurs. In this case, the LENGTH argument is set to the length of the record before truncation.

If you specify the SET option, the LENGTH option need not be specified but, if it is, the argument must be a data area.

RBA

specifies that the record identification field specified in the RIDFLD option contains a relative byte address. The RIDFLD value can be from 0 upward.

This option must be specified when browsing an ESDS directly.

REQID(*data-value*)

specifies as a halfword binary value a unique request identifier for the browse; it is used to control multiple browse operations on a file. If this option is not specified, a default value of zero is assumed.

RIDFLD(*data-area*)

specifies the record identification field. The contents can be a key, a relative byte address, or a relative record number. For a relative byte address or a relative record number, the format of this field must be fullword binary.

Even if the browse is generic, this field should always be large enough to accommodate the complete record identifier. This is because, on completion of the READNEXT command, the field is updated by CICS with the complete identification of the record retrieved.

RRN

specifies that the record identification field specified in the RIDFLD option contains a relative record number. Use this option only when browsing an RRDS. The RIDFLD value can be from 1 upward.

SET(*ptr-ref*)

specifies a pointer reference to be set to the address of the retrieved record.

The pointer reference is valid until the next READNEXT or READPREV command specifying SET for the same file. The pointer is no longer valid after an ENDBR or SYNCPOINT command. If you want to retain the data within the field addressed by the pointer, you should move it to your own area.

SYSID(*name*)

specifies the name of the system to which the request is directed. The name can be up to 4 characters long.

If you specify SYSID, and omit both RBA and RRN, you must also specify LENGTH and KEYLENGTH; they cannot be found in the FCT.

Exception Conditions

Note: RESP2 values are not set for files that are on remote systems.

DUPKEY

RESP2 values:

- 140** A record is retrieved from a VSAM emulated file that allows duplicate keys, and another record with the same key follows. It does not occur as a result of a READNEXT command that reads the last of the records having the nonunique key.

Default action: Terminate the task abnormally.

ENDFILE

RESP2 values:

- 90** An end-of-file condition is detected during the browse.

Default action: Terminate the task abnormally.

FILENOTFOUND

RESP2 values:

- 1** The name specified in the FILE option cannot be found in the FCT.

Default action: Terminate the task abnormally.

ILLOGIC

RESP2 values:

- 110** There is an error that does not fall within one of the other CICS response categories. (Further information is available in the EIBRCODE field; refer to Appendix A, "EXEC interface block," on page 529 for details.)

Default action: Terminate the task abnormally.

INVREQ

RESP2 values:

- 25** The KEYLENGTH option is specified for a generic browse (that is, one where GENERIC was specified on the STARTBR command or the last RESETBR command) and the value of KEYLENGTH was greater than or equal to the full key length.
- 26** The KEYLENGTH option is specified for a non-generic browse, and the specified length differs from the length defined for the underlying file.
- 34** The READNEXT command is issued for a file that has had no previous STARTBR command successfully issued.
- 37** The type of record identification (for example, key or relative byte address) used to access a file during the browse is changed by the READNEXT command.
- 42** The KEYLENGTH option is specified for a generic browse (that is, one where GENERIC was specified on the STARTBR command or the last RESETBR command) and the value of KEYLENGTH was less than zero.

Default action: Terminate the task abnormally.

IOERR

RESP2 values:

- 120** There is an I/O error during the READNEXT. An I/O error is any unusual event that is not covered by a CICS exception condition.
(Further information is available in the EIBRCODE field; refer to Appendix A, "EXEC interface block," on page 529 for details.)

Default action: Terminate the task abnormally.

ISCINVREQ

RESP2 values:

- 70** The remote system indicates a failure that does not correspond to a known condition.

Default action: Terminate the task abnormally.

LENGERR

RESP2 values:

- 10** Neither the LENGTH nor the SET option is specified for a file with variable-length records.
- 11** The length of the record read with the INTO option specified exceeds the value specified in the LENGTH option; the record is truncated, and the data area supplied in the LENGTH option is set to the actual length of the record.
- 13** An incorrect length is specified for a file with fixed-length records.

Default action: Terminate the task abnormally.

NOTAUTH

RESP2 values:

- 101** A resource security check has failed on FILE(*name*).

Default action: Terminate the task abnormally.

NOTFND

RESP2 values:

- 80** An attempt to retrieve a record based on the search argument provided is unsuccessful. This may occur if the READNEXT command immediately follows a STARTBR command that specified the key of the last record in the file (a complete key of X'FF' bytes).

Default action: Terminate the task abnormally.

SYSIDERR

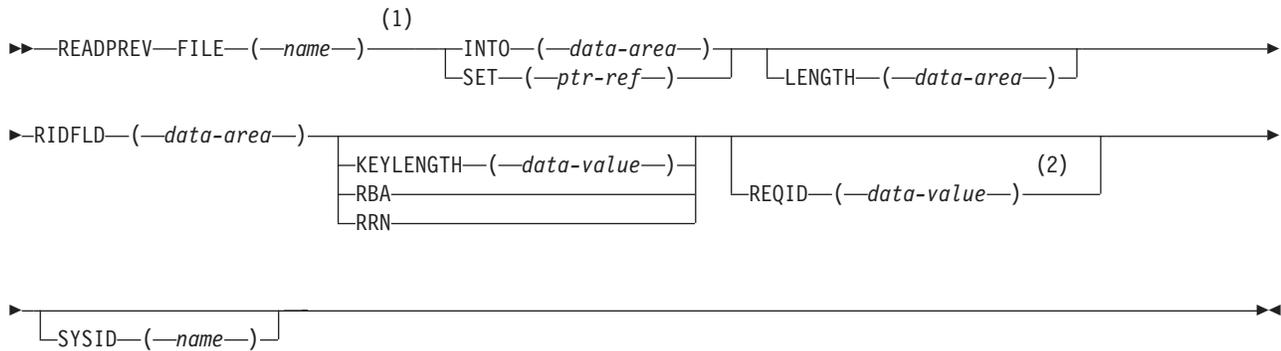
RESP2 values:

- 130** The SYSID option specifies either a name that is not defined in the terminal control system table (TCS), or a system to which the link is closed.

Default action: Terminate the task abnormally.

READPREV

Read the previous record during a browse.



Notes:

- 1 DATASET is also accepted, but FILE is the preferred term (see “DATASET option” on page 311).
- 2 Only supported when function shipping.

Conditions: DUPKEY, ENDFILE, FILENOTFOUND, ILLOGIC, INVREQ, IOERR, ISCINVREQ, LENGERR, NOTAUTH, NOTFND, SYSIDERR

Description

READPREV can be used repeatedly to read records in reverse sequential order from a file on a local or a remote system.

Such a series of sequential read commands is known as a **browse** of the file. A browse can also consist of a sequence of READNEXT and READPREV commands in any order. A browse must be initiated with the STARTBR command, to identify the starting point of the browse, and terminated with the ENDBR command.

You must provide, in the RIDFLD option, a data area that is sufficiently large to contain a complete identifier (full key, RBA, or RRN) of records in the file.

The first READPREV command reads the record at the position specified in the STARTBR command. On completion of the READPREV command, CICS places the complete identifier of the record just retrieved into the RIDFLD data area. When the next READPREV command is issued, CICS reads the record following the position identified by the contents of the RIDFLD field and updates the RIDFLD field with the identifier of the record just read.

You may modify the RIDFLD data area contents before issuing subsequent READPREV commands. This has the effect of starting a new browse, which means the record identified by the RIDFLD field is read on the next READPREV command, rather than the next record, in the same way as if a STARTBR command had been issued. The modified record identifier must always be a full key, RBA, or RRN. A generic key may not be specified, nor may a browse that was started with the GENERIC option include a READPREV command.

If you include a READPREV command immediately following a STARTBR command, your STARTBR command must specify the key of a record that exists on the file; otherwise, the NOTFND condition occurs for the READPREV command.

A READPREV command following a READNEXT command has the same effect as issuing a STARTBR command or repositioning the browse. The record identified by the RIDFLD field is read; that is CICS reads the same record as that read by the READNEXT command, not the previous record.

See Chapter 10, "File control," on page 115 for further information.

Options

FILE(*name*)

specifies the name of the file being browsed. The name must be alphanumeric, up to 8 characters long, and must have been defined in the file control table (FCT) unless the SYSID option specifies a remote system.

If a nonlocal SYSID is specified, the underlying file is assumed to be on a remote system irrespective of whether the name is defined in the local FCT. Otherwise, the FCT entry is used to determine whether the underlying file is on a local or a remote system.

INTO(*data-area*)

specifies the data area into which the record retrieved from the file is to be written.

KEYLENGTH(*data-value*)

specifies as a halfword binary value the length of the key supplied in the RIDFLD option.

If a specified KEYLENGTH value differs from the length defined for the underlying file, the INVREQ condition occurs.

LENGTH(*data-area*)

specifies as a halfword binary value the length of the data area where the retrieved record is to be placed. On completion of the retrieval operation, the LENGTH argument (which must be a data area) contains the actual length of the retrieved record.

This option must be specified with SYSID. It must also be specified with the INTO option on READPREV commands involving variable-length records. It need not be specified for fixed-length records, but its inclusion is recommended because:

- It causes a check to be made that the record being read is not too long for the available data area.
- When browsing fixed-length records into an area longer than the record being accessed, the LENGERR condition occurs if the LENGTH option is not specified. If the length specified exceeds the file record length, CICS uses the longer length for the move. If the target area in the application program is not large enough, storage is overlaid beyond the target area.

When browsing into a target data area longer than the record being read, the contents of the target data area, from the end of the retrieved record to the end of the target data area, are unpredictable.

If you specify the INTO option, the LENGTH argument must specify the largest record that the program accepts. If the retrieved record is longer than

the value specified in the LENGTH option, the record is truncated to the specified value and the LENGERR condition occurs. In this case, the LENGTH argument is set to the length of the record before truncation.

If you specify the SET option, the LENGTH option need not be specified but, if it is, the argument must be a data area.

RBA

specifies that the record identification field specified in the RIDFLD option contains a relative byte address. The RIDFLD value can be from 0 upward. You must specify this option when browsing an ESDS directly, but not at any other time.

REQID(*data-value*)

specifies as a halfword binary value a unique request identifier for the browse; it is used to control multiple browse operations on a file. If this option is not specified, a default value of zero is assumed.

RIDFLD(*data-area*)

specifies the record identification field. The contents can be a key, a relative byte address, or a relative record number. For a relative byte address or a relative record number, the format of this field must be fullword binary.

On completion of the READPREV command, this field is updated by CICS with the complete identification of the record retrieved.

RRN

specifies that the record identification field specified in the RIDFLD option contains a relative record number. Use this option only when browsing an RRDS. The RIDFLD value can be from 1 upward.

SET(*ptr-ref*)

specifies a pointer reference to be set to the address of the retrieved record.

The pointer reference is valid until the next READNEXT or READPREV command specifying the SET option for the same file. The pointer is no longer valid after an ENDBR or SYNCPOINT command. If you want to retain the data within the field addressed by the pointer, you should move it to your own area.

SYSID(*name*)

specifies the name of the system to which the request is directed. The name can be up to 4 characters long.

If you specify SYSID, and omit both RBA and RRN, you must also specify LENGTH and KEYLENGTH; they cannot be found in the FCT.

Exception Conditions

Note: RESP2 values are not set for files that are on remote systems.

DUPKEY

RESP2 values:

140 A record is retrieved from a VSAM emulated file that allows duplicate keys, and another record with the same key exists.

Default action: Terminate the task abnormally.

ENDFILE

RESP2 values:

90 An end-of-file condition is detected during a browse.

Default action: Terminate the task abnormally.

FILENOTFOUND

RESP2 values:

1 The name specified in the FILE option cannot be found in the FCT.

Default action: Terminate the task abnormally.

ILLOGIC

RESP2 values:

110 There is an error that does not fall within one of the other CICS response categories. (Further information is available in the EIBRCODE field; refer to Appendix A, "EXEC interface block," on page 529 for details.)

Default action: Terminate the task abnormally.

INVREQ

RESP2 values:

24 A READPREV command is issued for a file for which the previous STARTBR command has the GENERIC option.

26 The KEYLENGTH option is specified, and the specified length differs from the length defined for the underlying file.

37 The type of record identification (for example, key or relative byte address) used to access a file during the browse is changed by the READPREV command.

41 A READPREV command is issued for a file for which no previous STARTBR command has been successfully issued.

Default action: Terminate the task abnormally.

IOERR

RESP2 values:

120 There is an I/O error during the browse. An I/O error is any unusual event that is not covered by a CICS exception condition.

(Further information is available in the EIBRCODE field; refer to Appendix A, "EXEC interface block," on page 529 for details.)

Default action: Terminate the task abnormally.

ISCINVREQ

RESP2 values:

70 The remote system indicates a failure that does not correspond to a known condition.

Default action: Terminate the task abnormally.

LENGERR

RESP2 values:

10 Neither the LENGTH nor the SET option is specified for a file with variable-length records.

- 11 The length of the record read with the INTO option specified exceeds the value specified in the LENGTH option; the record is truncated, and the data area supplied in the LENGTH option is set to the actual length of the record.
- 13 An incorrect length is specified for a file with fixed-length records.

Default action: Terminate the task abnormally.

NOTAUTH

RESP2 values:

- 101 A resource security check has failed on FILE(*name*).

Default action: Terminate the task abnormally.

NOTFND

RESP2 values:

- 80 An attempt to retrieve a record based on the search argument provided is unsuccessful. This may occur if the READPREV command immediately follows a STARTBR command that specifies the key of a record that does not exist on the file.

Default action: Terminate the task abnormally.

SYSIDERR

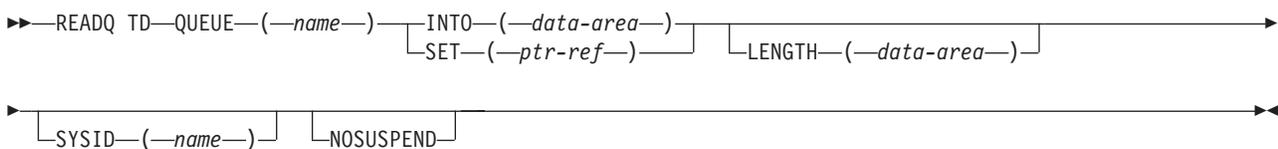
RESP2 values:

- 130 The SYSID option specifies either a name that is not defined in the terminal control system table (TCS), or a system to which the link is closed.

Default action: Terminate the task abnormally.

READQ TD

Read transient data from a queue.



Conditions: DISABLED, INVREQ, IOERR, ISCINVREQ, LENGERR, NOTAUTH, NOTOPEN, QBUSY, QIDERR, QZERO, SYSIDERR

Description

The READQ TD command reads transient data from a queue, after which the data is no longer available.

If you are using automatic transaction initiation (ATI) (see “Automatic transaction initiation (ATI)” on page 216 for introductory information), the HANDLE CONDITION QZERO command should be included in your application to ensure that termination of an automatically initiated task occurs only when the queue is empty.

See Chapter 23, “Transient data control,” on page 215 for more information.

Options

INTO(*data-area*)

specifies the user data area into which the data read from the transient data queue is to be placed.

LENGTH(*data-area*)

specifies as a halfword binary value the length of the data. The upper limit is determined by the maximum record length of the underlying TS/TD physical file. For a description of a safe upper limit, see “LENGTH options” on page 312..

If you specify the INTO option, the LENGTH argument must specify the maximum length of data that the program accepts. If the value specified is less than zero, zero is assumed. If the length of data exceeds the value specified, the data is truncated to that value and the LENGERR condition occurs. On completion of the retrieval operation, the data area is set to the original length of the data.

If you specify the SET option, the LENGTH option need not be specified. On completion of the retrieval operation, any specified LENGTH argument is set to the length of the data.

NOSUSPEND

specifies that application program suspension for the QBUSY condition is to be inhibited.

This applies only to intrapartition queues.

QUEUE(*name*)

specifies the symbolic name of the queue to be read from. The name must be alphanumeric, up to 4 characters long, and must have been defined in the destination control table (DCT) unless the SYSID option specifies a remote system.

If a nonlocal SYSID is specified, the queue is assumed to be on a remote system irrespective of whether the name is defined in the local DCT. Otherwise, the DCT entry is used to determine whether the queue is on a local or a remote system.

SET(*ptr-ref*)

specifies a pointer reference to be set to the address of the data read from the queue. CICS acquires an area large enough to hold the record and sets the pointer reference to the address of that area. The area is retained until another transient data command is executed. The pointer reference, unless changed by other commands or statements, is valid until the next READQ TD command or the end of task.

SYSID(*name*)

specifies the name of the system to which the request is directed. The name can be up to 4 characters long.

Exception Conditions

DISABLED

occurs if the queue has been disabled.

Default action: Terminate the task abnormally.

INVREQ

occurs if READQ names an extrapartition queue that has been opened for output.

This condition cannot occur for intrapartition queues.

Default action: Terminate the task abnormally.

IOERR

occurs if there is an I/O error during the transient data operation.

This condition occurs only if the queue can be read; the QZERO condition occurs if the queue cannot be read.

Default action: Terminate the task abnormally.

ISCINVREQ

occurs if the remote system indicates a failure that does not correspond to a known condition.

Default action: Terminate the task abnormally.

LENGERR

occurs if READQ TD names an INTO area that cannot hold all the data to be returned to the application.

Default action: Terminate the task abnormally.

NOTAUTH

occurs if a resource security check has failed on QUEUE(*name*).

Default action: Terminate the task abnormally.

NOTOPEN

occurs if the destination is closed.

This condition applies to extrapartition queues only.

Default action: Terminate the task abnormally.

QBUSY

occurs if a READQ TD command attempts to access a record in a recoverable intrapartition queue that is being written to, or deleted by, another task; and there are no more committed records.

Default action: The task issuing the READQ TD command waits until the queue is no longer being used for output. However, the NOSUSPEND option (see above) overrides this default action.

QIDERR

occurs if the symbolic destination to be used with READQ TD cannot be found.

Default action: Terminate the task abnormally.

QZERO

occurs if the destination (queue) is empty or the end of the queue has been reached.

Default action: Terminate the task abnormally.

SYSIDERR

occurs if the SYSID option specifies either a name that is not defined in the terminal control system table (TCS), or a system to which the link is closed.

Default action: Terminate the task abnormally.

LENGTH(*data-area*)

specifies as a halfword binary value the length of the data. For a description of a safe upper limit, see “LENGTH options” on page 312.

If you specify the INTO option, the LENGTH argument must specify the maximum length of data that the program accepts. If the value specified is less than zero, zero is assumed. If the length of data exceeds the value specified, the data is truncated to that value and the LENGERR condition occurs. On completion of the retrieval operation, the data area is set to the original length of the data.

If you specify the SET option, the LENGTH option need not be specified. On completion of the retrieval operation, any specified LENGTH argument is set to the length of the data.

NEXT

specifies that the next sequential logical record following the last record to be retrieved (by any task) is to be retrieved, or the first record, if no previous record has been retrieved.

NUMITEMS(*data-area*)

specifies a halfword binary field to receive a number indicating how many items there are in the queue.

QUEUE(*name*)

specifies the symbolic name (1–8 characters) of the queue to be read from. If the queue name appears in the temporary storage table (TST), and the entry is marked as remote, the request is shipped to a remote system.

The name must be alphanumeric and unique within the CICS system. Do not use X'FA' through X'FF' as the first character of the name; these characters are reserved for CICS use. The name cannot consist solely of binary zeros.

SET(*ptr-ref*)

specifies a pointer reference to be set to the address of the retrieved data. CICS acquires an area large enough to hold the record and sets the pointer reference to the address of the record. The area is retained until another READQ TS command, on any TS queue, is executed. The pointer reference, unless changed by other commands or statements, is valid until the next READQ TS command or the end of task.

SYSID(*name*)

specifies the name of the system to which the request is directed. The name can be up to 4 characters long.

Exception Conditions

INVREQ

occurs if the queue was created by CICS internal code.

Default action: Terminate the task abnormally.

IOERR

occurs if there is an unrecoverable input/output error.

Default action: Terminate the task abnormally.

ISCINVREQ

occurs if the remote system indicates a failure that does not correspond to a known condition.

Default action: Terminate the task abnormally.

ITEMERR

occurs in any of the following situations:

- The specified item number is not valid (that is, it is outside the range of item numbers written to the queue).
- An attempt is made to read beyond the end of the queue using the (default) NEXT option.

Default action: Terminate the task abnormally.

LENGERR

occurs if the length of the stored data is greater than the value specified by the LENGTH option. This condition can only occur when the INTO option is specified.

Default action: Terminate the task abnormally.

NOTAUTH

occurs if a resource security check has failed on QUEUE(*name*).

Default action: Terminate the task abnormally.

QIDERR

occurs if the queue specified cannot be found.

Default action: Terminate the task abnormally.

SYSIDERR

occurs if the SYSID option specifies either a name that is not defined in the terminal control system table (TCS), or a system to which the link is closed.

Default action: Terminate the task abnormally.

Examples

The following example shows how to read the first (or only) record from a temporary storage queue into a data area specified in the request:

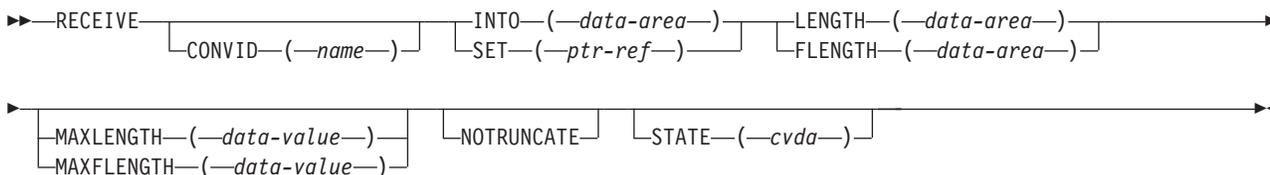
```
EXEC CICS READQ TS
      QUEUE(UNIQNAME)
      INTO(DATA)
      LENGTH(LDATA)
      ITEM(1)
      :
```

The following example shows how to read the next record from a temporary storage queue into a data area provided by CICS; the pointer reference specified by the SET option is set to the address of the storage area reserved for the data record:

```
EXEC CICS READQ TS
      QUEUE(DESCRQ)
      SET(PREF)
      LENGTH(LENG)
      NEXT
      :
```

RECEIVE (APPC)

Receive data from the conversation partner on an APPC conversation.



Conditions: EOC, INVREQ, LENGERR, NOTALLOC, SIGNAL, TERMERR

Description

The RECEIVE (APPC) command receives data from the conversation partner on an APPC conversation.

See part 3 of *CICS/400 Intercommunication* for more information.

Options

CONVID(*name*)

identifies the conversation to which the command relates. The 4-character name identifies the token returned by a previously executed ALLOCATE command in the EIBRSRCE field of the EIB.

If this option is omitted, the principal facility for the task is used by default.

FLENGTH(*data-area*)

is a fullword alternative to LENGTH(*data-area*).

INTO(*data-area*)

specifies the application target data area into which data is to be received from the application program connected to the other end of the current conversation. The length of this area must be greater than or equal to the maximum receive length specified in the LENGTH, FLENGTH, MAXLENGTH, or MAXFLENGTH options.

LENGTH(*data-area*)

specifies as a halfword binary value the length of the data to be received.

If you specify the INTO option, but omit the MAXLENGTH option, the LENGTH argument must be a data area that specifies the maximum length that the program accepts. If the value specified is less than zero, zero is assumed.

If you specify the SET option, the LENGTH argument must be a data area.

When the data has been received, the data area is set to the length of the data.

MAXFLENGTH(*data-value*)

is a fullword alternative to MAXLENGTH(*data-value*).

MAXLENGTH(*data-value*)

specifies as a halfword binary value the maximum amount of data that CICS is to recover in response to a RECEIVE command. If INTO is specified, MAXLENGTH overrides the use of LENGTH as an input to CICS. If SET is specified, MAXLENGTH provides a way for the program to limit the amount of data that it receives at one time.

If the length of data exceeds the value specified and the NOTRUNCATE option is not present, the data is truncated to that value and the LENGERR condition occurs. The data area specified in the LENGTH option is set to the original length of data (before any truncation).

If the length of data exceeds the value specified and the NOTRUNCATE option is present, CICS retains the remaining data and uses it to satisfy subsequent RECEIVE commands. The EIBCOMPL field indicates whether all the data was used or not. The data area specified in the LENGTH option is set to the length of data returned.

If MAXLENGTH is omitted, CICS uses the value specified in the LENGTH option as the maximum length that the program accepts.

NOTRUNCATE

specifies that, when the data available exceeds the length requested, the remaining data is not to be discarded but is to be retained for retrieval by subsequent RECEIVE commands.

SET(ptr-ref)

specifies a pointer reference to be set to the address of the data received from the partner transaction.

STATE(cvda)

gets the state of the current conversation. For a complete list of the CVDA values that can be returned on APPC commands and for information about receiving and testing these values, see “CICS-value data areas (CVDA)” on page 309.

Exception Conditions

EOC

occurs when no other condition is raised. Field EIBEOC also indicates this condition.

Default action: Ignore the condition.

INVREQ

RESP2 values:

- The CONVID value was obtained by an ASSIGN FACILITY command. However, the principal facility is *not* an APPC conversation.

200 The RECEIVE (APPC) command is issued in a DPL server program and refers to the principal facility.

Default action: Terminate the task abnormally.

LENGERR

RESP2 values:

- Received data is discarded by CICS because its length exceeds the maximum that the program accepts (see the LENGTH and MAXLENGTH options), and the NOTRUNCATE option is not specified.
- An out-of-range value is supplied in the LENGTH, FLENGTH, MAXLENGTH, or MAXFLENGTH option.

Default action: Terminate the task abnormally.

NOTALLOC

occurs if the specified CONVID value does not relate to a conversation owned by the application.

Default action: Terminate the task abnormally.

SIGNAL

occurs if an inbound SIGNAL data-flow control command is received from a partner transaction. EIBSIG is always set when an inbound signal is received.

Default action: Ignore the condition.

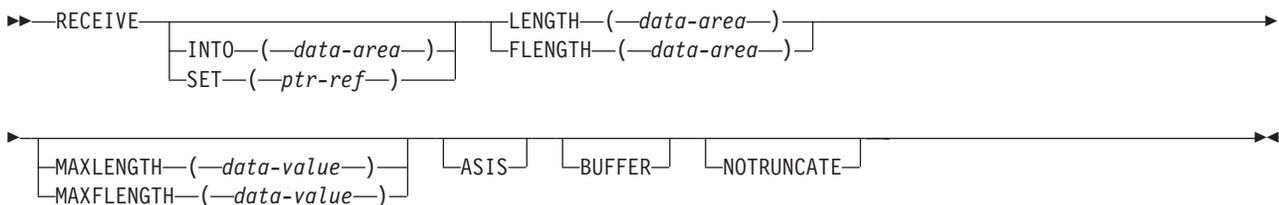
TERMERR

occurs if there is a session-related error. Any action on that conversation other than a FREE command causes an ATCV abend.

Default action: Terminate the task abnormally (with abend code ATNI).

RECEIVE (5250 or 3270 logical)

Receive data from a terminal display.



Conditions: EOC, INVREQ, LENGERR, TERMERR

Description

The RECEIVE (5250 or 3270 logical) command receives data from a terminal display.

If data is to be received, you must specify a length option and either the INTO or the SET option. If a RECEIVE is issued purely to detect an attention identifier (AID), you can omit the INTO and SET options. See "INTO and SET options" on page 311.

For a transaction started by automatic transaction initiation, a SEND command must precede the first RECEIVE command in the transaction.

Application programs should be written to handle data streams that contain both

- data that is preceded by a Set Buffer Address (SBA) order and buffer address, and
- data this is not preceded by an SBA order and buffer address.

Data that begins after row 1, column 1 requires an SBA; data that begins in row 1, column 1 does not require an SBA (since row 1, column 1 is implied). This note does not apply to RECEIVE commands issued with the BUFFER option because the data stream obtained in this instance does not contain SBA orders.

See Chapter 14, "Terminal control," on page 169 for more information about the RECEIVE command.

Options

ASIS

specifies that lowercase characters in the 5250 or 3270 input data stream are not translated to uppercase; this allows the current task to receive a message containing both uppercase and lowercase data.

This option has no effect on the first RECEIVE command of a transaction, because terminal control performs a read initial and uses the terminal defaults to translate the data. It also has no effect if the screen contains data before the transaction is initiated. This data is read and translated in preparation for the next task, and the first RECEIVE command in that task retrieves the translated data.

BUFFER

specifies that the contents of the 5250 or 3270 logical unit buffer are to be read, beginning at buffer location one and continuing until all contents of the buffer have been read. All character and attribute sequences (including nulls) appear in the input data stream in the order in which they appear in the 5250 or 3270 buffer.

Note: On 5250 devices the received data is presented to the application in '3270 Read Buffer Extended Field Mode' format. In addition, if the cursor has been moved by the user since the last SEND command or RECEIVE command without the BUFFER option, EIBCPOSN will reflect the cursor position at the time of the command rather than the position that the user has now moved it to. On 3270 devices the data is passed to the application without modification and the EIBCPOSN will be set to the cursor position as returned by the Read Buffer command.

FLNGTH(*data-area*)

is a fullword alternative to LENGTH(*data-area*).

INTO(*data-area*)

specifies the receiving field for the data read from the logical unit.

LENGTH(*data-area*)

specifies as a halfword binary value the length of the data to be received.

If you specify the INTO option, but omit the MAXLENGTH option, the LENGTH argument must be a data area that specifies the maximum length that the program accepts. If the value specified is less than zero, zero is assumed. If the length of data exceeds the value specified and the NOTRUNCATE option is not present, the data is truncated to that value and the LENGERR condition occurs. When the data has been received, the data area is set to the original length of data (before any truncation).

If you specify the SET option, the argument must be a data area. When the data has been received, the data area is set to the length of the data.

For a description of a safe upper limit, see "LENGTH options" on page 312.

MAXFLNGTH(*data-value*)

is a fullword alternative to MAXLENGTH(*data-value*).

MAXLENGTH(*data-value*)

specifies as a halfword binary value the maximum amount of data that CICS is to recover in response to a RECEIVE command. If INTO is specified, MAXLENGTH overrides the use of LENGTH as an input to CICS. If SET is specified, MAXLENGTH provides a way for the program to limit the amount of data that it receives at one time.

If the length of data exceeds the value specified and the NOTRUNCATE option is not present, the data is truncated to that value and the LENGERR condition occurs. The data area specified in the LENGTH option is set to the original length of data (before any truncation).

If the length of data exceeds the value specified and the NOTRUNCATE option is present, CICS retains the remaining data and uses it to satisfy subsequent RECEIVE commands. The data area specified in the LENGTH option is set to the length of data returned.

If MAXLENGTH is omitted, CICS uses the value specified in the LENGTH option as the maximum length that the program accepts.

NOTRUNCATE

specifies that, when the data available exceeds the length requested, the remaining data is not to be discarded, but is to be retained for retrieval by subsequent RECEIVE commands.

SET(*ptr-ref*)

specifies a pointer reference to be set to the address of the data read from the terminal or logical unit. The pointer reference, unless changed by other commands or statements, is valid until the next terminal I/O command or the end of task.

Exception Conditions

EOC

occurs when no other condition is raised. The EIBEOC field also contains this indicator.

Default action: Ignore the condition.

INVREQ

RESP2 values:

200 The command is issued in a DPL server program.

Default action: Terminate the task abnormally.

LENGERR

occurs in any of the following situations:

- Received data is discarded by CICS because its length exceeds the maximum that the program accepts (see the LENGTH and MAXLENGTH options), and the NOTRUNCATE option is not specified.
- An out-of-range value is supplied in the LENGTH, FLENGTH, MAXLENGTH, or MAXFLENGTH option.

Default action: Terminate the task abnormally.

TERMERR

occurs if there is a terminal-related error.

Default action: Terminate the task abnormally (with abend code ATNI).

name is coded as a literal, and INTO is not specified, the name of the data area defaults to the name of the map suffixed with an "I". The data mapped into the specified area includes the 12-byte prefix generated by TIOAPFX.

LENGTH(*data-value*)

specifies as a halfword binary value the length of the data to be formatted. It must not exceed the length of the FROM data area. This should include the length of the 12-byte prefix generated by the TIOAPFX=YES operand of the DFHMSD BMS map definition macro (see page 560).

For a description of a safe upper limit, see "LENGTH options" on page 312.

MAP(*name*)

specifies the name of the map to be used. The name can be up to 7 characters long.

MAPSET(*name*)

specifies either an unsuffixed or a suffixed name of the map set. An unsuffixed map set name can be up to 7 characters long, and a suffixed map set name can be from 2 through 8 characters long. The map set must reside in the library defined in the PPT. If *LIBL is used for the library of the map set, the map set must reside in the library list of the control region where the map is being used. If the MAPSET option is not specified, the name given in the MAP option is assumed to be that of the map set.

SET(*ptr-ref*)

specifies a pointer reference to be set to the address of the 12-byte prefix to the mapped data.

TERMINAL

specifies that input data is to be read from the terminal that originated the transaction.

Exception Conditions

Some of the following exception conditions may occur in combination with others. If more than one occurs, only the first is passed to the application program.

EIBRCODE, however, is set to indicate all the conditions that occurred.

INVMPSZ

occurs if the specified map is too wide or too long for the terminal.

Default action: Terminate the task abnormally.

INVREQ

RESP2 values:

- A RECEIVE MAP command is issued in a nonterminal task; these tasks do not have a TIOA or a TCTTE.

200 The command is issued in a DPL server program.

Default action: Terminate the task abnormally.

MAPFAIL

occurs if the data to be mapped has a length of zero or does not contain a set-buffer-address (SBA) sequence. The receiving data area contains the unmapped input data stream. The amount of unmapped data moved to the user's area is limited to the length specified in the LENGTH option of the RECEIVE MAP command. The input map is not set to nulls.

This condition occurs if a program issues a RECEIVE MAP command to which the terminal operator responds by pressing a CLEAR or PA key, or by pressing ENTER or a PF key without entering data.

Default action: Terminate the task abnormally.

NOTAUTH

RESP2 values:

70 A resource security check has failed on MAPSET(*name*).

Default action: Terminate the task abnormally.

RELEASE

Release a loaded OS/400 *USRSPC object defined to CICS as a program, table, or mapset.

▶▶—RELEASE—PROGRAM—(—*name*—)—————▶▶

Conditions: INVREQ, NOTAUTH, PGMIDERR

Description

RELEASE releases an object defined as *USRSPC, previously loaded by a LOAD command, so long as the use count is zero. The object could be a table or a mapset. RELEASE decrements the use count by 1.

If the HOLD option is specified in the LOAD command, the loaded resource is not released at the end of the task. It can only be released by a RELEASE command. This RELEASE command may be issued by the task that loaded the resource or by any other task.

If the HOLD option is not specified in the LOAD command, the loaded resource is released at the end of the task. It may, however, be released before this by the task that loaded the resource issuing a RELEASE command.

Tasks may not issue a RELEASE request against objects loaded, without the HOLD option, by other tasks.

Refer to Chapter 20, “Program control,” on page 199 for more information about this command.

Options

PROGRAM(*name*)

specifies the identifier (1–8 characters) of a program, table, or mapset to be released. The specified name must have been defined as a program to CICS.

See

Exception Conditions

INVREQ

RESP2 values:

Description

The RESETBR command specifies, during a browse, the record in a file, where you want the browse to be repositioned. The file may be on a local or a remote system.

When browsing a file, you can use this command not only to reposition the browse (which can be achieved more simply by modifying the RIDFLD data area on a READNEXT or READPREV command), but also to change its characteristics from those specified on the previous STARTBR command, without ending the browse. The characteristics that may be changed are those specified by the GENERIC and GTEQ options. The RBA and RRN characteristics must not be changed by the RESETBR command.

Options

EQUAL

specifies that the search is satisfied only by a record having the same key (complete or generic) as that specified in the RIDFLD option.

FILE(*name*)

specifies the name of the file to be accessed. The name must be alphanumeric, up to 8 characters long, and must have been defined in the file control table (FCT) unless the SYSID option specifies a remote system.

If a nonlocal SYSID is specified, the underlying file is assumed to be on a remote system irrespective of whether the name is defined in the local FCT. Otherwise, the FCT entry is used to determine whether the underlying file is on a local or a remote system.

GENERIC

specifies that the search key is a generic key whose length is specified in the KEYLENGTH option. Use this option only with a KSDS or a path over a KSDS or ESDS. The search for a record is satisfied when a record is found that has the same starting characters (generic key) as those specified.

GTEQ

specifies that if the search for a record having the same key (complete or generic) as that specified in the RIDFLD option is unsuccessful, the first record having a greater key satisfies the search. Use this option only with a KSDS or a path over a KSDS or ESDS.

KEYLENGTH(*data-value*)

specifies as a halfword binary value the length of the key supplied in the RIDFLD option. If a specified KEYLENGTH value differs from the length defined for the underlying file and the operation is not generic, the INVREQ condition occurs.

The INVREQ condition also occurs if you specify GENERIC, and the KEYLENGTH value is not less than that defined for the file.

If KEYLENGTH(0) is used with the object of positioning to the first record in the file, the GTEQ option must also be specified; otherwise, the NOTFND condition may occur.

Note that GTEQ is the default for RESETBR.

RBA

specifies that the record identification field specified in the RIDFLD option contains a relative byte address. Use this option only when browsing an ESDS directly. The RIDFLD value can be from 0 upward.

REQID(*data-value*)

specifies as a halfword binary value a unique request identifier for the browse; it is used to control multiple browse operations on a file. If this option is not specified, a default value of zero is assumed.

RIDFLD(*data-area*)

specifies the record identification field. The contents can be a key, a relative byte address, or a relative record number. For a relative byte address or a relative record number, the format of this field must be fullword binary.

RRN

specifies that the record identification field specified in the RIDFLD option contains a relative record number. Use this option only when browsing an RRDS. The RIDFLD value can be from 1 upward.

SYSID(*name*)

specifies the name of the system to which the request is directed. The name can be up to 4 characters long.

If you specify SYSID, and omit both RBA and RRN, you must also specify KEYLENGTH; it cannot be found in the FCT.

Exception Conditions

Note: RESP2 values are not set for files that are on remote systems.

FILENOTFOUND

RESP2 values:

- 1** The name specified in the FILE option cannot be found in the FCT.

Default action: Terminate the task abnormally.

ILLOGIC

RESP2 values:

- 110** There is an error that does not fall within one of the other CICS response categories. (Further information is available in the EIBRCODE field; refer to Appendix A, "EXEC interface block," on page 529 for details.)

Default action: Terminate the task abnormally.

INVREQ

RESP2 values:

- 25** The KEYLENGTH and GENERIC options are specified, and the length specified in the KEYLENGTH option is greater than or equal to the length of a full key.
- 26** The KEYLENGTH option is specified (but the GENERIC option is not specified), and the specified length differs from the length defined for the underlying file.
- 36** A RESETBR command is issued for a file for which no previous STARTBR command has been successfully issued.
- 42** The KEYLENGTH and GENERIC options are specified, and the length specified in the KEYLENGTH option is less than zero.

Default action: Terminate the task abnormally.

IOERR

RESP2 values:

- 120** There is an I/O error during the file control operation. An I/O error is any unusual event that is not covered by a CICS exception condition.

(Further information is available in the EIBRCODE field; refer to Appendix A, "EXEC interface block," on page 529 for details.)

Default action: Terminate the task abnormally.

ISCINVREQ

RESP2 values:

- 70** The remote system indicates a failure that does not correspond to a known condition.

Default action: Terminate the task abnormally.

NOTAUTH

RESP2 values:

- 101** A resource security check has failed on FILE(*name*).

Default action: Terminate the task abnormally.

NOTFND

RESP2 values:

- 80** An attempt to retrieve a record based on the search argument provided is unsuccessful.

NOTFND can also occur if a generic RESETBR with KEYLENGTH(0) specifies the EQUAL option.

Default action: Terminate the task abnormally.

SYSIDERR

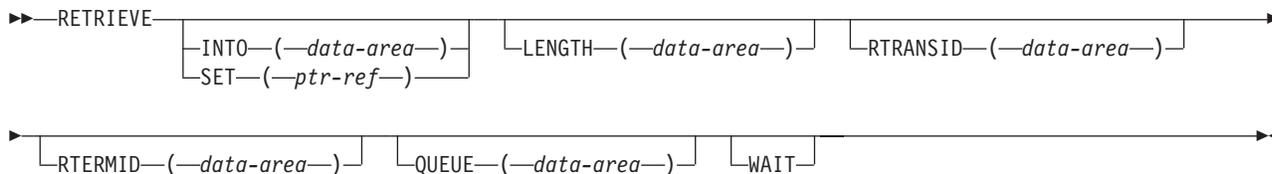
RESP2 values:

- 130** The SYSID option specifies either a name that is not defined in the terminal control system table (TCS), or a system to which the link is closed.

Default action: Terminate the task abnormally.

RETRIEVE

Retrieve data stored for a task.



Conditions: ENDDATA, ENVDEFERR, INVREQ, IOERR, LENGERR, NOTFND

Description

The RETRIEVE command retrieves data stored by expired START commands. It is the only method available for accessing such data.

You can use the RTRANSID, RTERMID, and QUEUE options to retrieve further data stored by expired START commands. These options can contain arbitrary data values whose meanings depend on what you have specified in the starting and started tasks. For further information, see the START command on page 445.

A task that is not associated with a terminal can access only the single data record associated with the original START command; it does so by issuing a RETRIEVE command. The storage occupied by the data associated with the task is released on execution of the RETRIEVE command, or on termination of the task if no RETRIEVE command is executed prior to termination.

A task that is associated with a terminal can access all data records associated with all expired START commands having the same transaction identifier and terminal identifier as this task, that is the task issuing the RETRIEVE command; it does so by issuing consecutive RETRIEVE commands. Expired data records are presented to the task on request in expiration-time sequence, starting with any data stored by the command that started the task, and including data from any commands that have expired since the task started. Each data record is retrieved from temporary storage using the REQID of the original START command as the identification of the record in temporary storage.

When all expired data records have been retrieved, the ENDDATA condition occurs. The storage occupied by the single data record associated with a START command is released after the data has been retrieved by a RETRIEVE command; any storage occupied by data that has not been retrieved is released when the CICS system is terminated.

Options

INTO(*data-area*)

specifies the user data area into which retrieved data is to be written.

LENGTH(*data-area*)

specifies a halfword binary value to define the length of the data area the retrieved data is written into.

If you specify the INTO option, the argument must be a data area that specifies the maximum length of data that the program is prepared to handle. If the value specified is less than zero, zero is assumed. If the length of the data exceeds the value specified, the data is truncated to that value and the LENGERR condition occurs. On completion of the retrieval operation, the data area is set to the original length of the data.

If you specify the SET option, the argument must be a data area. On completion of the retrieval operation, the data area is set to the length of the data.

For a description of a safe upper limit, see “LENGTH options” on page 312.

QUEUE(*data-area*)

specifies the 8-character area for the temporary storage queue name that may be accessed by the transaction issuing the RETRIEVE command.

RTERMID(*data-area*)

specifies a 4-character area that can be used in the TERMID option of a START command that may be executed subsequently.

RTRANSID(*data-area*)

specifies a 4-character area that can be used in the TRANSID option of a START command that may be executed subsequently.

SET(*ptr-ref*)

specifies the pointer reference to be set to the address of the retrieved data.

WAIT

specifies that, if all expired data records have already been retrieved, the task is to be put into a wait state until further expired data records become available. Although this means that the ENDDATA condition is not raised at the time the RETRIEVE command is issued, it is raised later if CICS enters shutdown or if the task is subject to deadlock time-out and it waits for longer than the deadlock time-out interval. (See the DTIMOUT option of RDO DEFINE TRANSACTION.)

An attempt to issue RETRIEVE WAIT during shutdown leads to an AICB abend if there is no data record already available to satisfy the request.

Exception Conditions

ENDDATA

occurs in any of the following situations:

- No more data is stored for the task issuing a RETRIEVE command. It can be considered a normal end-of-file response when retrieving data records sequentially.
- The RETRIEVE command is issued by a task that is started by a START command that did not specify any of the data options FROM, RTRANSID, RTERMID, or QUEUE.
- The RETRIEVE command is issued by a nonterminal task that was not created as a result of a START command.
- WAIT was specified and the task was waiting for a data record but none became available before the deadlock time-out (see the WAITTIME option of the transaction definition).
- WAIT was specified and the task was waiting when CICS entered shutdown. An attempt to issue RETRIEVE WAIT during shutdown leads to an AICB abend if there is no data record already available to satisfy the request.

Default action: terminate the task abnormally.

ENVDEFERR

occurs when a RETRIEVE command specifies an option not specified by the corresponding START command.

Default action: terminate the task abnormally.

INVREQ

occurs if the RETRIEVE command is not valid for processing by CICS.

Default action: terminate the task abnormally.

IOERR

occurs if an input/output error occurs during a RETRIEVE operation. The operation can be retried by reissuing the RETRIEVE command.

Default action: terminate the task abnormally.

The LENGTH option specifies the length of the data to be passed. The LENGTH value being passed must not be greater than the length of the data area specified in the COMMAREA option; otherwise the results are unpredictable. This may result in a LENGERR condition, as described in “Passing data to other programs” on page 201.

The valid range for the COMMAREA length is 0 through 32 763 bytes. If the length provided is outside this range, the LENGERR condition occurs. The COMMAREA and IMMEDIATE options can be used only when the EXEC CICS RETURN command is returning control to CICS; otherwise, the INVREQ condition occurs.

No resource security checking occurs on the RETURN TRANSID command. However, transaction security checking is still available when CICS attaches the returned transaction.

See Chapter 20, “Program control,” on page 199 for more information about program control.

Options

COMMAREA(*data-area*)

specifies a communication area that is to be made available to the next program that receives control. In a COBOL receiving program, you must give this data area the name DFHCOMMAREA. Because the data area is freed before the next program starts, a copy of the data area is created and a pointer to the copy is passed. In a C receiving program, this data area must be referenced by an EXEC CICS ADDRESS COMMAREA command.

The specified communication area is passed to the initial program of the next transaction that runs at the terminal. To ensure that the communication area is passed to the correct program, include the IMMEDIATE option.

This option is valid only on an EXEC CICS RETURN command issued by a program at the highest logical level, that is, a program returning control to CICS.

If you have defined the terminal with ATISTS(*YES) in the ADDCICSTCT CL command, the next program is not guaranteed to be part of the transaction specified by TRANSID. It could be part of a transaction started by a previously specified automatic transaction initiation (ATI) that has already been passed to that program. To make sure that the communication area is passed to the correct program, either define the terminal with ATISTS(*NO) (indicating no ATI) in the ADDCICSTCT CL command, or use the IMMEDIATE option.

IMMEDIATE

ensures that the transaction specified in the TRANSID option is attached as the next transaction regardless of any other transactions enqueued by ATI for this terminal. The next transaction starts immediately and appears to the operator as having been started by terminal data. If the terminal is using bracket protocol, the terminal is also held in bracket. This option is valid only on an EXEC CICS RETURN command issued by a program at the highest logical level, that is a program returning control to CICS.

INPUTMSG(*data-area*)

specifies data to be passed either to another transaction, identified by the TRANSID option, or to a calling program in a multiprogram transaction.

The data in the INPUTMSG data area is passed to the first program to issue an EXEC CICS RECEIVE command following the RETURN.

See Chapter 20, “Program control,” on page 199 for more information and illustrations about the use of INPUTMSG.

INPUTMSGLEN(*data-value*)

specifies as a halfword binary value the length of the INPUTMSG data. If the value is negative, zero is assumed.

LENGTH(*data-value*)

specifies as a halfword binary value the length in bytes of the communication area. If a negative value is supplied, zero is assumed. For a description of a safe upper limit, see “LENGTH options” on page 312.

TRANSID(*name*)

specifies the transaction identifier to be used with the next input message entered from the terminal with which the task that issued the RETURN command has been associated. The specified name can be up to 4 characters long and must have been defined as a transaction to CICS.

If the terminal is defined with ATISTS(*YES) in the ADDCICSTCT CL command, and the IMMEDIATE option is not specified, a transaction started by ATI may be run before the next transaction started by terminal input. If this happens, and the transaction identifier of the transaction started by ATI is the same as that specified in the TRANSID option, CICS assumes that the transaction started by ATI performs the same function and erases the “name” specified in the TRANSID option.

This option is not valid if the transaction issuing the RETURN command is not associated with a terminal, or is associated with an APPC logical unit.

Exception Conditions

INVREQ

RESP2 values:

- 1 A RETURN command with the TRANSID option is issued in a program that is not associated with a terminal.
- 2 A RETURN command with the COMMAREA or IMMEDIATE option is issued in a program that is not at the highest logical level.
- 3 A RETURN command is issued with a null address in the COMMAREA option.
- 4 A RETURN command with the TRANSID option is issued in a program that is associated with an APPC logical unit.
- 8 A RETURN command is issued in a program that is not associated with a terminal.
- 200 A RETURN command is issued with an INPUTMSG option in a program invoked by DPL.

Default action: terminate the task abnormally.

LENGERR

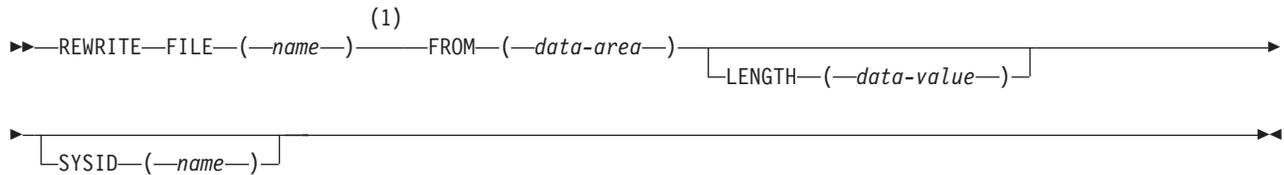
- The length specified on the LENGTH option is greater than the length of the data area specified in the COMMAREA option, and while that data was being copied a destructive overlap occurred because of the incorrect length.
- The LENGTH value is outside the range 1 through 32 763.
- The INPUTMSGLEN value is outside the range 1 through 32 767.

- INPUTMSG is supplied with a null address.

Default action: Terminate the task abnormally.

REWRITE

Update a record in a file.



Notes:

- 1 DATASET is also accepted, but FILE is the preferred term (see “DATASET option” on page 311).

Conditions: DUPREC, FILENOTFOUND, ILLOGIC, INVREQ, IOERR, ISCINVREQ, LENGERR, NOSPACE, NOTAUTH, SYSIDERR

Description

The REWRITE command updates a record in a file on a local or a remote system. You must always precede this command with a READ UPDATE to read the record to be updated.

Any key embedded in the record of the physical file should not be changed.

Refer to Chapter 10, “File control,” on page 115 for more information about this command.

Options

FILE(name)

specifies the name of the file to be accessed. The name must be alphanumeric, up to 8 characters long, and must have been defined in the file control table (FCT) unless the SYSID option specifies a remote system.

If a nonlocal SYSID is specified, the underlying file is assumed to be on a remote system irrespective of whether the name is defined in the local FCT. Otherwise, the FCT entry is used to determine whether the underlying file is on a local or a remote system.

FROM(data-area)

specifies the record to be written to the file.

LENGTH(data-value)

specifies as a halfword binary value the length of the record to be written.

If SYSID is specified, LENGTH must be coded. You must also specify this option for a file defined as containing variable-length records. It need not be specified if the file contains fixed-length records, but its inclusion is recommended because it causes a check to be made to ensure that the length of data being written is equal to that defined for the file. If the lengths are not equal, the LENGERR condition occurs.

SYSID(*name*)

specifies the name of the system to which the request is directed. The name can be up to 4 characters long.

If you specify *SYSID*, you must also specify *LENGTH*; it cannot be found in the *FCT*.

Exception Conditions

Note: *RESP2* values are not set for files that are on remote systems.

DUPREC

RESP2 values:

- 150** The record to be written contains a key that is the same as the key of a record already in the file, and the file accepts only unique keys. The key may be a primary key of a *KSDS* or an alternate key of a logical view of a file.

Default action: Terminate the task abnormally.

FILENOTFOUND

RESP2 values:

- 1** The name specified in the *FILE* option cannot be found in the *FCT*.

Default action: Terminate the task abnormally.

ILLOGIC

RESP2 values:

- 110** There is an error that does not fall within one of the other *CICS* response categories. (Further information is available in the *EIBRCODE* field; refer to Appendix A, "EXEC interface block," on page 529 for details.)

Default action: Terminate the task abnormally.

INVREQ

RESP2 values:

- 30** A *REWRITE* command is issued for a file that has not had a previous *READ UPDATE* successfully issued.

Default action: Terminate the task abnormally.

IOERR

RESP2 values:

- 120** There is an I/O error during the file control operation. An I/O error is any unusual event that is not covered by a *CICS* exception condition.

(Further information is available in the *EIBRCODE* field; refer to Appendix A, "EXEC interface block," on page 529 for details.)

Default action: Terminate the task abnormally.

ISCINVREQ

RESP2 values:

- 70** The remote system indicates a failure that does not correspond to a known condition.

Default action: Terminate the task abnormally.

LENGERR

RESP2 values:

- 10 The LENGTH option is not specified for a file with variable-length records.
- 12 The length specified exceeds the maximum record size; the record is truncated.
- 14 An incorrect length is specified for a file with fixed-length records.

Default action: Terminate the task abnormally.

NOSPACE

RESP2 values:

- 100 No space is available on the disk for adding the updated record to the file

Default action: Terminate the task abnormally.

NOTAUTH

RESP2 values:

- 101 A resource security check has failed on FILE(*name*).

Default action: Terminate the task abnormally.

SYSIDERR

RESP2 values:

- 130 The SYSID option specifies either a name that is not defined in the terminal control system table (TCS), or a system to which the link is closed.

Default action: Terminate the task abnormally.

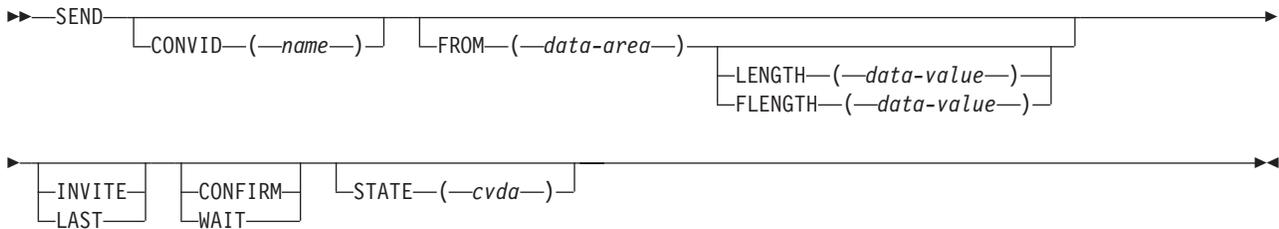
Examples

For example:

```
EXEC CICS REWRITE
      FROM(RECORD)
      FILE('MASTER')
      LENGTH(RLENGTH)
      :
```

SEND (APPC)

Send data and control information to a conversation partner in an APPC conversation.



Conditions: INVREQ, LENGERR, NOTALLOC, SIGNAL, TERMERR

Description

The SEND command sends data and control information to a conversation partner in an APPC conversation.

See part 3 of *CICS/400 Intercommunication* for more information.

Options

CONFIRM

indicates that an application using a sync level 1 or 2 conversation requires a response from the remote application. A remote CICS application can respond positively by executing an `ISSUE CONFIRMATION` command; or negatively by executing an `ISSUE ERROR` command, in which case the sending application has `EIBERR` and `EIBERRCD` set. CICS does not return control to the sending application until the response is received.

CONVID(*name*)

identifies the conversation to which the command relates. The 4-character name identifies the token returned by a previously executed `ALLOCATE` command in the `EIBRSRCE` field of the `EIB`. If this option is omitted, the principal facility for the task is used by default.

FLENGTH(*data-value*)

is a fullword alternative to `LENGTH(data-value)`.

FROM(*data-area*)

specifies the data to be sent to a partner transaction. This option may be omitted if `INVITE`, `LAST`, `CONFIRM`, or `WAIT` is specified.

INVITE

sets change direction (CD) on the next flow.

LAST

specifies that this is the last SEND command for a transaction.

LENGTH(*data-value*)

specifies as a halfword binary value the length of the data to be sent.

STATE(*cvda*)

gets the state of the current conversation. For a complete list of the CVDA values that can be returned on APPC commands and for information about receiving and testing these values, see "CICS-value data areas (CVDA)" on page 309.

WAIT

specifies that processing of the command must be completed before any subsequent processing is attempted.

If the WAIT option is not specified, control is returned to the application program when processing of the command has started. A subsequent input or output request (terminal control or BMS) to the terminal associated with the task causes the application program to wait until the previous request has been completed.

Exception Conditions

INVREQ

RESP2 values:

- The CONFIRM option has been specified, but the APPC conversation is not sync level 1.

The CONVID value was obtained by an ASSIGN FACILITY command. However, the principal facility is *not* an APPC conversation.

- 200** The SEND (APPC) command is issued in a DPL server program and refers to the principal facility.

Default action: Terminate the task abnormally.

LENGERR

occurs if an out-of-range value is specified for the LENGTH or FLENGTH option.

Default action: Terminate the task abnormally.

NOTALLOC

occurs if the specified CONVID value does not relate to a conversation owned by the application.

Default action: Terminate the task abnormally.

SIGNAL

occurs if an inbound SIGNAL data-flow control command has been received from a logical unit or session.

Default action: Ignore the condition.

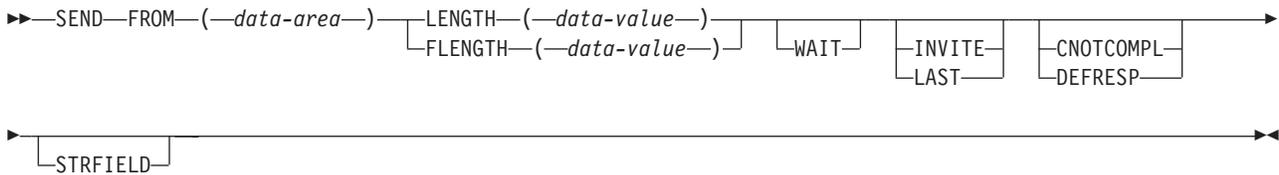
TERMERR

occurs if there is a session-related error. Any action on that conversation other than a FREE command causes an ATCV abend.

Default action: Terminate the task abnormally (with abend code ATNI).

SEND (SCS)

Write data to a 3270 SCS printer logical unit.



Conditions: INVREQ, LENGERR, TERMERR

Description

The SCS printer logical unit accepts a character string as defined by the Systems Network Architecture (SNA).

See Chapter 14, "Terminal control," on page 169 for more information about this command.

Options

CNOTCOMPL (ignored by CICS/400)

indicates that the request/response unit (RU) sent as a result of this SEND command does not complete the chain. If this option is omitted and chain assembly has been specified, the RU terminates the chain.

DEFRESP (ignored by CICS/400)

indicates that a definite response is required when the output operation has been completed.

FLENGTH(data-value)

is a fullword alternative to LENGTH(data-value).

FROM(data-area)

specifies the data to be written to the logical unit.

INVITE

specifies that the next terminal control command to be executed for this facility is a RECEIVE. This allows optimal flows to occur.

LAST (ignored by CICS/400)

specifies that this is the last output operation for a transaction and therefore the end of a bracket.

LENGTH(data-value)

specifies as a halfword binary value the length of the data to be written.

For a description of a safe upper limit, see "LENGTH options" on page 312.

STRFIELD

specifies that the data area specified in the FROM option contains structured fields. This option applies to 3270 devices only. When this option is specified, the contents of all structured fields must be handled by the application program. (Structured fields are described in the *IBM 3270 Information Display System Data Stream Programmer's Reference* manual.)

WAIT

specifies that processing of the command must be completed before any subsequent processing is attempted.

If the WAIT option is not specified, control is returned to the application program when processing of the command has started. A subsequent input or output request (terminal control or BMS) to the terminal associated with the task causes the application program to wait until the previous request has been completed.

Exception Conditions

INVREQ

RESP2 values:

200 The command is issued in a DPL server program.

Default action: Terminate the task abnormally.

LENGERR

occurs if an out-of-range value is supplied in the LENGTH or FLENGTH option.

Default action: Terminate the task abnormally.

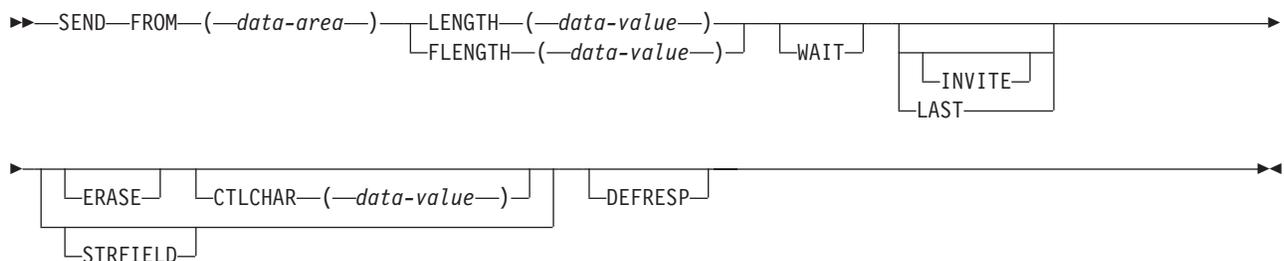
TERMERR

occurs if there is a terminal-related error.

Default action: Terminate the task abnormally (with abend code ATNI).

SEND (5250 or 3270 logical)

Write (or send) data to a 3270 or 5250 display.



Conditions: INVREQ, LENGERR, TERMERR

Description

The SEND (5250 or 3270 logical) command writes (or sends) data to a 3270 or 5250 display.

See Chapter 14, "Terminal control," on page 169 for more information about this command.

Options

CTLCHAR(*data-value*)

specifies a 1-byte write control character (WCC) that controls a SEND command for a terminal. A COBOL user must specify a data area containing this character. If the option is omitted, all modified data tags are reset to zero and the keyboard is restored.

DEFRESP (ignored by CICS/400)

indicates that a definite response is required when the output operation has been completed.

ERASE

specifies that the screen is to be erased and the cursor returned to the upper left corner of the screen before writing occurs.

Normally, ERASE should be specified in the first output command of a transaction. This clears the screen ready for the new output data.

However, when switching from one screen size to another on a transaction basis, if ERASE is not specified in the first output command of the transaction, the screen size is unchanged from its previous setting (that is, the previous transaction setting, or the default screen size if the CLEAR key has been pressed).

FLENGTH(data-value)

is a fullword alternative to LENGTH(data-value).

FROM(data-area)

specifies the data to be written to the logical unit.

INVITE

specifies that the next terminal control command to be executed for this facility is a RECEIVE. This allows optimal flows to occur.

LAST (ignored by CICS/400)

specifies that this is the last output operation for a transaction and therefore the end of a bracket.

LENGTH(data-value)

specifies as a halfword binary value the length of the data to be written.

For a description of a safe upper limit, see "LENGTH options" on page 312.

STRFIELD

specifies that the data area specified in the FROM option contains structured fields. This option is valid for 3270 devices only. When this option is specified, the contents of all structured fields must be handled by the application program. (Structured fields are described in the *IBM 3270 Information Display System Data Stream Programmer's Reference* manual.) CTLCHAR and ERASE are mutually exclusive with STRFIELD, and their use with STRFIELD generates an error message.

WAIT

specifies that processing of the command must be completed before any subsequent processing is attempted.

If the WAIT option is not specified, control is returned to the application program when processing of the command has started. A subsequent input or output request (terminal control or BMS) to the terminal associated with the task causes the application program to wait until the previous request has been completed.

Exception Conditions

INVREQ

RESP2 values:

200 The command is issued in a DPL server program.

Default action: Terminate the task abnormally.

LENGERR

occurs if an out-of-range value is supplied in the LENGTH or FLENGTH option.

Default action: Terminate the task abnormally.

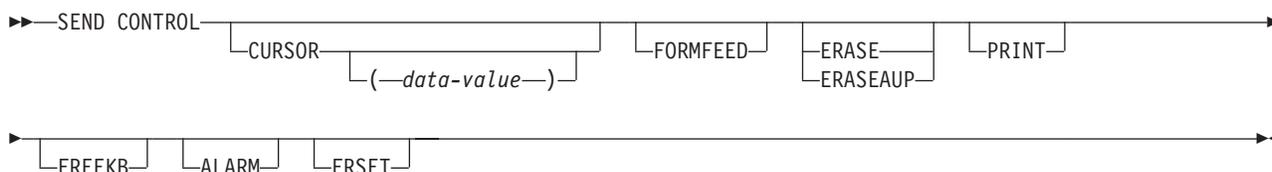
TERMERR

occurs if there is a terminal-related error.

Default action: Terminate the task abnormally (with abend code ATNI).

SEND CONTROL

Send device controls to a terminal without map or text data.



Conditions: INVREQ

Description

The SEND CONTROL command sends device controls to a terminal without map or text data.

For further information about SEND CONTROL, see “Sending data to a display device” on page 155.

Options

ALARM

specifies that the audible alarm feature is to be activated.

CURSOR(*data-value*)

specifies as a halfword binary value the position to which the cursor is to be returned on completion of a SEND CONTROL command.

The supplied value gives the cursor position relative to zero; the range of possible values depends on the size of the screen being used.

The value specified in the CURSOR option must be positive. A negative value leads to unpredictable results.

If this option is omitted, the cursor is positioned at position zero of the screen.

ERASE

specifies that the screen printer buffer is to be erased and the cursor returned to the upper left corner of the screen. The first output operation in any transaction, or in a series of pseudoconversational transactions, should always specify ERASE. For transactions attached to 3270 or 5250 screens or printers, this also ensures that the correct screen size is selected, as defined for the transaction using the ADDCICSPCT CL command. This command is described in the *CICS/400 Administration and Operations Guide*.

ERASEAUP

specifies that all unprotected character locations on the entire screen are to be erased.

FORMFEED

specifies that a new page is required. The FORMFEED character is positioned at the start of the display or printer buffer. The application program must ensure that this buffer position is not overwritten by map or text data.

FREEKB

specifies that the 3270 keyboard is to be unlocked. If FREEKB is omitted, the keyboard remains locked. This option has no effect on 5250 keyboards.

FRSET

specifies that the modified data tags (MDTs) of all fields currently in the screen buffer are to be reset to the not-modified condition (that is, field reset).

This allows the ATTRB operand of the DFHMDF macro for the next requested map to control the final status of fields written or rewritten in response to a BMS command, if no other attribute information has been written in the symbolic map.

PRINT

specifies that a print operation is to be started at a 3270 or SCS printer, or that data on a 3270 display is to be printed on a printer allocated by the controller.

Exception Conditions

INVREQ

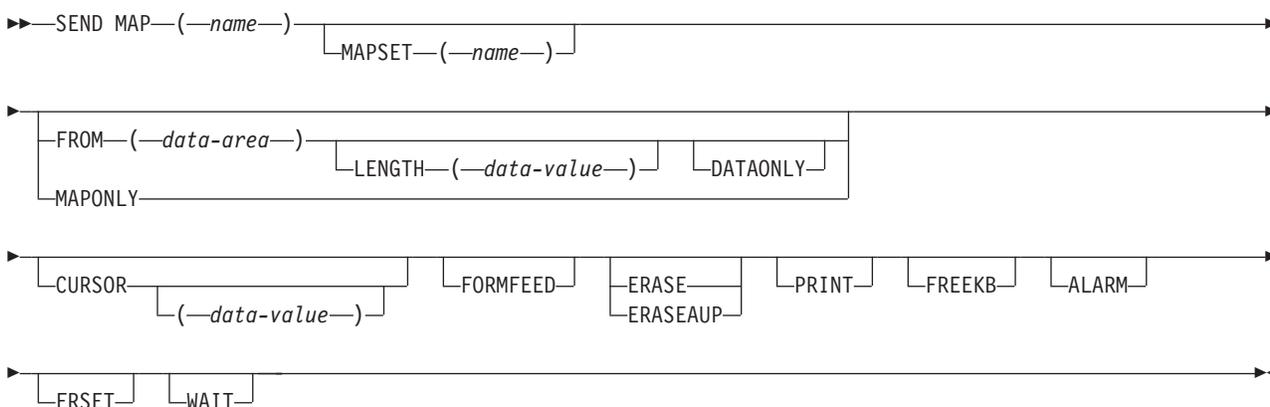
RESP2 values:

200 The command is issued in a DPL server program.

Default action: Terminate the task abnormally.

SEND MAP

SEND MAP sends output data to a terminal.



Conditions: INVMPSZ, INVREQ, NOTAUTH

Description

SEND MAP sends output data to a terminal.

When using the SEND MAP command with any of the ALARM, FREEKB, FRSET, or PRINT options, refer to the CTRL operand of the DFHMDF macro on page 563 for a description of the option priority.

For further information about SEND MAP, see “Sending data to a display device” on page 155.

See Appendix D, “BMS macro summary,” on page 553 for the map definition macros.

Options

ALARM

specifies that the audible alarm feature is to be activated.

When using the ALARM option, refer to the CTRL operand on page 563 for a description of the option priority.

CURSOR(*data-value*)

specifies as a halfword binary value the position to which the cursor is to be returned on completion of a SEND MAP command.

The supplied value gives the cursor position relative to zero; the range of possible values depends on the size of the screen being used. If no data value is specified, symbolic cursor positioning is assumed. See “Cursor positioning” on page 159 for more information about symbolic cursor positioning.

This option overrides any IC option of the ATTRB operand of the DFHMDF macro.

The value specified in the CURSOR option must be positive. A negative value leads to unpredictable results.

DATAONLY

specifies that only application program data is to be written. The attribute characters must be specified for each field in the supplied data. If the attribute byte in the user-supplied data is set to X'00', the attribute byte on the screen is unchanged. Any default data or attributes from the map are ignored.

ERASE

specifies that the screen or printer buffer is to be erased and the cursor returned to the upper left corner of the screen before the map is displayed. The first output operation in any transaction, or in a series of pseudoconversational transactions, should always specify ERASE. For transactions attached to 3270 or 5250 screens or printers, this also ensures that the correct screen size is selected, as defined for the transaction using the ADDCICSPCT CL command. This command is described in the *CICS/400 Administration and Operations Guide*.

ERASEAUP

specifies that, before this map is displayed, all unprotected character locations on the entire screen are to be erased.

FORMFEED

specifies that a new page is required. The FORMFEED character is positioned at the start of the display or printer buffer. The application program must thus ensure that this buffer position is not overwritten by map or text data.

FREEKB

specifies that the 3270 keyboard is to be unlocked. If FREEKB is omitted, the keyboard remains locked. This option has no effect on 5250 keyboards.

When using the FREEKB option, refer to the CTRL operand on page 563 for a description of the option priority.

FROM(*data-area*)

specifies the data area containing the data to be processed. If the FROM and

MAPONLY options are both omitted, the name of the data area defaults to the name of the map set suffixed with an O. This includes the 12-byte prefix generated by the TIOAPFX=YES operand of the DFHMSD BMS map set definition macro (see page 560).

FRSET

specifies that the modified data tags (MDTs) of all fields currently in the terminal buffer are to be reset to the not-modified condition (that is, field reset) before any map data is written to the buffer.

This allows the ATTRB operand of the DFHMDF macro for the requested map to control the final status of fields written or rewritten in response to a BMS command, if no other attribute information has been written in the symbolic map.

When using the FRSET option, refer to the CTRL operand on page 563 for a description of the option priority.

LENGTH(*data-value*)

specifies as a halfword binary value the length of the data to be formatted.

If the data area sending the map is longer than the data to be mapped, LENGTH should be specified. This must include the length of the 12-byte prefix generated by the TIOAPFX=YES operand of the DFHMSD BMS map definition macro (see page 560).

For a description of a safe upper limit, see “LENGTH options” on page 312.

MAP(*name*)

specifies the name of the map to be used. The name can be up to 7 characters long.

MAPONLY

specifies that only default data from the map is to be written.

MAPSET(*name*)

specifies either an unsuffixed or a suffixed name of the map set. An unsuffixed map set name can be up to 7 characters long, and a suffixed map set name can be from 2 through 8 characters long. The map set must reside in the user's OS/400 library list, and it must be defined using the appropriate program definition. If the MAPSET option is not specified, the name given in the MAP option is assumed to be that of the map set.

PRINT

specifies that a print operation is to be started at a 3270 or SCS printer, or that data on a 3270 display is to be printed on a printer allocated by the controller. If this option is omitted, the data is sent to the printer buffer but is not printed.

When using the PRINT option, refer to the CTRL operand on page 563 for a description of the option priority.

WAIT

specifies that control should not be returned to the application program until the output operation has been completed.

If WAIT is not specified, control returns to the application program when the output operation has started. A subsequent input or output request (terminal control or BMS) causes the application program to wait until the previous request has been completed.

Exception Conditions

INVMP SZ

occurs if the specified map is too wide or too long for the terminal.

Default action: Terminate the task abnormally.

INVREQ

RESP2 values:

- A SEND MAP command is issued for a map without field specifications by specifying the FROM option without the DATAONLY option.

200 The command is issued in a DPL server program.

Default action: Terminate the task abnormally.

NOTAUTH

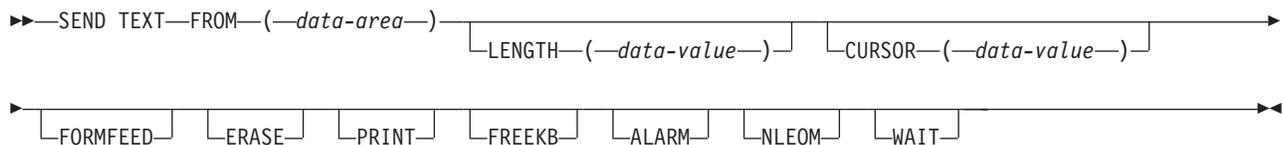
RESP2 values:

70 A resource security check has failed on MAPSET(name).

Default action: Terminate the task abnormally.

SEND TEXT

Send text data without mapping.



Conditions: INVREQ

Description

The SEND TEXT command sends text data without mapping.

The text is split into lines of the same width as the terminal, such that words are not broken across line boundaries. If the text exceeds a page, it is split into pages that fit on the terminal with application-defined headers and trailers.

For further information about SEND TEXT, see “Text processing” on page 164.

Options

ALARM

specifies that the audible alarm feature is to be activated.

CURSOR(*data-value*)

specifies as a halfword binary value the position to which the cursor is to be returned on completion of a SEND TEXT command.

The supplied value gives the cursor position relative to zero; the range of possible values depends on the size of the screen being used.

The value specified in the CURSOR option must be positive. A negative value leads to unpredictable results.

ERASE

specifies that the screen printer buffer is to be erased and the cursor returned to the upper left corner of the screen before this page of output is displayed. The first output operation in any transaction, or in a series of pseudoconversational transactions, should always specify ERASE. For transactions attached to 3270 or 5250 screens or printers, this also ensures that the correct screen size is selected, as defined for the transaction using the ADDCICSPCT CL command. This command is described in the *CICS/400 Administration and Operations Guide*.

5250 screens are automatically erased to avoid possible device errors.

FORMFEED

specifies that a new page is required. The FORMFEED character is positioned at the start of the display or printer buffer. The application program must thus ensure that this buffer position is not overwritten by the map or text data.

FREEKB

specifies that the 3270 keyboard is to be unlocked. If FREEKB is omitted, the keyboard remains locked. This option has no effect on 5250 keyboards.

FROM(*data-area*)

specifies the data area containing the data to be sent.

LENGTH(*data-value*)

specifies as a halfword binary value the length of the data to be sent. For a description of a safe upper limit, see "LENGTH options" on page 312.

NLEOM

specifies that data for a 3270 or SCS printer is to be built with blanks and new-line (NL) characters, and that an end-of-message (EM) character is to be placed at the end of the data. As the data is printed, each NL character causes printing to continue on the next line, and the EM character terminates printing.

This option must be specified in the first SEND TEXT command used to build a logical message. The option is ignored if the device receiving the message is not a 3270 or SCS printer.

The NLEOM option overrides the ALARM option if the latter is present.

PRINT

specifies that a print operation is to be started at a 3270 or SCS printer, or that data on a 3270 display is to be printed on a printer allocated by the controller. If this option is omitted, the data is sent to the printer buffer but is not printed.

WAIT

specifies that control should not be returned to the application program until the output operation has been completed.

If WAIT is not specified, control returns to the application program when the output operation has started. A subsequent input or output request (terminal control or BMS) causes the application program to wait until the previous request has been completed.

Exception Conditions

INVREQ

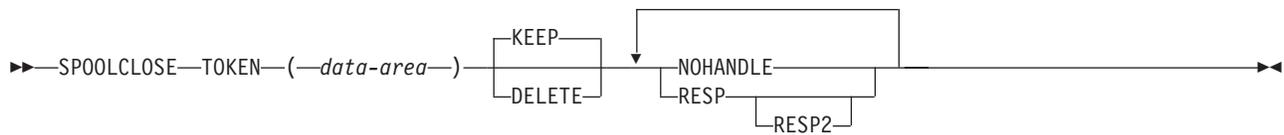
RESP2 values:

200 The command is issued in a DPL server program.

Default action: Terminate the task abnormally.

SPOOLCLOSE

Close a CICS spool report and optionally change its retention characteristics in the CICS control region.



Conditions: INVREQ, NOTFND, NOTOPEN

Description

SPOOLCLOSE closes a CICS spool report and optionally changes its retention characteristics in the CICS control region.

You must specify either the NOHANDLE option or the RESP and RESP2 options when you use any of the printer spooling commands. Failure to do so causes your program to abend with abend code APST.

Refer to Chapter 25, “Printer spooling,” on page 223 for more information about spooling.

Options

DELETE

specifies that the CICS spool report is to be deleted (that is, purged).

KEEP

specifies that the CICS spool report is to be kept (that is, it remains on the OS/400 system).

A default disposition of KEEP is taken if both KEEP and DELETE are omitted from the SPOOLCLOSE command, or if the report is closed implicitly by an EXEC CICS SYNCPOINT or EXEC CICS RETURN command.

TOKEN(*data-area*)

specifies the 8-character token allocated by CICS to identify a spool report.

Exception Conditions

INVREQ

occurs if the specified TOKEN is not a valid token.

Default action: Terminate the task abnormally.

NOTFND

occurs if the spool file indicated by the TOKEN option is open when the SPOOLCLOSE command is issued but an external CICS error has occurred preventing CICS from successfully closing the spool file.

Default action: Terminate the task abnormally.

NOTOPEN

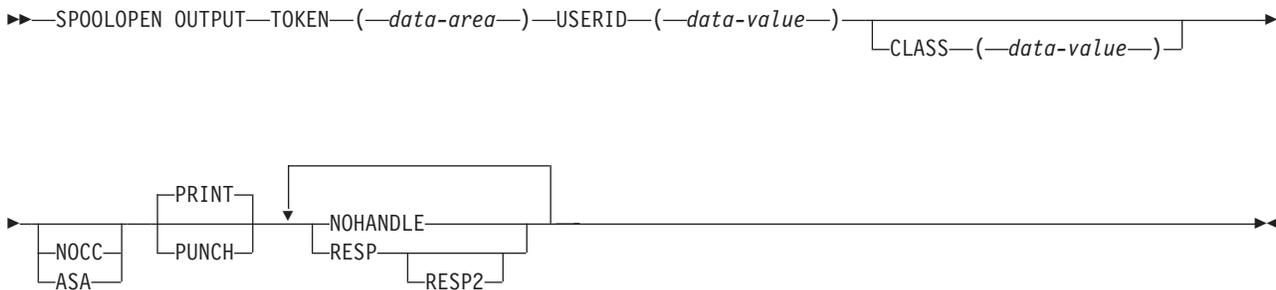
RESP2 values:

8 The spool file indicated by the TOKEN option is no longer open.

Default action: Terminate the task abnormally.

SPOOLOPEN OUTPUT

Open a CICS spool report.



Conditions: NOTFND, NOTOPEN, OPENERR, SPOLBUSY

Description

SPOOLOPEN OUTPUT opens a CICS spool report from the CICS control region to the OS/400 spooler and defines its characteristics.

It results in a dynamic allocation of the local spool file using the USERID value to specify the user data. SPOOLOPEN OUTPUT enables users to acquire the token for a CICS spool report that it expects to create (write). This token is used to identify the report in later SPOOLWRITE and SPOOLCLOSE commands.

Use the NOCC or ASA options to control output formatting. If a format is not specified, the default value associated with the local spool files is used.

If a SPOOLCLOSE command is not issued before the end of the CICS transaction, CICS performs an implicit SPOOLCLOSE KEEP.

When you create the print file on the AS/400, if the data area to be written, using the EXEC CICS SPOOLWRITE command, will exceed the page width defined by the print file, you must ensure that you specify option SPOOL(*YES) on the CRTPRTF command. If you do not, and output is written directly to a printer, CICS/400 cannot accurately determine whether to fold a data stream over multiple lines.

You must specify either the NOHANDLE option or the RESP and RESP2 options when you use any of the printer spooling commands. Failure to do so causes your program to abend with abend code APST.

See Chapter 25, “Printer spooling,” on page 223 for more information about spooling.

Options

ASA

specifies that the CICS spool report to be created has each record prefixed with an ASA carriage-control character, and that this character must be used by the operating system to control formatting when the report is printed.

CLASS*(data-value)*

specifies a 1-character CICS class designation. If this option is omitted, class A is assumed.

NOCC

specifies that the CICS spool report to be created has no internal formatting controls. When the report is printed, the operating system prefixes each record with a carriage-control character that causes page skipping according to the default operating system lines-per-page value.

PRINT

specifies that the CICS spool report is to be produced as a listing. This is the default setting.

PUNCH

specifies that the CICS spool report is to be produced in card-image format.

TOKEN*(data-area)*

specifies an 8-character field to receive the token allocated by CICS to identify the spool report.

USERID*(data-value)*

specifies a 10-character field that may be used to identify the writer program or user id that will be used to process the CICS spool report for spooled records intended for a printer. The report carries this identifier in the user data area that is used to select the report.

Exception Conditions

NOTFND

RESP2 values:

- 4 A printer file for the control region and CLASS cannot be found or if the printer file name contains invalid characters. (The CLASS designation becomes part of the name of the printer file used.)

Default action: Terminate the task abnormally.

NOTOPEN

RESP2 values:

- 8 The spool file indicated by the CLASS option cannot be opened.

Default action: Terminate the task abnormally.

OPENERR

occurs in any of the following situations:

- The spool file indicated by the CLASS option indicates that output is not to be spooled, but the printer device to which the output is to be directed is unavailable at the time the command is issued.
- The number of SPOOLOPEN OUTPUT commands within a single job exceeds 9 999.
- You do not have the necessary authority to open the printer file

Default action: Terminate the task abnormally.

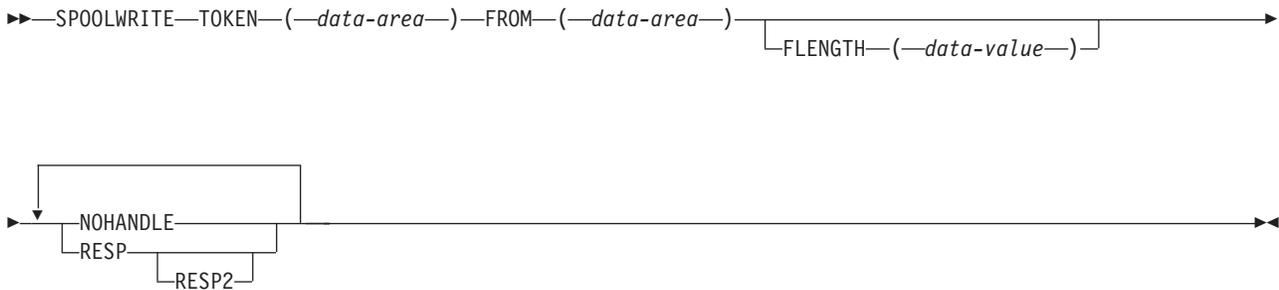
SPOLBUSY

occurs if there is an error while trying to open the file indicated by the CLASS and ASA/NOCC options on the command.

Default action: Terminate the task abnormally.

SPOOLWRITE

Write data to a CICS spool report.



Conditions: LENGERR, NOTOPEN, SPOLERR

Description

SPOOLWRITE writes data to a CICS spool report.

You must specify either the NOHANDLE option or the RESP and RESP2 options when you use any of the printer spooling commands. Failure to do so causes your program to abend with abend code APST.

See Chapter 25, “Printer spooling,” on page 223 for more information about spooling.

Options

FLENGTH(*data-value*)

specifies as a fullword binary value the length of data to be transferred to the CICS spool report. If this option is omitted, CICS uses the length of the data area.

FROM(*data-area*)

specifies the data area from which variable-length data is transferred. The data itself is not altered in any way by CICS. If the data area exceeds the page width defined in the AS/400 print file, CICS will fold the data stream over multiple lines if FOLD(*YES) is specified in the print file definition. CICS can ascertain this value accurately only if the print file is defined with the option SPOOL(*YES).

TOKEN(*data-area*)

specifies the 8-character token allocated by CICS to identify a spool report.

Exception Conditions

LENGERR

occurs in either of these situations:

- The value specified in the FLENGTH option is less than one or greater than the allowed maximum of 32760. The RESP2 field returns a zero value.
- The value specified in the FLENGTH option is greater than the line length specified for the OS/400 printer file, and the FOLD option for this file is set to *NO. CICS truncates the data to the length of the line and returns the number of characters lost through truncation in RESP2.

Default action: Terminate the task abnormally.

NOTOPEN

RESP2 values:

- 8 The spool file indicated by the TOKEN option is not open when the SPOOLWRITE is issued.

Default action: Terminate the task abnormally.

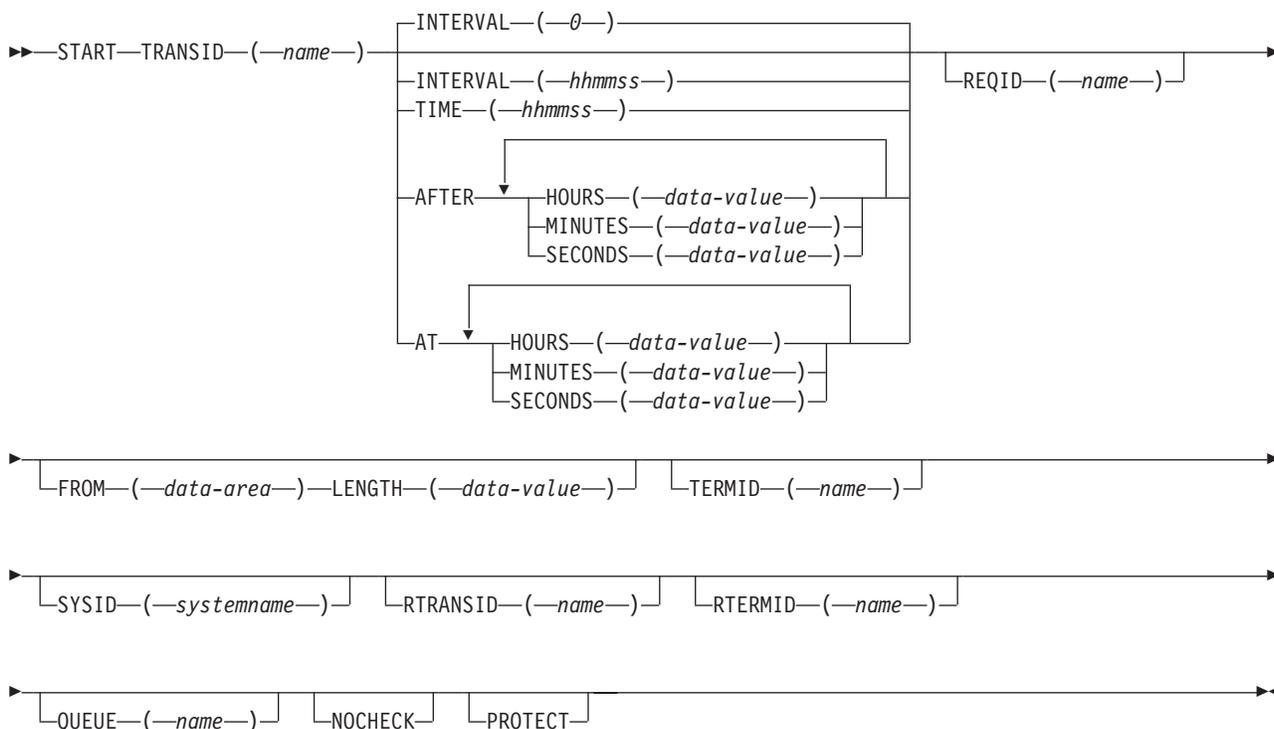
SPOLERR

occurs if an I/O error has been detected by the OS/400 data management modules, preventing successful completion of the request.

Default action: Terminate the task abnormally.

START

Start task at a specified time.



Conditions: INVREQ, IOERR, ISCVREQ, LENGERR, NOTAUTH, SYSIDERR, TERMIDERR, TRANSIDERR

Description

START starts a task, on a local or remote system, at a specified time. The time is specified by INTERVAL, AFTER, AT or TIME.

The starting task may pass data to the started task. This data must not contain any pointers. The starting task may also specify a terminal to be used by the started task as its principal facility.

The default is INTERVAL(0), but for ILE C the default is AFTER HOURS(0) MINUTES(0) SECONDS(0).

Note that **CEDF** is an exception to the **START** command and is not valid as a **TRANSID** name. You should therefore not attempt to start **CEDF** in this way.

You can use the **RTRANSID**, **RTERMID**, and **QUEUE** options to pass further data to the started task. These options can contain arbitrary data values whose meanings depend on what you have specified in the started and starting tasks. One possible way of using them is in the following situation. One task can start a second task, passing it a transaction name and a terminal name to be used when the second task starts a third task. The first task may also pass the name of a queue to be accessed by the second task. If you choose to use **QUEUE** to pass the name of a temporary storage queue on which to store data to a started transaction, it must not be the same as the name you specified in **REQID**, if used.

One or more constraints have to be satisfied before the transaction to be executed can be started, as follows:

- The specified interval must have elapsed or the specified expiration time must have been reached. (For more information, see Chapter 18, “Interval control,” on page 193.) The **INTERVAL** or **AFTER** options should be specified when a transaction is to be executed on a remote system; this avoids complications arising when the local and remote systems are in different time zones.
- If the **TERMID** option is specified, the named terminal must exist and be available. If the named terminal does not exist when the time interval expires, the **START** is discarded.
- If the **PROTECT** option is specified, the starting task must have taken a successful syncpoint. This option, coupled to extensions to system tables, reduces the exposure to lost or duplicated data caused by failure of a starting task.
- If the transaction to be executed is on a remote system, the format of the data must be declared to be the same as that at the local system. the **RDO** options **DATASTREAM** and **RECORDFORMAT**, or **DATASTR** and **RECFM** on the terminal control table **TYPE=SYSTEM**.

Execution of a **START** command naming a transaction in the local system cancels any outstanding **POST** commands executed by the starting task.

START commands are queued by means of the transaction definition created using the **ADDCICSPCT CL** command, as described in the *CICS for iSeries Administration and Operations Guide*.

If data is to be passed by interval control (using the **FROM** option), it is queued on a temporary storage queue. The **REQID** option allows you to specify the name of the temporary storage queue to be used. This identifier may be recoverable (in temporary storage terms) or nonrecoverable.

If you also specify the **PROTECT** option, the temporary storage queue identified by the **REQID** option should be defined as recoverable. If you do not specify the **PROTECT** option, the temporary storage queue should not be defined as recoverable. Unpredictable results can occur if these rules are not followed.

The **NOCHECK** option specifies that no response (to execution of the **START** command) is expected by the starting transaction. For **START** commands naming tasks to be started on a local system, error conditions are returned; error conditions are not returned for tasks to be started on a remote system. The **NOCHECK** option allows **CICS** to improve performance when the **START** command has to be

shipped to a remote system; it is also a prerequisite if the shipping of the START command is queued pending the establishing of links to the remote system.

When a START command is issued against an existing but idle CICS/400 shell, if the terminal is a 5250 the keyboard locks until the transaction is complete. However, if the terminal is a 3270, the keyboard does not lock. You should not enter anything until the transaction has finished running.

Examples

Starting tasks without terminals

If the task to be started is not associated with a terminal, each START command results in a separate task being started. This happens regardless of whether or not data is passed to the started task. The following examples show how to start a specified task, not associated with a terminal, in one hour:

```
EXEC CICS START
      TRANSID('TRNL')
      INTERVAL(10000)
      REQID('NONGL')
:
:
EXEC CICS START
      TRANSID('TRNL')
      AFTER HOURS(1)
      REQID('NONGL')
:
:
```

Starting tasks with terminals but without data

Only one task is started if several START commands, each specifying the same transaction and terminal, expire at the same time or before the terminal is available. When this happens the error message AEG0815 will appear on the job log.

The following examples show how to request initiation of a task associated with a terminal. Because no request identifier is specified in these examples, CICS assigns one and returns it to the application program in the EIBREQID field of the EXEC interface block.

```
EXEC CICS START
      TRANSID('TRN1')
      TIME(185000)
      TERMID('STA5')
:
:
EXEC CICS START
      TRANSID('TRN1')
      AT HOURS(18) MINUTES(50)
      TERMID('STA5')
:
:
```

Starting tasks with terminals and data

Data is passed to a started task if one or more of the FROM, RTRANSID, RTERMID, and QUEUE options is specified. Such data is accessed by the started task by using a RETRIEVE command.

It is possible to pass many data records to a new task by issuing several START commands, each specifying the same transaction and terminal.

Execution of the first START command ultimately causes the new task to be started and allows it to retrieve the data specified on the command. The new task is also able to retrieve data specified on subsequently executed START commands that expire before the new task is terminated. If such data has not been retrieved before the new task is terminated, another new task is started and is able to retrieve the outstanding data. If this second new task fails to retrieve the outstanding data, a third task is started, and so on, up to a maximum of 5 times, after which the data is lost.

The following examples show how to start a task associated with a terminal and pass data to it:

```
EXEC CICS START
  TRANSID('TRN2')
  TIME(173000)
  TERMID('STA3')
  REQID(DATAREC)
  FROM(DATAFLD)
  LENGTH(100)
:
EXEC CICS START
  TRANSID('TRN2')
  AT HOURS(17) MINUTES(30)
  TERMID('STA3')
  REQID(DATAREC)
  FROM(DATAFLD)
  LENGTH(100)
:
```

ILE C supports the packed-decimal argument type and, consequently, the use of the INTERVAL and TIME options. It is recommended, however, that you use the AFTER and AT options for portability of applications between CICS platforms.

START failures with out exception conditions

There are some circumstances in which a START command is executed without error, but the started task never takes place:

- When the transaction or its initial program is disabled at the time CICS attempts to create the task.
- When the START specifies a terminal and an expiration time, and the terminal is not defined at expiration time.
- When the START specifies a terminal that is not defined at the time CICS attempts to create the task.

These exposures result from the delay between the execution of the START and the time of task creation. Even when the START is immediate, CICS may delay creating the task, either because the required terminal is not free or because of other system constraints.

You can use INQUIRE commands to ensure that the transaction and program are enabled at the time of the START command, but either may become disabled before task creation.

You get a `TERMIDERR` condition if the requested terminal does not exist at the time of the `START`, but if it is deleted subsequently, as occurs if the user logs off, your `START` request is discarded with the terminal definition.

Refer to Chapter 18, “Interval control,” on page 193 for further information about Interval Control.

Options

AFTER

specifies the interval of time that is to elapse before the new task is started.

There are two ways to enter the time under `AFTER` and `AT`.

1. A combination of at least two of `HOURS(0–99)`, `MINUTES(0–59)`, and `SECONDS(0–59)`. `HOURS(1) SECONDS(3)` would mean one hour and three seconds (the minutes default to zero).
2. As one of `HOURS(0–99)`, `MINUTES(0–5999)`, or `SECONDS(0–359 999)`. `HOURS(1)` means one hour. `MINUTES(62)` means one hour and two minutes. `SECONDS(3723)` means one hour, two minutes, and three seconds.

AT

specifies the time at which the new task is to be started. For the ways to enter the time, see the `AFTER` option.

FROM(*data-area*)

specifies the data to be stored for a task that is to be started at some future time.

HOURS(*data-value*)

specifies a fullword binary value in the range 0–99. This is a suboption of the `AFTER` and `AT` options. For its use and meaning, see the `AFTER` option.

INTERVAL(*hhmmss*)

specifies the expiration time as an interval of time that is to elapse from the time at which the `START` command is issued. The specified interval is added to the current clock time by CICS to calculate the expiration time. See Chapter 18, “Interval control,” on page 193 for an explanation of how expiration times are used within interval control.

`INTERVAL(0)` is the default. The maximum permitted `INTERVAL` value is 995959.

For compatibility between CICS platforms, it is recommended that the `INTERVAL` option is not used in ILE C programs. You should use the `AFTER` option instead.

LENGTH(*data-value*)

specifies a halfword binary data value that is the length of the data to be stored for the new task.

MINUTES(*data-value*)

specifies as a fullword binary value the number of minutes for use in conjunction with `AFTER` or `AT`. The value must be in the range 0 through 59 if `HOURS` or `SECONDS` is also specified, or in the range 0 through 5999 otherwise. This is a suboption of the `AFTER` and `AT` options. For its use and meaning, see the `AFTER` option.

NOCHECK

specifies that, for a remote system, CICS should improve performance of the `START` command by providing less error checking and slightly less function.

For more information, see the section about improving the performance of intersystem START requests in the *CICS for iSeries Intercommunication* book.

PROTECT

specifies that the new task is not started until the starting task has taken a syncpoint. If the starting task abends before the syncpoint is taken, the request to start the new task is canceled. The PROTECT option also causes the start request entry to be held on the TS/TD recoverable file. If the REQID option is also specified, the request identifier should be a name defined as recoverable to temporary storage. If the started transaction is remote, PROTECT specifies that it must not be scheduled until the local transaction has successfully completed a syncpoint. For more information about the PROTECT option with remote transactions, see the *CICS for iSeries Intercommunication* book name defined as recoverable to temporary storage.

QUEUE(name)

specifies the name (1–8 characters) of a temporary storage queue that may be used by the started transaction.

If you are also specifying REQID, make sure that the name of the REQID and the name of the QUEUE are not the same.

REQID(name)

specifies a name (1–8 characters), which must be unique, to identify a command. This option can be used when another task is to be provided with the capability of canceling an unexpired command.

If this option is omitted, CICS generates a unique request identifier in the EIBREQID field of the EXEC interface block, unless the NOCHECK option is specified, in which case field EIBREQID is set to nulls and cannot be used subsequently to cancel the START command.

If you include any of the data options (FROM, RTERMID, RTRANSID or QUEUE), the data is stored in a TS queue using the REQID name specified (or CICS generated) as the identifier. The temporary storage queue must be a local queue on the CICS system where the command is processed, and if the start is protected (using the PROTECT option), it should be defined as recoverable in the TST on that system. The START command is processed on the system identified by the SYSID option or, if the SYSID option is omitted, on the system associated with the TRANSID option. If the same REQID name is specified on multiple START commands (that is, it is not unique) there may be unexpected consequences such as:

- Multiple commands are established which can only be canceled in 'time expiry' order, each requiring a separate CANCEL command.
- The data (if any) from each START command is added to the same temporary storage queue in the order that the START commands are issued rather than in 'time expiry' order, possibly resulting in a mismatch between the START command and its data.

RTERMID(name)

specifies a value (1–4 characters), for example a terminal name, that may be retrieved when the transaction, specified in the TRANSID option in the START command, is started.

When retrieved, the value may be used in the TERMID option of a subsequent START command.

RTRANSID(*name*)

specifies a value (1–4 characters), for example a transaction name, that may be retrieved when the transaction, specified in the TRANSID option in the START command, is started.

When retrieved, the value may be used in the TRANSID option of a subsequent START command.

SECONDS(*data-value*)

specifies a fullword binary value in the range 0–59, when HOURS or MINUTES are also specified, or 0–359 999 when SECONDS is the only option specified. This is a suboption of the AFTER and AT options. For its use and meaning, see the AFTER option.

SYSID(*systemname*)

specifies the name of the system to which the request is directed.

TERMID(*name*)

specifies the symbolic identifier (1–4 alphanumeric characters) of the principal facility associated with a transaction to be started as a result of a START command. This principal facility can be either a terminal (the usual case) or an APPC session. Where an APPC session is specified, the connection (or modeset) name is used instead of a terminal identifier. This option is required when the transaction to be started must communicate with a terminal; it should be omitted otherwise.

The terminal identifier must be defined as either a local or a remote terminal on the system in which the START command is executed, *when the start of the transaction takes effect*.

TIME(*hhmmss*)

specifies the time when a new task should be started.

For compatibility between CICS platforms, it is recommended that the INTERVAL option is not used in ILE C programs. You should use the AFTER option instead.

TRANSID(*name*)

specifies the symbolic identifier of the transaction to be executed by a task started as the result of a START command. The name can be up to 4 characters long and must have been defined in the program control table (PCT) unless the SYSID option specifies a remote system.

If a nonlocal SYSID is specified, the transaction is assumed to be on a remote system irrespective of whether the name is defined in the local PCT. Otherwise, the PCT entry is used to determine whether the transaction is on a local or a remote system.

Exception Conditions

INVREQ

RESP2 values:

- 4 Hours out of range.
- 5 Minutes out of range.
- 6 Seconds out of range.

also occurs (RESP2 not set) in any of the following situations:

- The START command is not valid for processing by CICS.

Default action: terminate the task abnormally.

IOERR

occurs in any of the following situations:

- An input/output error occurred during a START operation.
- A START operation attempts to write to a queue when the queue is full.

Default action: terminate the task abnormally.

ISCINVREQ

occurs when the remote system indicates a failure that does not correspond to a known condition.

Default action: terminate the task abnormally.

LENGERR

occurs if LENGTH is not greater than zero.

Default action: terminate the task abnormally.

NOTAUTH

occurs in any of the following situations:

- The user is not authorized to use the temporary storage auxiliary file.

Default action: terminate the task abnormally.

SYSIDERR

occurs if the SYSID option specifies either a name that is not defined in the terminal control system table (TCS), or a system to which the link is closed.

Default action: terminate the task abnormally.

TERMIDERR

occurs if the terminal identifier in a START command cannot be found in the terminal control table.

Default action: terminate the task abnormally.

TRANSIDERR

occurs if the transaction identifier specified in a START command cannot be found in the program control table.

Default action: terminate the task abnormally.

STARTBR

Specify where in a file you want a browse to start.

Options

EQUAL

specifies that the search is satisfied only by a record having the same key (complete or generic) as that specified in the RIDFLD option. EQUAL is the default for an ESDS browse.

FILE(*name*)

specifies the name of the file to be accessed. The name must be alphanumeric, up to 8 characters long, and must have been defined in the file control table (FCT) unless the SYSID option specifies a remote system.

If a nonlocal SYSID is specified, the underlying file is assumed to be on a remote system irrespective of whether the name is defined in the local FCT. Otherwise, the FCT entry is used to determine whether the underlying file is on a local or a remote system.

GENERIC

specifies that the search key is a generic key whose length is specified in the KEYLENGTH option. Use this option only with a KSDS or a path over a KSDS or ESDS. The search for a record is satisfied when a record is found that has the same starting characters (generic key) as those specified.

GTEQ

specifies that if the search for a record having the same key (complete or generic) as that specified in the RIDFLD option is unsuccessful, the first record having a greater key satisfies the search. Use this option only with a KSDS or a path over a KSDS or ESDS. GTEQ is the default for a KSDS browse.

KEYLENGTH(*data-value*)

specifies as a halfword binary value the length of the key supplied in the RIDFLD option. If a specified KEYLENGTH value differs from the length defined for the underlying file and the operation is not generic, the INVREQ condition occurs.

INVREQ also occurs if you specify GENERIC, and the KEYLENGTH value is not less than that defined for the file.

If KEYLENGTH(0) is used with the object of positioning on the first record in the file, the GTEQ option must also be specified; otherwise, the NOTFND condition may occur.

RBA

specifies that the record identification field specified in the RIDFLD option contains a relative byte address. Use this option only when browsing an ESDS directly. The RIDFLD value can be from 0 upward.

REQID(*data-value*)

specifies as a halfword binary value a unique request identifier for the browse; it is used to control multiple browse operations on a file. If this option is not specified, a default value of zero is assumed.

RIDFLD(*data-area*)

specifies the record identification field. The contents can be a key, a relative byte address, or a relative record number. For a relative byte address or a relative record number, the format of this field must be fullword binary.

RRN

specifies that the record identification field specified in the RIDFLD option contains a relative record number. Use this option only when browsing an RRDS. The RIDFLD value can be from 1 upward.

SYSID(*name*)

specifies the name of the system to which the request is directed. The name can be up to 4 characters long.

If you specify *SYSID*, and omit both *RBA* and *RRN*, you must also specify *KEYLENGTH*; it cannot be found in the *FCT*.

Exception Conditions

Note: *RESP2* values are not set for files that are on remote systems.

DISABLED

RESP2 values:

- 50** A file is disabled. A file may be disabled because:
- It was initially defined as disabled and has not since been enabled.
 - It has been disabled by an EXEC CICS SET FILE command.
 - It has been disabled by the CEMT transaction.

Default action: Terminate the task abnormally.

FILENOTFOUND

RESP2 values:

- 1** The name specified in the *FILE* option cannot be found in the *FCT*.

Default action: Terminate the task abnormally.

ILLOGIC

RESP2 values:

- 110** There is an error that does not fall within one of the other CICS response categories.

(Further information is available in the *EIBRCODE* field; refer to Appendix A, "EXEC interface block," on page 529 for details.)

Default action: Terminate the task abnormally.

INVREQ

RESP2 values:

- 20** Browse operations are not allowed according to the file entry specification in the *FCT*.
- 25** The *KEYLENGTH* and *GENERIC* options are specified, and the length specified in the *KEYLENGTH* option is greater than or equal to the length of a full key.
- 26** The *KEYLENGTH* option is specified (but the *GENERIC* option is not specified), and the specified length differs from the length defined for the underlying file.
- 33** An attempt is made to start a browse with a *REQID* already in use for another browse.
- 42** The *KEYLENGTH* and *GENERIC* options are specified, and the length specified in the *KEYLENGTH* option is less than zero.

Default action: Terminate the task abnormally.

IOERR

RESP2 values:

- 120** There is an I/O error during the file control operation. An I/O error is any unusual event that is not covered by a CICS exception condition.
- (Further information is available in the EIBRCODE field; refer to Appendix A, "EXEC interface block," on page 529 for details.)

Default action: Terminate the task abnormally.

ISCINVREQ

RESP2 values:

- 70** The remote system indicates a failure that does not correspond to a known condition.

Default action: Terminate the task abnormally.

NOTAUTH

RESP2 values:

- 101** A resource security check has failed on FILE(*name*).

Default action: Terminate the task abnormally.

NOTFND

RESP2 values:

- 80** An attempt to position on a record based on the search argument provided is unsuccessful.
- NOTFND can also occur if a generic STARTBR with KEYLENGTH(0) specifies the EQUAL option.

Default action: Terminate the task abnormally.

NOTOPEN

RESP2 values:

- 60** One of the following has occurred:
- The requested file is CLOSED and UNENABLED. The CLOSED, UNENABLED state is reached after a CLOSE request has been received against an OPEN ENABLED file and the file is no longer in use.
 - The requested file is OPEN and ENABLED and in use by other transactions, but a CLOSE request against the file has been received.

This condition does not occur if the request is made to either a CLOSED, ENABLED file or a CLOSED, DISABLED file. In the first case, the file is opened as part of executing the request. In the second case, the DISABLED condition occurs.

Default action: Terminate the task abnormally.

SYSIDERR

RESP2 values:

- 130** The SYSID option specifies either a name that is not defined in the terminal control system table (TCS), or a system to which the link is closed.

Default action: Terminate the task abnormally.

SUSPEND

Suspend a task.

(1)

▶▶—SUSPEND—▶▶

Notes:

1 Command is accepted but ignored by CICS/400.

Description

On other CICS platforms, SUSPEND relinquishes control to a task of higher or equal dispatching priority. Control is returned to the task issuing the command as soon as no other task of a higher or equal priority is ready to be processed.

Note: In CICS/400 this is a no-operation. It is retained for portability with other CICS platforms.

SYNCPOINT

Establish a syncpoint.

▶▶—SYNCPOINT—▶▶

Conditions: INVREQ, ROLLEDBACK

Description

SYNCPOINT divides a task (usually a long-running one) into smaller logical units of work. It specifies that all changes to recoverable resources made by the task since its last syncpoint are to be committed.

See Chapter 8, "Recovery considerations," on page 105 for further information.

Exception Conditions

INVREQ

RESP2 values:

200 The SYNCPOINT command is issued in a DPL server program and SYNCONRETURN was not specified on the corresponding LINK command, or the SYNCPOINT command is issued from a program defined with APISET(DPLSUBSET) in the PPT.

Default action: terminate the task abnormally.

ROLLEDBACK

occurs when a SYNCPOINT command is driven into rollback by a remote system that is unable to commit the syncpoint. All changes made to recoverable resources in the current logical unit of work are backed out.

Default action: terminate the task abnormally.

SYNCPOINT ROLLBACK

Back out to last syncpoint.

▶▶—SYNCPOINT ROLLBACK—◀◀

Condition: INVREQ

Description

All changes to recoverable resources made by the task since its last syncpoint are backed out.

This command can be used, for example, to tidy up in a HANDLE ABEND routine, or to revoke database changes after the application program finds irrecoverable errors in its input data.

If the LUW updates remote recoverable resources using an APPC session, the rollback is propagated to the back-end transaction.

When a distributed transaction processing conversation is in use, the remote application program has the EIB fields EIBSYNRB, EIBERR, and EIBERRCD set. For the conversation to continue, the remote application program must execute a SYNCPOINT ROLLBACK command.

When the mirror transaction is involved in the LUW using an APPC session, the mirror honors the rollback request, revokes changes, and then terminates normally.

See Chapter 8, “Recovery considerations,” on page 105 for further information.

Exception Conditions

INVREQ

RESP2 value:

200 The SYNCPOINT command is issued in a DPL server program and SYNCONRETURN was not specified on the corresponding LINK command, or the SYNCPOINT command is issued from a program defined with APISET(DPLSUBSET) in the PPT.

Default action: terminate the task abnormally.

UNLOCK

Release exclusive control.

▶▶—UNLOCK—FILE—(—name—)⁽¹⁾—[SYSID—(—name—)]—◀◀

Notes:

1 DATASET is also accepted, but FILE is the preferred term (see “DATASET option” on page 311).

Conditions: DISABLED, FILENOTFOUND, ILLOGIC, IOERR, ISCINVREQ, NOTAUTH, NOTOPEN, SYSIDERR

Description

UNLOCK releases the exclusive control established in response to a READ command with the UPDATE option. You use it if you retrieve a record for update, and decide that you do not want to update the record after all. For a recoverable file, however, the resource remains locked until either a SYNCPOINT command is executed or the task is terminated. The file may be on a local or a remote system.

You can also use this command to terminate a WRITE MASSINSERT (emulated VSAM) operation that has been function shipped to a non-CICS/400 system.

Options

FILE(*name*)

specifies the name of the file to be released. The name must be alphanumeric, up to 8 characters long, and must have been defined in the file control table (FCT) unless the SYSID option specifies a remote system.

If a nonlocal SYSID is specified, the underlying file is assumed to be on a remote system irrespective of whether the name is defined in the local FCT. Otherwise, the FCT entry is used to determine whether the underlying file is on a local or a remote system.

SYSID(*name*)

specifies the name of the system to which the request is directed. The name can be up to 4 characters long.

Exception Conditions

Note: RESP2 values are not set for files that are on remote systems.

DISABLED

RESP2 values:

- 50** A file is disabled. A file may be disabled because:
- It was initially defined as disabled and has not since been enabled.
 - It has been disabled by an EXEC CICS SET FILE command.
 - It has been disabled by the CEMT transaction.

This condition cannot occur when the UNLOCK follows a successful READ UPDATE command or a WRITE MASSINSERT (emulated VSAM) operation.

Default action: Terminate the task abnormally.

FILENOTFOUND

RESP2 values:

- 1** The name specified in the FILE option cannot be found in the FCT.

Default action: Terminate the task abnormally.

ILLOGIC

RESP2 values:

- 110** There is an error that does not fall within one of the other CICS response categories.

(Further information is available in the EIBRCODE field; refer to Appendix A, "EXEC interface block," on page 529 for details.)

Default action: Terminate the task abnormally.

IOERR

RESP2 values:

120 There is an I/O error during the file control operation. An I/O error is any unusual event that is not covered by a CICS exception condition.

(Further information is available in the EIBRCODE field; refer to Appendix A, "EXEC interface block," on page 529 for details.)

Default action: Terminate the task abnormally.

ISCINVREQ

RESP2 values:

70 The remote system indicates a failure that does not correspond to a known condition.

Default action: Terminate the task abnormally.

NOTAUTH

RESP2 values:

101 A resource security check has failed on FILE(*name*).

Default action: Terminate the task abnormally.

NOTOPEN

RESP2 values:

- 60** One of the following has occurred:
- The requested file is CLOSED and UNENABLED. The CLOSED, UNENABLED state is reached after a CLOSE request has been received against an OPEN ENABLED file and the file is no longer in use.
 - The requested file is OPEN and UNENABLED and in use by other transactions, but a CLOSE request against the file has been received.

This condition does not occur if the request is made to either a CLOSED, ENABLED file or a CLOSED, DISABLED file. In the first case, the file is opened as part of executing the request. In the second case, the DISABLED condition occurs.

It also cannot occur when the UNLOCK follows a successful READ UPDATE command or a WRITE MASSINSERT (emulated VSAM) operation.

Default action: Terminate the task abnormally.

SYSIDERR

RESP2 values:

130 The SYSID option specifies either a name that is not defined in the terminal control system table (TCS), or a system to which the link is closed.

Default action: Terminate the task abnormally.

WAIT CONVID

Ensure accumulated data is transmitted on an APPC conversation.

►►—WAIT CONVID—(—*name*—)———
 └—STATE—(—*cvda*—)—┘———►►

Conditions: INVREQ, NOTALLOC

Description

WAIT CONVID allows an application program to ensure that any accumulated application data and control indicators from a SEND command or the results of a CONNECT PROCESS command are transmitted to the partner transaction.

See part 3 of *CICS/400 Intercommunication* for more information.

Options

CONVID(*name*)

identifies the conversation to which the command relates. The 4-character name identifies the token returned by a previously executed ALLOCATE command in the EIBRSRCE field of the EIB.

STATE(*cvda*)

gets the state of the current conversation. For a complete list of the CVDA values that can be returned on APPC commands and for information about receiving and testing these values, see “CICS-value data areas (CVDA)” on page 309.

Exception Conditions

INVREQ

RESP2 values:

- The CONVID value was obtained by an ASSIGN FACILITY command. However, the principal facility is *not* an APPC conversation.

200 The command is issued in a DPL server program and refers to the principal facility.

Default action: Terminate the task abnormally.

NOTALLOC

occurs if the specified CONVID value does not relate to a conversation owned by the application.

Default action: Terminate the task abnormally.

WAIT EVENT

Wait for an event to occur.

►►—WAIT EVENT—ECADDR—(—*ptr-value*—)———►►

Condition: INVREQ

Description

WAIT EVENT synchronizes a task with the completion of an event initiated by the same task or by another task. The event would normally be the posting, at the expiration time, of a timer-event control area provided in response to a POST command, as described in "POST" on page 387. The WAIT EVENT command provides a method of directly relinquishing control to some other task until the event being waited on is completed.

See Chapter 18, "Interval control," on page 193 for more information.

Options

ECADDR(*ptr-value*)

specifies the address of the timer-event control area that must be posted before task activity can be resumed.

Exception Conditions

INVREQ

RESP2 values:

- 0 A task is already waiting for this event.
- 2 The address specified for the timer event control area is zero or null.

Default action: terminate the task abnormally.

Examples

The following example shows you how to suspend the processing of a task until the specified event control area is posted:

```
EXEC CICS WAIT EVENT ECADDR(PVALUE)
```

WAIT JOURNALNUM

Synchronize with journal output.

```
▶▶—| WAIT JOURNALNUM—(—data-value—)|——| WAIT JOURNAL JFILEID—(—data-value—)|——| REQID—(—data-value—)|——| STARTIO—|——▶▶
```

Description

On other CICS platforms, WAIT JOURNALNUM synchronizes the task with the output of one or more journal records that have been created but whose output has been deferred; that is, with asynchronous journal output requests.

Notes:

1. This is a no-operation for CICS/400. It is retained for portability with other CICS platforms.
2. The WAIT JOURNAL command, with the JFILEID option in place of JOURNALNUM but with all other options the same as on the WAIT JOURNALNUM command, is supported for compatibility purposes.

Options

JOURNALNUM(*data-value*)

specifies as a halfword binary value in the range 1 through 99 the number to be taken as the journal identifier.

REQID(*data-value*)

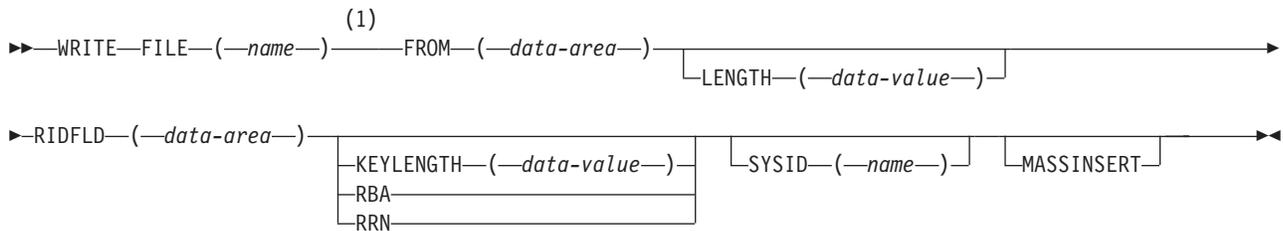
specifies as a fullword binary value a number that identifies the journal record that has been created but possibly not yet written out. If REQID is not specified, the task is synchronized with the output of the last record created for the journal specified by JOURNALNUM.

STARTIO

specifies that output of the journal record is to be initiated immediately.

WRITE

Write a record.



Notes:

- 1 DATASET is also accepted, but FILE is the preferred term (see “DATASET option” on page 311).

Conditions: DISABLED, DUPREC, FILENOTFOUND, ILLOGIC, INVREQ, IOERR, ISCINVREQ, LENGERR, NOSPACE, NOTAUTH, NOTOPEN, SYSIDERR

Description

The WRITE command writes a new record to a file on a local or a remote system. For an emulated VSAM ESDS, the record is always added at the end of the file. CICS/400 does not use the identification field specified in RIDFLD when calculating the RBA of the new record, but the new RBA is returned to the application in the record identification field specified in the RIDFLD option; that is, when RBA is specified, RIDFLD is an output field only.

For an emulated VSAM KSDS, the record is added in the logical location specified by the associated key; this location may be anywhere in the file.

Records for an ESDS or a KSDS can be either fixed-length or variable-length; those for an RRDS must be fixed-length. MASSINSERT operations must proceed with ascending keys, and (when function shipped to a non-CICS/400 system) must be terminated by an UNLOCK before any other request to the same file.

See Chapter 10, “File control,” on page 115 for more information.

Options

FILE(*name*)

specifies the name of the file to be accessed. The name must be alphanumeric,

up to 8 characters long, and must have been defined in the file control table (FCT) unless the SYSID option specifies a remote system.

If a nonlocal SYSID is specified, the underlying file is assumed to be on a remote system irrespective of whether the name is defined in the local FCT. Otherwise, the FCT entry is used to determine whether the underlying file is on a local or a remote system.

FROM(*data-area*)

specifies the record to be written to the file.

KEYLENGTH(*data-value*)

specifies as a halfword binary value the length of the key supplied in the RIDFLD option. You must code KEYLENGTH if you are also using SYSID, unless you are using RBA or RRN.

If a specified KEYLENGTH value differs from the length defined for the underlying file, the INVREQ condition occurs.

LENGTH(*data-value*)

specifies as a halfword binary value the length of the record to be written.

If SYSID is specified, LENGTH must be coded. You must also specify this option for a file defined as containing variable-length records. It need not be specified if the file contains fixed-length records, but its inclusion is recommended because it causes a check to be made to ensure that the length of data being written is equal to that defined for the file. If the lengths are not equal, the LENGERR condition occurs.

MASSINSERT

specifies that the WRITE command is part of a mass-insert operation, that is, a series of WRITE commands each specifying MASSINSERT. This option is passed to remote systems, but is a no-operation within CICS/400.

Use of the MASSINSERT option also prevents a nonrecoverable file from being closed in response to a SET FILE CLOSED command, until either an UNLOCK command is issued or a syncpoint is taken.

RBA

specifies that the record identification field specified in the RIDFLD option contains a relative byte address. Use this option only when writing directly to an ESDS. On completion of the WRITE command, the RIDFLD data area contains the relative byte address of the record that was written.

RIDFLD(*data-area*)

specifies the record identification field. The contents can be a key, a relative byte address, or a relative record number. For a relative byte address or a relative record number, the format of this field must be fullword binary.

OS/400 keyed files work with keys embedded in the data record, and CICS/400 makes no use of RIDFLD when writing to keyed files. However, if compatibility with VSAM KSDS files is important, RIDFLD must reference a duplicate of the key that is expressed in the data record.

RRN

specifies that the record identification field specified in the RIDFLD option contains a relative record number. Use this option only when writing to an RRDS. The RIDFLD value can be from 1 upward.

SYSID(*name*)

specifies the name of the system to which the request is directed. The name can be up to 4 characters long.

If you specify `SYSID`, and omit both `RBA` and `RRN`, you must also specify `LENGTH` and `KEYLENGTH`; they cannot be found in the `FCT`.

Exception Conditions

Note: `RESP2` values are not set for files that are on remote systems.

DISABLED

`RESP2` values:

- 50** A file is disabled. A file may be disabled because:
- It was initially defined as disabled and has not since been enabled.
 - It has been disabled by an `EXEC CICS SET FILE` command.
 - It has been disabled by the `CEMT` transaction.

Default action: Terminate the task abnormally.

DUPREC

`RESP2` values:

- 150** The record to be written contains a key that is the same as the key of a record already in the file, and the file accepts only unique keys. The key may be a primary key of a `KSDS` or an alternate key of a logical view of a file.

Default action: Terminate the task abnormally.

FILENOTFOUND

`RESP2` values:

- 1** The name specified in the `FILE` option cannot be found in the `FCT`.

Default action: Terminate the task abnormally.

ILLOGIC

`RESP2` values:

- 110** There is an error that does not fall within one of the other `CICS` response categories.

(Further information is available in the `EIBRCODE` field; refer to Appendix A, "EXEC interface block," on page 529 for details.)

Default action: Terminate the task abnormally.

INVREQ

`RESP2` values:

- 20** Add operations are not allowed according to the file entry specification in the `FCT`.
- 26** The `KEYLENGTH` option is specified, and the specified length differs from the length defined for the underlying file.
- 29** Following a `READ UPDATE` command for a file, a `WRITE` command is issued for a file referring to the same underlying file before exclusive control is released by a `REWRITE`, `UNLOCK`, or `DELETE` command.

Default action: Terminate the task abnormally.

IOERR

`RESP2` values:

120 There is an I/O error during the file control operation. An I/O error is any unusual event that is not covered by a CICS exception condition.

(Further information is available in the EIBRCODE field; refer to Appendix A, "EXEC interface block," on page 529 for details.)

Default action: Terminate the task abnormally.

ISCINVREQ

RESP2 values:

70 The remote system indicates a failure that does not correspond to a known condition.

Default action: Terminate the task abnormally.

LENGERR

RESP2 values:

12 The length specified for the write operation exceeds the maximum record size; the record is truncated.

14 An incorrect length is specified for a write operation involving fixed-length records.

Default action: Terminate the task abnormally.

NOSPACE

RESP2 values:

100 No space is available on the direct access device for adding records to a file.

Default action: Terminate the task abnormally.

NOTAUTH

RESP2 values:

101 A resource security check has failed on FILE(*name*).

Default action: Terminate the task abnormally.

NOTOPEN

RESP2 values:

60 One of the following has occurred:

- The requested file is CLOSED and UNENABLED. The CLOSED, UNENABLED state is reached after a close request has been received against an OPEN ENABLED file and the file is no longer in use.
- The requested file is OPEN and UNENABLED and in use by other transactions, but a close request against the file has been received.

This condition does not occur if the request is made to either a CLOSED, ENABLED file or a CLOSED, DISABLED file. In the first case, the file is opened as part of executing the request. In the second case, the DISABLED condition occurs.

Default action: Terminate the task abnormally.

SYSIDERR

RESP2 values:

130 The SYSID option specifies either a name that is not defined in the terminal control system table (TCS), or a system to which the link is closed.

Default action: Terminate the task abnormally.

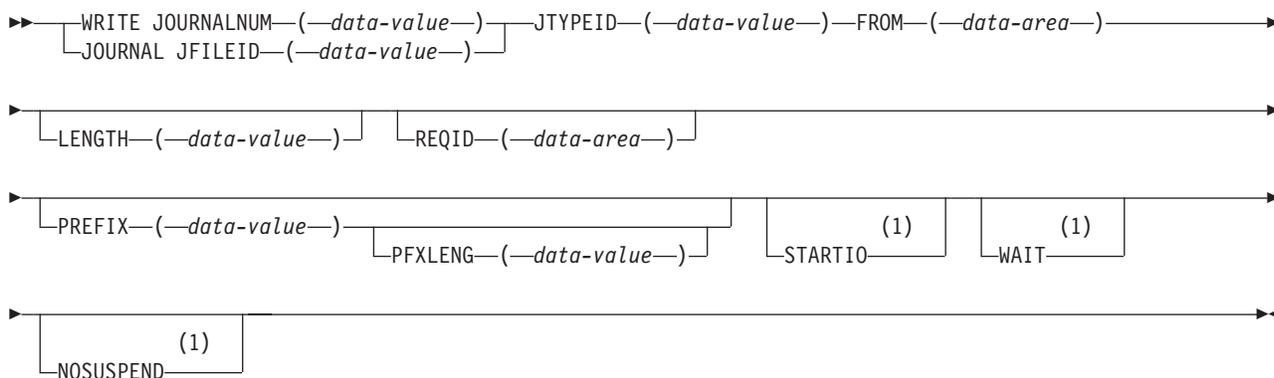
Examples

For example:

```
EXEC CICS WRITE
      FROM(RECORD)
      LENGTH(DATLEN)
      FILE('MASTER')
      RIDFLD(KEYFLD)
      :
```

WRITE JOURNALNUM

Create a journal record.



Notes:

1 Ignored by CICS/400.

Conditions: IOERR, JIDERR, LENGERR, NOJBUFSP, NOTAUTH, NOTOPEN

Description

`WRITE JOURNALNUM` creates a journal record.

Note: The `JOURNAL` command, with the `JFILEID` option in place of `JOURNALNUM` but with all other options the same as on the `WRITE JOURNALNUM` command, is supported for compatibility purposes.

Within CICS/400, the request is a synchronous output request. The `WAIT`, `STARTIO`, and `NOSUSPEND` options are accepted by the API translator, but are ignored at execution time.

See Chapter 8, "Recovery considerations," on page 105 for more information about journaling.

Options

FROM(*data-area*)

specifies the user data to be built into the journal record.

JOURNALNUM(*data-value*)

specifies as a halfword binary value in the range 1 through 99 the number to be taken as the journal identifier (as used in the JCT.)

JTYPEID(*data-value*)

specifies a 2-character identifier to be placed in the journal record to identify its origin.

LENGTH(*data-value*)

specifies as a halfword binary value the length in bytes of the user data to be built into the journal record. The minimum value is 0, and the maximum value is (buffer size–72) minus PFXLENG.

NOSUSPEND

This option is accepted but is ignored by CICS/400.

PFXLENG(*data-value*)

specifies as a halfword binary value the length in bytes of the user prefix data to be included in the journal record. The minimum value is 0 and the maximum value is (buffer size–72) minus LENGTH.

PREFIX(*data-value*)

specifies the user prefix data to be included in the journal record.

REQID(*data-area*)

specifies a fullword binary field to receive a number that identifies the journal record being created by its position in the journal.

STARTIO

This option is accepted but is ignored by CICS/400.

WAIT

This option is accepted but is ignored by CICS/400.

Exception Conditions

IOERR

occurs if the physical output of a journal record was not accomplished because of an irrecoverable I/O error.

Default action: Terminate the task abnormally.

JIDERR

occurs if the specified journal identifier does not exist in the journal control table (JCT).

Default action: Terminate the task abnormally.

LENGERR

occurs if the computed length for the journal record exceeds the total buffer space allocated for the journal, as specified in the JCT entry for the journal, or if the length specified for the prefix or for the data is negative.

Default action: Terminate the task abnormally.

NOJBUFSP

occurs if the journal file is full and the journal is defined as nonswitchable.

Default action: Suspend task activity until the journal request can be satisfied. CICS ensures that both buffers are written out to auxiliary storage, thus freeing them for new records.

NOTAUTH

occurs if a resource security check has failed on JOURNALNUM(*data-value*).

Default action: Terminate the task abnormally.

NOTOPEN

occurs if the journal command cannot be satisfied because the specified journal is not open.

Default action: Terminate the task abnormally.

Examples

The following example shows how to request journal output:

```
EXEC CICS WRITE
      JOURNALNUM(2)
      JTYPEID('XX')
      FROM(KEYDATA)
      LENGTH(8)
      PREFIX(PROGNAME)
      PFXLENG(6)
      :
```

WRITEQ TD

Write data to a transient data queue.

```
▶▶ WRITEQ TD QUEUE(—name—) FROM(—data-area—)
   └── LENGTH(—data-value—) ───▶
▶ └── SYSID(—name—) ───▶
```

Conditions: DISABLED, INVREQ, IOERR, ISCINVREQ, LENGERR, NOSPACE, NOTAUTH, NOTOPEN, QIDERR, SYSIDERR

Description

The WRITEQ TD command writes data to a transient data queue (predefined symbolic destination).

The data must not contain any pointers.

See Chapter 23, “Transient data control,” on page 215 for more information.

Options

FROM(*data-area*)

specifies the data to be written to the transient data queue.

LENGTH(*data-value*)

specifies as a halfword binary value the length of the data to be written. For a description of a safe upper limit, see “LENGTH options” on page 312.

QUEUE(*name*)

specifies the symbolic name of the queue to be written to. The name must be alphanumeric, up to 4 characters long, and must have been defined in the destination control table (DCT) unless the SYSID option specifies a remote system.

If a nonlocal SYSID is specified, the queue is assumed to be on a remote system irrespective of whether the name is defined in the local DCT. Otherwise, the DCT entry is used to determine whether the queue is on a local or a remote system.

SYSID(*name*)

specifies the name of the system to which the request is directed. The name can be up to 4 characters long.

Exception Conditions

DISABLED

occurs if the queue has been disabled.

Default action: Terminate the task abnormally.

INVREQ

occurs if WRITEQ TD names an extrapartition queue that has been opened for input.

Default action: Terminate the task abnormally.

IOERR

occurs if there is an I/O error during the transient data operation.

Default action: Terminate the task abnormally.

ISCINVREQ

occurs if the remote system indicates a failure that does not correspond to a known condition.

Default action: Terminate the task abnormally.

LENGERR

occurs in any of the following situations:

- WRITEQ TD names an extrapartition queue and does not specify a length consistent with the defined file.
- WRITEQ TD names an intrapartition queue and does not specify a length consistent with that defined for the intrapartition physical file.

Default action: Terminate the task abnormally.

NOSPACE

occurs if no more space exists on the queue. When this happens, no more data should be written to the queue because it may be lost.

Default action: Terminate the task abnormally.

NOTAUTH

occurs if a resource security check has failed on QUEUE(*name*).

Default action: Terminate the task abnormally.

NOTOPEN

occurs if the destination is closed. This condition applies to extrapartition queues only.

Default action: Terminate the task abnormally.

QIDERR

occurs if the symbolic destination to be used with a transient data control command cannot be found.

Default action: Terminate the task abnormally.

SYSIDERR

occurs if the SYSID option specifies either a name that is not defined in the terminal control system table (TCS), or a system to which the link is closed.

Default action: Terminate the task abnormally.

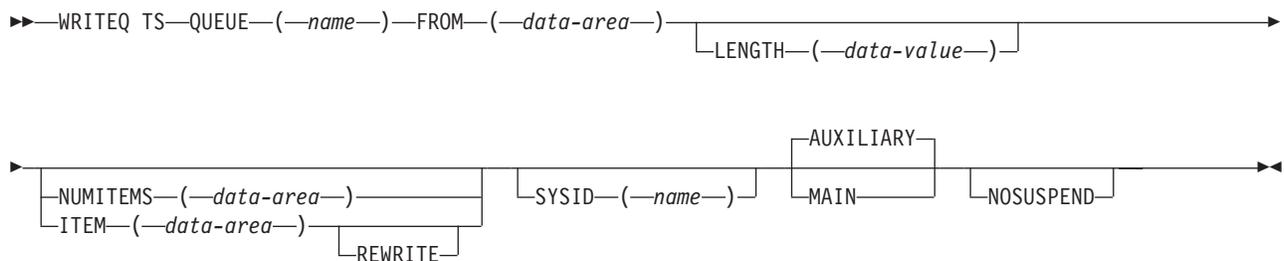
Examples

The following example shows how to write data to a predefined symbolic destination:

```
EXEC CICS WRITEQ TD
      QUEUE('ABCD')
      FROM(MESSAGE)
      LENGTH(LENG)
      :
```

WRITEQ TS

Store temporary data in a temporary storage queue.



Conditions: INVREQ, IOERR, ISCINVREQ, ITEMERR, LENGERR, NOSPACE, NOTAUTH, QIDERR, SYSIDERR

Description

The WRITEQ TS command stores temporary data (records) in a temporary storage queue in main or auxiliary storage.

If a queue has been defined as recoverable, the program must not issue a WRITEQ TS if a DELETEQ TS has previously been issued within the same logical unit of work. Thus, following a DELETEQ TS, no WRITEQ TS can be issued until after a syncpoint has occurred.

Data written to an auxiliary temporary storage queue must not contain pointers.

Note: There is a maximum number of uniquely named auxiliary temporary storage queues available for any CICS system. This is determined by the size of the auxiliary temporary storage queue physical file as defined by your system administrator.

See Chapter 24, “Temporary storage control,” on page 219 for more information.

Options

AUXILIARY

specifies that the temporary storage queue is on a direct-access storage device in auxiliary storage.

This is the default for the first write and is ignored for an existing queue.

FROM(*data-area*)

specifies the data to be written to temporary storage.

ITEM(*data-area*)

specifies a halfword binary field containing the number of the item to be replaced in the queue (REWRITE option also specified).

Note: In earlier versions of CICS on other platforms, ITEM on a WRITEQ TS without REWRITE would perform a similar function to NUMITEMS. This function is retained for compatibility.

LENGTH(*data-value*)

specifies as a halfword binary value the length of the data to be written. For a description of a safe upper limit, see "LENGTH options" on page 312.

For AUXILIARY, upper bound is determined by the length of the underlying TS/TD physical file.

MAIN

specifies that the temporary storage queue is in main storage.

If you use the MAIN option to write data to a temporary storage queue on a remote system, the data is stored in main storage. If this condition is not met, the data is stored in auxiliary storage.

This option is ignored for an existing queue.

NOSUSPEND

specifies that application program suspension for the NOSPACE condition is to be inhibited.

This does not apply to a temporary storage queue in main storage.

NUMITEMS(*data-area*)

specifies a halfword binary field to receive a number indicating how many items there are in the queue after the WRITEQ TS command is executed.

If the record starts a new queue, the item number assigned is 1; subsequent item numbers follow on sequentially. NUMITEMS is not valid if REWRITE is specified.

QUEUE(*name*)

specifies the symbolic name of the queue to be written to. If the queue name appears in the TST, and the entry is marked as remote, the request is shipped to a remote system. The name must be alphanumeric, 1–8 characters long, and unique within the CICS system. Do not use X'FA' through X'FF' as the first character of the name; these characters are reserved for CICS use. The name cannot consist solely of binary zeros.

REWRITE

specifies that the existing record in the queue is to be overwritten with the data provided. If the REWRITE option is specified, the ITEM option must also be specified. If the specified queue does not exist, the QIDERR condition occurs. If the correct item within an existing queue cannot be found, the ITEMERR condition occurs and the data is not stored.

SYSID(*name*)

specifies the name of the system to which the request is directed. The name can be up to 4 characters long.

Exception Conditions

INVREQ

occurs in any of the following situations:

- A WRITEQ TS command specifies a queue name that consists solely of binary zeros.
- A WRITEQ TS command specifies a queue that is locked and awaiting ISC session recovery.
- The queue was created by CICS internal code.

Default action: Terminate the task abnormally.

IOERR

occurs if there is an irrecoverable input/output error.

Default action: Terminate the task abnormally.

ISCINVREQ

occurs if the remote system indicates a failure that does not correspond to a known condition.

Default action: Terminate the task abnormally.

ITEMERR

occurs in any of the following situations:

- The item number specified or implied by a WRITEQ TS command with the REWRITE option is not valid (that is, it is outside the range of entry numbers assigned for the queue).
- The maximum number of items (32 767) is exceeded.

Default action: Terminate the task abnormally.

LENGERR

occurs in any of the following situations:

- The length of the stored data is zero or negative.
- For AUXILIARY, the length is greater than that of the underlying TS/TD physical file.

Default action: Terminate the task abnormally.

NOSPACE

occurs if insufficient space is available in the temporary storage file to contain the data.

Default action: Suspend the task until space becomes available as it is released by other tasks; then return normally.

NOTAUTH

occurs if a resource security check has failed on QUEUE(*name*).

Default action: Terminate the task abnormally.

QIDERR

occurs if the queue specified by a WRITEQ TS command with the REWRITE option cannot be found.

Default action: Terminate the task abnormally.

SYSIDERR

occurs if the SYSID option specifies either a name that is not defined in the terminal control system table (TCS), or a system to which the link is closed.

Default action: Terminate the task abnormally.

Examples

The following example shows how to write a record to a temporary storage queue in auxiliary storage:

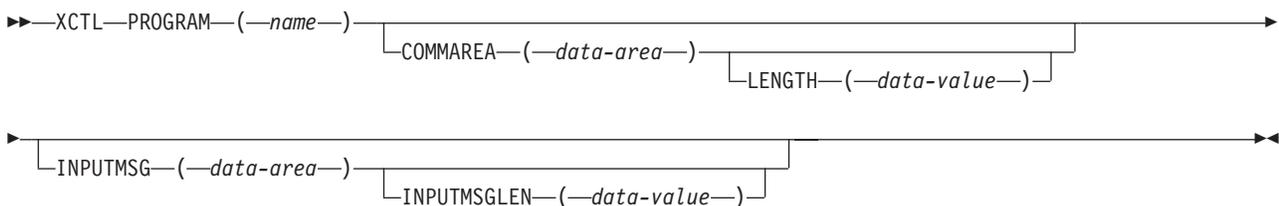
```
EXEC CICS WRITEQ TS
      QUEUE(UNIQNAME)
      FROM(MESSAGE)
      LENGTH(LENGTH)
      NUMITEMS(DREF)
      :
```

The following example shows how to update a record in a temporary storage queue in main storage:

```
EXEC CICS WRITEQ TS
      QUEUE('TEMPQ1')
      FROM(DATAFLD)
      LENGTH(40)
      ITEM(ITEMFLD)
      REWRITE
      MAIN
      :
```

XCTL

Transfer control from one application program to another.



Conditions: INVREQ, LENGERR, NOTAUTH, PGMIDERR

Description

XCTL transfers control from one application program to another at the same logical level.

See Chapter 20, "Program control," on page 199 for more information.

Options

COMMAREA(*data-area*)

specifies a communication area that is to be made available to the invoked

program. In this option, a pointer to the data area is passed. In an invoked COBOL program, you must give this data area the name DFHCOMMAREA. (See “COMMAREA in EXEC CICS LINK and EXEC CICS XCTL commands” on page 70 and “Passing data to other programs” on page 201.) In a C receiving program, this data area must be referenced by an EXEC CICS ADDRESS COMMAREA command.

INPUTMSG(*data-area*)

specifies data to be passed to the invoked program when it first issues an EXEC CICS RECEIVE command.

This data remains available until the execution of an EXEC CICS RECEIVE or EXEC CICS RETURN command. An invoked program can invoke a further program and so on, creating a chain of linked programs. If a linked-to chain exists, CICS supplies the INPUTMSG data to the first EXEC CICS RECEIVE command executed in the chain.

See Chapter 20, “Program control,” on page 199 for more information about using the INPUTMSG option.

INPUTMSGLEN(*data-value*)

specifies as a halfword binary value the length of the INPUTMSG data. If the value is negative, zero is assumed.

LENGTH(*data-value*)

specifies as a halfword binary value the length in bytes of the communication area. If a negative value is supplied, zero is assumed. For a description of a safe upper limit, see “LENGTH options” on page 312.

PROGRAM(*name*)

specifies the identifier of the program to which control is to be passed unconditionally. The name must be alphanumeric, up to 8 characters long, and must have been defined in the processing program table (PPT) as a local program.

Exception Conditions

INVREQ

RESP2 values:

- 8 The INPUTMSG option is supplied for a program that is not associated with a terminal, or that is associated with an APPC logical unit.
- 200 The INPUTMSG option is supplied in a program invoked by DPL.
- The INPUTMSG option is supplied with a null address.

Default action: Terminate the task abnormally.

LENGERR

RESP2 values:

- The length specified in the LENGTH option is greater than the length of the data area specified in the COMMAREA option, and while that data was being copied a destructive overlap occurred because of the incorrect length.
- The INPUTMSGLEN value is outside the range 1 through 32 767.
- 11 A negative LENGTH value is supplied.

Default action: Terminate the task abnormally.

NOTAUTH

occurs if a resource security check has failed on PROGRAM(name).

Default action: Terminate the task abnormally.

PGMIDERR

RESP2 values:

- 1 A program does not have an installed resource definition.
- 2 The program is disabled.
- 3 The program cannot be loaded.
- 9 The program is defined as remote.

Default action: Terminate the task abnormally.

Examples

The following example shows how to request a transfer of control to an application program called PROG2:

```
EXEC CICS XCTL PROGRAM('PROG2')  
:  
:
```

Chapter 33. System programming reference

This chapter contains details of the following EXEC CICS commands:

- “DISCARD commands”
- “INQUIRE commands” on page 480
- “PERFORM SHUTDOWN command” on page “PERFORM SHUTDOWN command” on page 509
- “SET commands” on page “SET commands” on page 509

See “System programming commands” on page 314 for information on specifying these commands.

DISCARD commands

The EXEC CICS DISCARD commands allow you to remove resources from the control region. The resources are not deleted, and will be reinstalled when the control region is restarted.

DISCARD AUTINSTMODEL

▶▶—DISCARD AUTINSTMODEL—(—*name*—)————▶▶

Conditions: INVREQ, MODELIDERR

The EXEC CICS DISCARD AUTINSTMODEL command removes the named CICS autoinstall terminal model definition from the CICS control region.

Options

AUTINSTMODEL(name)

is the name of the CICS autoinstalled terminal model as specified in the CICSDEV parameter within the OS/400 ADDCICSTCT CL command. The name may be up to 4 characters long.

Exception Conditions

INVREQ

occurs if:

- The name of the autoinstall model begins with “C” and cannot be discarded (RESP2=3).
- The named autoinstall model could not be discarded (RESP2=4).

Default action: Terminate the task abnormally.

MODELIDERR

occurs if the named autoinstall model cannot be found (RESP2=1).

Default action: Terminate the task abnormally.

DISCARD FILE

▶▶—DISCARD FILE—(—*name*—)————▶▶

Conditions: FILENOTFOUND, INVREQ

The EXEC CICS DISCARD FILE command removes the named CICS file definition from the CICS control region.

Options

FILE(name)

is the name of the CICS file as specified in the FILEID parameter within the OS/400 ADDCICSFCT CL command. The name may be up to 8 characters long.

Exception Conditions

FILENOTFOUND

occurs if the named file cannot be found (RESP2=18).

Default action: Terminate the task abnormally.

INVREQ

occurs if:

- The named file is defined as remote (RESP2=1)
- The named file is not closed (RESP2=2)
- The named file is not disabled (RESP2=3)
- The named file is currently in use (RESP2=25)
- The name of the file begins with AEG and cannot be discarded (RESP2=26)
- The file could not be discarded (RESP2=27)

Default action: Terminate the task abnormally.

DISCARD PROGRAM

▶▶—DISCARD PROGRAM—(—*name*—)————▶▶

Conditions: INVREQ, PGMIDERR

The EXEC CICS DISCARD PROGRAM command removes the named CICS program definition from the CICS control region.

Options

PROGRAM(name)

is the name of the CICS program or map set as specified in the PGMID parameter within the OS/400 ADDCICSPPT CL command. The name may be up to 8 characters long.

Exception Conditions

INVREQ

occurs if:

- The name of the program begins with AEG and cannot be discarded (RESP2=1)
- The program is not disabled. The program needs to be disabled for the discard process to work (RESP2=10)
- The program is currently in use (RESP2=11)
- The program is named in the program control table (PCT) (RESP2=12)
- The named program could not be discarded (RESP2=16)

Default action: Terminate the task abnormally.

PGMIDERR

occurs if the named program cannot be found (RESP2=7).

Default action: Terminate the task abnormally.

DISCARD TRANSACTION

►►—DISCARD TRANSACTION—(—*name*—)—————►►

Conditions: INVREQ, TRANSIDERR

The EXEC CICS DISCARD TRANSACTION command removes the named CICS transaction definition from the CICS control region.

Options

TRANSACTION(*name*)

is the name of the CICS transaction as specified in the TRANSID parameter within the OS/400 ADDCICSPCT CL command. The name may be up to 4 characters long.

Exception Conditions

INVREQ

occurs if:

- The name of the transaction begins with either C or AEG and cannot be discarded (RESP2=4)
- The PCT profile is not available (RESP2=10)
- The transaction is not disabled. The transaction needs to be disabled for the discard request to work (RESP2=11)
- The named transaction is currently in use (RESP2=12)
- The transaction is in use by an automatic initiate descriptor (RESP2=15)
- The named transaction could not be discarded (RESP2=16)

Default action: Terminate the task abnormally.

TRANSIDERR

occurs if the named transaction cannot be found (RESP2=1).

Default action: Terminate the task abnormally.

INQUIRE commands

This section describes the EXEC CICS INQUIRE commands in alphabetic order.

INQUIRE AUTINSTMODEL

▶▶—INQUIRE AUTINSTMODEL—(*—name—*)—▶▶

Condition: MODELIDERR

The EXEC CICS INQUIRE AUTINSTMODEL command retrieves information about the named CICS autoinstall terminal model definition in the CICS control region.

Options

AUTINSTMODEL(name)

is the name of the CICS autoinstall terminal model as specified in the CICSDEV parameter within the OS/400 ADDCICSTCT CL command. The name may be up to 4 characters long.

Exception Conditions

MODELIDERR

occurs if the named autoinstall model cannot be found (RESP2=1).

Default action: Terminate the task abnormally.

INQUIRE AUTINSTMODEL (browse)

▶▶—INQUIRE AUTINSTMODEL—START—▶▶

Condition: ILLOGIC

▶▶—INQUIRE AUTINSTMODEL—(*—name—*)—NEXT—▶▶

Conditions: END, ILLOGIC

▶▶—INQUIRE AUTINSTMODEL—END—▶▶

Condition: ILLOGIC

Browse information about the named CICS autoinstall terminal model definition in the CICS/400 control region.

These commands allow you to browse through the autoinstall terminal model entries in the TCT. The order of browsing is strictly undefined, but continued browsing does guarantee to return all definitions. Only one browse of the TCT is allowed at any one time in a given task. See “Browsing resource definitions” on page 315 for more information.

Options

AUTINSTMODEL(name)

returns a 4-character string representing the name of the terminal model definition which is retrieved on an INQUIRE AUTINSTMODEL NEXT command.

END

terminates the browse and frees any held resources.

NEXT

retrieves the name and any requested attributes of the next (that is, first or subsequent) TCT entry to be accessed.

START

initializes the scan of the TCT in preparation for a series of INQUIRE AUTINSTMODEL NEXT commands. Note that no information about autoinstall terminals is retrieved until an INQUIRE AUTINSTMODEL NEXT command is executed.

Exception Conditions

END

occurs on an INQUIRE AUTINSTMODEL NEXT command if all TCT entries have been accessed (RESP2=2).

Default action: Terminate the task abnormally.

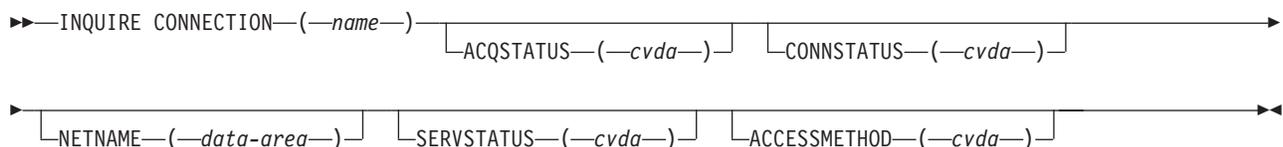
ILLOGIC

occurs in any of the following situations (RESP2=1):

- An INQUIRE AUTINSTMODEL START command is issued when a browse of the TCT is already in progress.
- An INQUIRE AUTINSTMODEL NEXT or INQUIRE AUTINSTMODEL END command is issued, but an INQUIRE AUTINSTMODEL START command has not been successfully issued.

Default action: Terminate the task abnormally.

INQUIRE CONNECTION



Condition: SYSIDERR

The EXEC CICS INQUIRE CONNECTION command retrieves information from the terminal control system table (TCS) about the named CICS connection (sometimes known as a “system entry”) in the CICS control region.

Options

ACCESSMETHOD(cvda)

returns a CVDA value indicating which access method is in use for the CICS connection. CVDA values are:

VTAM^{®*} The CICS connection is treated as a VTAM connection.

INDIRECT The intercommunication with the CICS connection uses the CICS connection specified in the INDSYS parameter within the OS/400 ADDCICSTCS CL command.

ACQSTATUS(cvda)

is retained only for compatibility purposes. CONNSTATUS should be used instead in new CICS application programs.

CONNECTION(name)

is the name of the remote system or another CICS control region as specified in the SYSID parameter within the OS/400 ADDCICSTCS CL command. The name may be up to 4 characters long.

CONNSTATUS(cvda)

returns a CVDA value indicating the state of the CICS connection between the CICS control region and the logical unit represented by the CONNECTION name. CVDA values are:

RELEASED The CICS connection is released.

ACQUIRED The CICS connection is acquired. The criteria for ACQUIRED are:

- The partner LU has been contacted
- Initial CNOS exchange has been done

NETNAME(data-area)

returns an 8-character string representing the name by which the remote system is known to the network.

SERVSTATUS(cvda)

returns a CVDA value indicating whether the CICS connection can receive and send data. CVDA values are:

INSERVICE Data can be received and sent.

OUTSERVICE Data cannot be received or sent.

GOINGOUT OUTSERVICE has been requested on a SET CONNECTION command, but the request cannot be acted upon until some current work has been completed.

Exception Conditions

SYSIDERR

occurs if the named connection cannot be found (RESP2=1).

Default action: Terminate the task abnormally.

INQUIRE CONNECTION (browse)

▶▶—INQUIRE CONNECTION—START—▶▶

Condition: ILLOGIC

▶▶—INQUIRE CONNECTION—(*—data-area—*)—NEXT—Other options as for INQUIRE CONNECTION—▶▶

Conditions: END, ILLOGIC

▶▶—INQUIRE CONNECTION—END—▶▶

Condition: ILLOGIC

Browse information about the system connections defined to CICS/400.

These commands allow you to browse through the terminal control system table (TCS) entries. The order of browsing is strictly undefined, but continued browsing does guarantee to return all the remote system definitions. Only one browse of the TCS is allowed at any one time in a given task. See “Browsing resource definitions” on page 315 for more information.

Options

CONNECTION(*data-area*)

returns a 4-character string representing the name of the remote system or CICS control region for which information is retrieved on an INQUIRE CONNECTION NEXT command.

END

terminates the browse and frees any held resources.

NEXT

retrieves the name and any requested attributes of the next (that is, first or subsequent) TCS entry to be accessed.

START

initializes the scan of the TCS in preparation for a series of INQUIRE CONNECTION NEXT commands. Note that no information about remote systems is retrieved until an INQUIRE CONNECTION NEXT command is executed.

Exception Conditions

END

occurs on an INQUIRE CONNECTION NEXT command if all TCS entries have been accessed (RESP2=2).

Default action: Terminate the task abnormally.

ILLOGIC

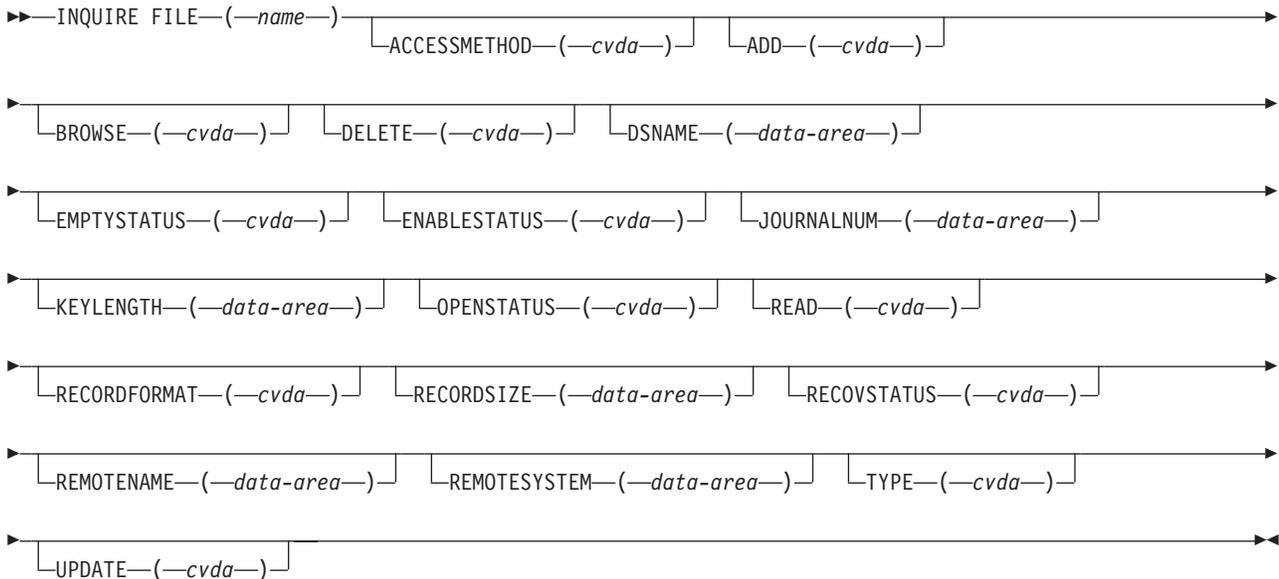
occurs in any of the following situations (RESP2=1):

- An INQUIRE CONNECTION START command is issued when a browse of the TCS is already in progress.

- An INQUIRE CONNECTION NEXT or INQUIRE CONNECTION END command is issued, but an INQUIRE CONNECTION START command has not been successfully issued.

Default action: Terminate the task abnormally.

INQUIRE FILE



Condition: FILENOTFOUND

Note: This command replaces the EXEC CICS INQUIRE DATASET command. The parameter DATASET is supported by the translator as a synonym for FILE, but FILE should be used for all new CICS application programs.

The EXEC CICS INQUIRE FILE command retrieves information about the named CICS file definition in the CICS control region.

Note: The values that are returned can vary according to when the command is issued. For example, if the CICS file is closed when the command is issued, much of the information received describes the state of the CICS file when it is next opened. If the CICS file has never been opened, default or null values will be received for some of the options that could change when the file is opened.

Options

ACCESSMETHOD(cvda)

returns a CVDA value indicating the access method for this CICS file. CVDA values are:

VSAM

The CICS file is to be treated like a VSAM file.

REMOTE

The CICS file resides on another CICS system.

ADD(cvda)

returns a CVDA value indicating whether new records can be added to the CICS file. CVDA values are:

ADDABLE New records can be added to the CICS file.

NOTADDABLE

New records cannot be added to the CICS file.

BROWSE(cvda)

returns a CVDA value indicating whether browsing the CICS file is allowed. CVDA values are:

BROWSABLE The CICS file can be browsed.

NOTBROWSABLE

The CICS file cannot be browsed.

DELETE(cvda)

returns a CVDA value indicating whether records can be deleted from the CICS file. CVDA values are:

DELETABLE Records can be deleted from the CICS file.

NOTDELETABLE

Records cannot be deleted from the CICS file.

DSNAME(data-area)

returns a 33-character string representing the name of the OS/400 file object with which this CICS file is associated.

EMPTYSTATUS(cvda)

returns a CVDA value indicating whether the OS/400 file object is to be made empty when a CICS file that refers to it is next opened, and is valid only for OS/400 file objects that have been defined as reusable. CVDA values are:

EMPTYREQ The OS/400 file object has been defined as reusable, and is to be made empty when a CICS file that refers to it is next opened.

NOEMPTYREQ

The OS/400 file object has been defined as reusable, but is not to be made empty when a CICS file that refers to it is next opened.

ENABLESTATUS(cvda)

returns a CVDA value indicating whether application programs can use the file. CVDA values are:

ENABLED The file is available for use by application programs.

DISABLED The file is unavailable for use by application programs.

DISABLING The CICS file is in the process of being disabled.

FILE(name)

is the name of the file as specified in the FILEID parameter within the OS/400 ADDCICSFCT CL command. The name may be up to 8 characters long.

JOURNALNUM(data-area)

returns a fullword binary field indicating the CICS journal number with which this CICS file is associated, in the range 1-99.

KEYLENGTH(data-area)

returns a fullword binary field indicating the length of the key, if the CICS file is to be treated as a VSAM KSDS.

OPENSTATUS(cvda)

returns a CVDA value indicating whether the file is open, closed, or in a transitional state. Possible CVDA values are:

- OPEN** The file can be accessed by an application program.
- CLOSED** The file cannot be accessed by an application program.
- CLOSING** The file is in the process of being closed.

READ(cvda)

returns a CVDA value indicating whether records can be read from the CICS file. CVDA values are:

- READABLE**
 Records can be read from the file.
- NOTREADABLE**
 Records cannot be read from the file.

RECORDFORMAT(cvda)

indicates whether the CICS file has fixed- or variable-length records. CVDA values are:

- FIXED** Records in the CICS file all have the same length.
- VARIABLE** Records in the file do not necessarily have the same length.

RECORDSIZE(data-area)

returns a fullword binary field indicating the maximum record length in bytes for CICS files defined as VARIABLE, and the actual record length for CICS files defined as FIXED.

RECOVSTATUS(cvda)

returns a CVDA value indicating whether the CICS file is recoverable or not. CVDA values are:

- RECOVERABLE**
 The CICS file is recoverable.
- NOTRECOVERABLE**
 The CICS file is not recoverable.

REMOTENAME(data-area)

returns an 8-character string representing the name that this CICS file has in the remote system.

REMOTESYSTEM(data-area)

returns a 4-character string representing the name of the remote system, if the CICS file is remote.

TYPE(cvda)

returns a CVDA value indicating how the records are organized on the CICS file. CVDA values are:

- ESDS** The records are organized by their original entry sequence.
- KSDS** The records are organized by key.
- RRDS** The records are organized by relative record number.

UPDATE(cvda)

returns a CVDA value indicating whether records can be updated in the CICS file.

UPDATABLE Records can be updated in the CICS file. Records can be read from the file, and either changed or deleted.

NOTUPDATABLE

Records cannot be updated in the CICS file.

Exception Conditions

FILENOTFOUND

occurs if the named file cannot be found (RESP2=1).

Default action: Terminate the task abnormally.

INQUIRE FILE (browse)

▶▶—INQUIRE FILE—START—▶▶

Condition: ILLOGIC

▶▶—INQUIRE FILE—(*—data-area—*)—NEXT—Other options as for INQUIRE FILE—▶▶

Conditions: END, ILLOGIC

▶▶—INQUIRE FILE—END—▶▶

Condition: ILLOGIC

Browse information about the files defined to CICS/400.

These commands allow you to browse through the file control table (FCT) entries. The order of browsing is strictly undefined, but continued browsing does guarantee to return all the definitions. Only one browse of the FCT is allowed at any one time in a given task. See “Browsing resource definitions” on page 315 for more information.

Options

END

terminates the browsing operation and frees any held resources.

FILE(data-area)

returns an 8-character string representing the name of the file for which information is retrieved on an INQUIRE FILE NEXT command.

NEXT

retrieves the name and any requested attributes of the next (that is, first or subsequent) FCT entry to be accessed.

START

initializes the scan of the FCT in preparation for a series of INQUIRE FILE NEXT commands. Note that no information about files is retrieved until an INQUIRE FILE NEXT command is executed.

Exception Conditions

END

occurs on an INQUIRE FILE NEXT command if all FCT entries have been accessed (RESP2=2).

Default action: Terminate the task abnormally.

ILLOGIC

occurs in any of the following situations (RESP2=1):

- An INQUIRE FILE START command is issued when a browse of the FCT is already in progress.
- An INQUIRE FILE NEXT or INQUIRE FILE END command is issued, but an INQUIRE FILE START command has not been successfully issued.

Default action: Terminate the task abnormally.

INQUIRE JOURNALNUM

►►—INQUIRE JOURNALNUM—(—*data-value*—)——┐ JTYPE—(—*cvda*—)┐┐ OPENSTATUS—(—*cvda*—)┐┐►►

Condition: JIDERR

The EXEC CICS INQUIRE JOURNALNUM command retrieves information about the named CICS journal definition in the CICS control region.

Options

JOURNALNUM(*data-value*)

specifies as a halfword binary value the number, in the range 1 through 99, of the CICS journal, as specified in the JFILE parameter within the OS/400 ADDCICSJCT CL command.

JTYPE(*cvda*)

returns a CVDA value indicating the type of CICS journal. CVDA values are:

DISK1 The CICS journal is to be written to a single, reusable OS/400 file object on disk.

DISK2 The CICS journal is to be written to multiple OS/400 file objects on disk. When the first OS/400 file object is full, then another OS/400 file object is automatically created on disk.

OPENSTATUS(*cvda*)

returns a CVDA value indicating whether the CICS journal is open for output or closed. CVDA values are:

OPENOUTPUT

The CICS journal is open to have records written to it.

CLOSED

The CICS journal is closed.

Exception Conditions

JIDERR

occurs if the named journal cannot be found (RESP2=1).

Default action: Terminate the task abnormally.

INQUIRE JOURNALNUM (browse)

▶▶—INQUIRE JOURNALNUM—START—▶▶

Condition: ILLOGIC

▶▶—INQUIRE JOURNALNUM—(*data-area*)—NEXT—Other options as for INQUIRE JOURNALNUM—▶▶

Conditions: END, ILLOGIC

▶▶—INQUIRE JOURNALNUM—END—▶▶

Condition: ILLOGIC

Browse through the CICS journal definitions in the JCT within the control region.

These commands allow you to browse through the journal control table (JCT) entries. The order of browsing is strictly undefined, but continued browsing does guarantee to return all the definitions. Only one browse of the JCT is allowed at any one time in a given task. See “Browsing resource definitions” on page 315 for more information.

Options

END

terminates the browsing operation and frees any held resources.

JOURNALNUM(*data-area*)

returns a halfword binary field indicating the number, in the range 1 through 99, of the CICS journal for which information is retrieved on an INQUIRE JOURNALNUM NEXT command.

NEXT

retrieves the number and any requested attributes of the next (that is, first or subsequent) JCT entry to be accessed.

START

initializes the scan of the JCT in preparation for a series of INQUIRE JOURNALNUM NEXT commands. Note that no information about journals is retrieved until an INQUIRE JOURNALNUM NEXT command is executed.

Options

END

occurs on an INQUIRE JOURNALNUM NEXT command if all JCT entries have been accessed (RESP2=2).

Default action: Terminate the task abnormally.

ILLOGIC

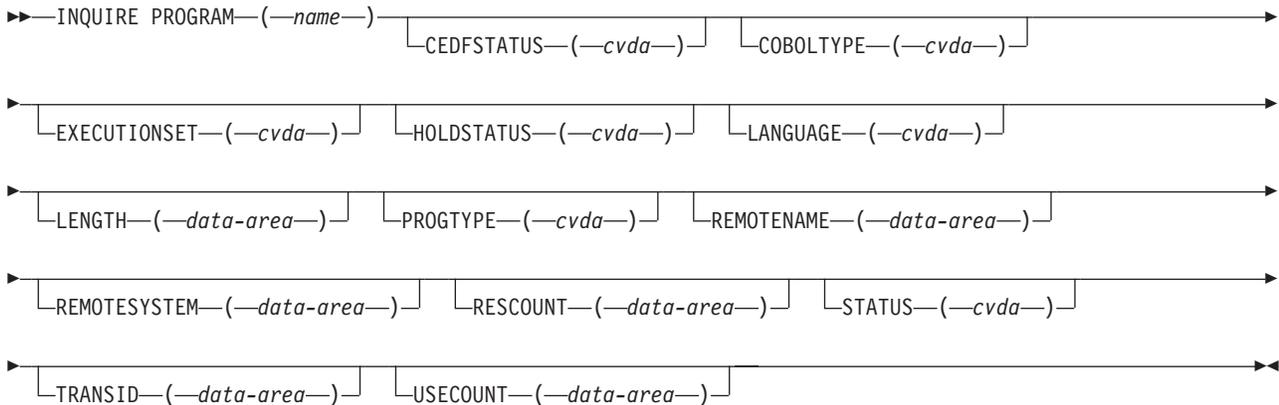
occurs in any of the following situations (RESP2=1):

- An INQUIRE JOURNALNUM START command is issued when a browse of the JCT is already in progress.

- An INQUIRE JOURNALNUM NEXT or INQUIRE JOURNALNUM END command is issued, but an INQUIRE JOURNALNUM START command has not been successfully issued.

Default action: Terminate the task abnormally.

INQUIRE PROGRAM



Condition: PGMIDERR

The EXEC CICS INQUIRE PROGRAM command retrieves information about a named CICS/400 program or map set definition in the CICS control region.

Options

CEDFSTATUS(cvda)

returns a CVDA value indicating the action of CEDF with this CICS/400 program. CVDA values are:

- | | |
|---------------|--|
| CEDF | CEDF works normally when this CICS program is being processed. |
| NOCEDF | CEDF actions, including program initiation and termination, are inhibited when this CICS program is being processed. |

COBOLTYPE(cvda)

returns a CVDA value indicating the type of COBOL being used with this CICS program. CVDA values are:

- | | |
|------------------|---|
| COBOL | The CICS program was written in AD/Cycle COBOL/400. |
| NOTINIT | The CICS program is not currently loaded. |
| NOTAPPLIC | The CICS program was not written in COBOL. |

EXECUTIONSET(cvda)

returns a CVDA value indicating whether CICS is to run the CICS program as if it is a linked-to CICS program running in a remote system, and subject to the CICS API restrictions of a distributed program link (DPL) program. CVDA values are:

- | | |
|------------------|---|
| DPLSUBSET | The CICS program is restricted, when it runs in the local CICS control region, to the same subset of the CICS API that applies when the CICS program is linked to by a DPL request. |
|------------------|---|

- FULLAPI** The CICS program is not restricted to the DPL subset of the CICS API when it runs in the local CICS control region, and can use the full CICS API.
- NOTAPPLIC** The PPT entry defines either a remote program or a BMS map set.

A CICS program is always restricted to the DPL subset when it is invoked in a remote system by a DPL request, regardless of the EXECUTIONSET option.

HOLDSTATUS(cvda)

returns a CVDA value indicating whether a copy of the CICS program is currently loaded with the HOLD option. CVDA values are:

- HOLD** The CICS program copy is currently loaded with the HOLD option.
- NOHOLD** The CICS program copy is not currently loaded with the HOLD option.
- NOTAPPLIC** The CICS program is not currently loaded.

LANGUAGE(cvda)

returns a CVDA value returns a CVDA value indicating the calling convention specified in the PPT. CVDA values are:

- COBOL** The COBOL/400 calling convention was specified.
- C** The ILE C calling convention was specified.

LENGTH(data-area)

returns a fullword binary field indicating the length of the CICS program in bytes. A value of 0 is returned if the CICS program is not currently loaded.

PROGRAM(name)

specifies the name of the CICS program or map set as specified in the PGMID parameter within the OS/400 ADDCICSPPT CL command. The name may be up to 8 characters long.

PROGTYPE(cvda)

returns a CVDA value indicating the type of CICS program. CVDA values are:

- PROGRAM** The CICS program is a CICS application program.
- MAP** The CICS program is a CICS BMS map set.

REMOTENAME(data-area)

returns a 4-character string representing the name that this CICS program has in the remote system.

REMOTESYSTEM(data-area)

returns an 8-character string indicating the name of the remote system, if the CICS program is remote.

RESCOUNT(data-area)

returns a fullword binary field indicating the number of separate invocations of this CICS program that are taking place at the time of this inquiry.

STATUS(cvda)

returns a CVDA value indicating whether CICS/400 can use the program. CVDA values are:

- ENABLED** The program is available for use.
- DISABLED** The program is unavailable for use.

TRANSID(data-area)

returns a 4-character string representing the name of the server transaction that the remote system attaches to run the CICS program, when the CICS program is defined as remote. If *NONE is specified in the TRANSID parameter within the ADDCICSPPT CL command, CICS returns blanks (X'40404040').

USECOUNT(data-area)

returns a fullword binary field indicating the total number of times the CICS program has been executed since the CICS program was loaded.

Exception Conditions

PGMIDERR

occurs if the named program cannot be found (RESP2=1).

Default action: Terminate the task abnormally.

INQUIRE PROGRAM (browse)

▶▶—INQUIRE PROGRAM—START—▶▶

Condition: ILLOGIC

▶▶—INQUIRE PROGRAM—(—data-area—)—NEXT—Other options as for INQUIRE PROGRAM—▶▶

Conditions: END, ILLOGIC

▶▶—INQUIRE PROGRAM—END—▶▶

Condition: ILLOGIC

Browse information about the programs, map sets, and tables defined to CICS/400.

These commands allow you to browse through the processing program table (PPT) entries. The order of browsing is strictly undefined, but continued browsing does guarantee to return all the definitions. Only one browse of the PPT is allowed at any one time in a given task. See "Browsing resource definitions" on page 315 for more information.

Options

END

terminates the browsing operation and frees any held resources.

NEXT

retrieves the name and any requested attributes of the next (that is, first or subsequent) PPT entry to be accessed.

PROGRAM(data-area)

returns an 8-character string representing the name of the program for which information is retrieved on an INQUIRE PROGRAM NEXT command.

START

initializes the scan of the PPT in preparation for a series of INQUIRE

PROGRAM NEXT commands. Note that no information about programs is retrieved until an INQUIRE PROGRAM NEXT command is executed.

Exception Conditions

END

occurs on an INQUIRE PROGRAM NEXT command if all PPT entries have been accessed (RESP2=2).

Default action: Terminate the task abnormally.

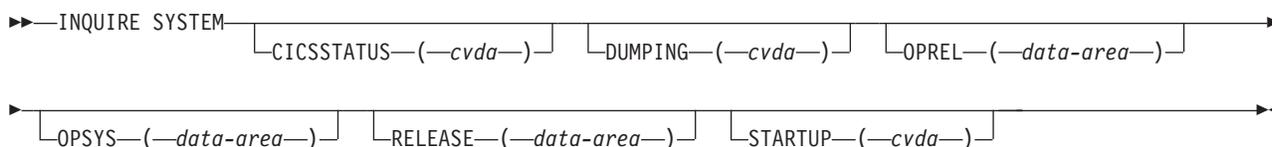
ILLOGIC

occurs in any of the following situations (RESP2=1):

- An INQUIRE PROGRAM START command is issued when a browse of the PPT is already in progress.
- An INQUIRE PROGRAM NEXT or INQUIRE PROGRAM END command is issued, but an INQUIRE PROGRAM START command has not been successfully issued.

Default action: Terminate the task abnormally.

INQUIRE SYSTEM



The EXEC CICS INQUIRE SYSTEM command retrieves information about the CICS/400 control region.

Options

CICSTATUS(cvda)

returns a CVDA value indicating the current status of the CICS control region. CVDA values are:

ACTIVE The control region is fully active.

FIRSTQUIESCE

The control region is in the first quiesce stage of shutdown.

DUMPING(cvda)

returns a CVDA value indicating whether the taking of a CICS control region dump is to be globally suppressed. CVDA values are:

SYSDUMP CICS control region dumps are not globally suppressed.

NOSYSDUMP

CICS control region dumps are globally suppressed.

OPREL(data-area)

returns a halfword binary field indicating the release number of the operating system currently running. It is a halfword binary integer equal to 10 times the formal release number. For example, a returned value of "30" represents OS/400 Version 5 Release 2.

DEST The task was initiated by a destination trigger level as defined in the destination control table (DCT).

STARTCODE(data-area)

returns a 2-character string representing a value that indicates how this CICS task was started. Possible values are:

- D** Distributed program link (DPL).
- DS** DPL plus sync-on-return.
- QD** A CICS transient data trigger level was reached.
- S** START command (no data).
- SD** START command (with data).
- TO** Operator entered a CICS transaction code at the terminal.
- TP** Transaction was started by presetting the transaction ID for the CICS terminal, either by using RETURN TRANSID, or by having a CICS transaction permanently assigned to the CICS terminal.
- U** CICS user-attached task.

TASK(data-value)

specifies as a 4-byte packed decimal value the sequence number that identifies the task.

TRANSACTION(data-area)

returns a 4-character string representing the transaction name (if any) associated with the task.

USERID(data-area)

returns a 10-character string representing the identifier of the CICS user currently associated with the CICS task.

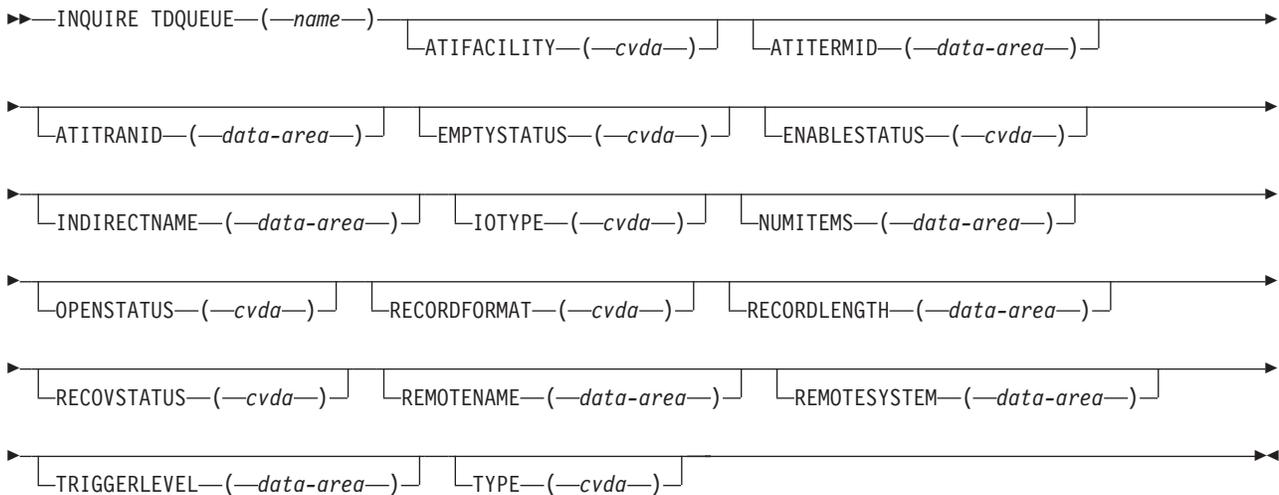
Exception Conditions

TASKIDERR

occurs if the named task cannot be found (RESP2=1).

Default action: Terminate the task abnormally.

INQUIRE TDQUEUE



Condition: QIDERR

The EXEC CICS INQUIRE TDQUEUE command retrieves information about the named transient data queue definition in the CICS/400 control region.

Note: For a closed extrapartition queue, the values that are returned can vary according to when the command is issued. For example, if the CICS transient data queue is closed when the command is issued, much of the information received describes the state of the CICS transient data queue when it is next opened. If the CICS transient data queue has never been opened, default or null values will be received for some of the options, that could change when the CICS transient data queue is opened.

Options

ATIFACILITY(cvda) (intrapartition queues only)

returns a CVDA value indicating whether or not the CICS task that is to be started when the TRIGGERLEVEL value is reached, is associated with a CICS terminal (or session). CVDA values are:

TERMINAL A CICS terminal (or session) is associated with the CICS task that is to be started when the TRIGGERLEVEL value is reached.

NOTERMINAL

A CICS terminal (or session) is not associated with the CICS task that is to be started when the TRIGGERLEVEL value is reached.

ATITERMID(data-area) (intrapartition queues only)

returns a 4-character string representing the name of the terminal or session to be associated with the queue when automatic transaction initiation occurs. A null value is returned if the transaction does not need a terminal or session.

ATITRANID(data-area) (intrapartition queues only)

returns a 4-character string representing the name of the CICS transaction to be started when the TRIGGERLEVEL value is reached.

EMPTYSTATUS(cvda) (intrapartition queues only)

returns a CVDA value indicating whether the queue is full, empty, or neither. CVDA values are:

- FULL** The TRIGGERLEVEL value has been reached.
- EMPTY** The queue does not have data.
- NOTEMPTY** The queue has data, but the TRIGGERLEVEL value has not been reached.
- NOTAPPLIC** The queue is not enabled.

ENABLESTATUS(cvda) (all except indirect queues)

returns a CVDA value indicating whether CICS application programs can use the queue. CVDA values are:

- ENABLED** The queue is available for use by CICS application programs.
- DISABLED** The queue is unavailable for use by CICS application programs, although it may still be open.

INDIRECTNAME(data-area) (indirect queues only)

returns a 4-character string representing the name of the queue that this indirect queue points to, as specified in the PHYDEST parameter within the OS/400 ADDCICSDCT CL command.

IOTYPE(cvda) (extrapartition queues only)

returns a CVDA value indicating whether records can be read or written to the queue. CVDA values are:

- INPUT** Records can only be read from the queue, and only in a forward capacity.
- OUTPUT** Records can only be written to the queue.
- READBACK** Records can only be read from the queue, and only in a backward capacity.

NUMITEMS(data-area) (intrapartition queues only)

returns a fullword binary field representing the logical number of records in the queue.

OPENSTATUS(cvda) (extrapartition queues only)

returns a CVDA value indicating whether the queue is open, closed, or in a transitional state. CVDA values are:

- OPEN** The queue is open to have records read or written to it.
- CLOSED** The queue is closed.
- OPENING** The queue is in the process of being opened.
- CLOSING** The queue is in the process of being closed.

RECORDFORMAT(cvda)(extrapartition queues only)

returns a CVDA value indicating whether the queue has fixed- or variable-length records. CVDA values are:

- FIXED** Records in the queue all have the same length.
- VARIABLE** Records in the queue do not necessarily have the same length.
- NOTAPPLIC** The specified queue is not enabled.

RECORDLENGTH(data-area) (extrapartition queues only)

returns a fullword binary field indicating the maximum record length for

queues defined as VARIABLE, and the actual record length for queues defined as FIXED. The value returned is the number of bytes.

RECOVSTATUS(cvda) (intrapartition queues only)

returns a CVDA value indicating whether the queue is recoverable or not. CVDA values are:

LOGICAL The queue is recoverable.

NOTRECOVERABLE
 The queue is not recoverable.

REMOTENAME(data-area) (remote queues only)

returns a 4-character string representing the name that this queue has in the remote system.

REMOTESYSTEM(data-area) (remote queues only)

returns a 4-character string representing the name of the remote system, if the queue is remote.

TDQUEUE(name)

is the name of the transient data queue (the destination) as specified in the DEST parameter within the OS/400 ADDCICSDCT CL command. The name may be up to 4 characters long.

TRIGGERLEVEL(data-area) (intrapartition queues only)

returns a fullword binary field indicating the number of requests for output to the queue that there must be before automatic transaction initiation (ATI) can occur.

TYPE(cvda)

returns a CVDA value indicating the type of queue. CVDA values are:

EXTRA The queue is an extrapartition transient data queue.

INDIRECT The queue is an indirect transient data queue.

INTRA The queue is an intrapartition transient data queue.

REMOTE The queue is a remote transient data queue.

Exception Conditions

QIDERR

occurs if the named queue cannot be found (RESP2=1).

Default action: Terminate the task abnormally.

INQUIRE TDQUEUE (browse)

▶▶—INQUIRE TDQUEUE—START—▶▶

Condition: ILLOGIC

▶▶—INQUIRE TDQUEUE—(*—data-area—*)—NEXT—Other options as for INQUIRE TDQUEUE—▶▶

Conditions: END, ILLOGIC

▶▶—INQUIRE TDQUEUE—END—▶▶

Condition: ILLOGIC

Browse information about the transient data queues defined to CICS/400.

These commands allow you to browse through the destination control table (DCT) entries. The order of browsing is strictly undefined, but continued browsing does guarantee to return all the definitions. Only one browse of the DCT is allowed at any one time in a given task. See “Browsing resource definitions” on page 315 for more information.

Options

END

terminates the browsing operation and frees any held resources.

NEXT

retrieves the name and any requested attributes of the next (that is, first or subsequent) DCT entry to be accessed.

START

initializes the scan of the DCT in preparation for a series of INQUIRE TDQUEUE NEXT commands. Note that no information about transient data queues is retrieved until an INQUIRE TDQUEUE NEXT command is executed.

TDQUEUE(*data-value*)

returns a 4-character string representing the name of the transient data queue for which information is retrieved on an INQUIRE TDQUEUE NEXT command.

Exception Conditions

END

occurs on an INQUIRE TDQUEUE NEXT command if all DCT entries have been accessed (RESP2=2).

Default action: Terminate the task abnormally.

ILLOGIC

occurs in any of the following situations (RESP2=1):

- An INQUIRE TDQUEUE START command is issued when a browse of the DCT is already in progress.

- An INQUIRE TDQUEUE NEXT or INQUIRE TDQUEUE END command is issued, but an INQUIRE TDQUEUE START command has not been successfully issued.

Default action: Terminate the task abnormally.

INQUIRE TERMINAL or NETNAME

```

▶▶ INQUIRE TERMINAL—(—name—) [NETNAME—(—data-area—)] | Other options |

```

or

```

▶▶ INQUIRE NETNAME—(—name—) [TERMINAL—(—data-area—)] | Other options |

```

Other options:

```

| [ACQSTATUS—(—cvda—)] [ATISTATUS—(—cvda—)] [DEVICE—(—cvda—)]
▶ [GCHARS—(—data-area—)] [GCODES—(—data-area—)] [MODENAME—(—data-area—)]
▶ [NEXTTRANSID—(—data-area—)] [REMOTENAME—(—data-area—)]
▶ [REMOTESYSTEM—(—data-area—)] [SCREENHEIGHT—(—data-area—)]
| [SCRNHT—(—data-area—)]
▶ [SCREENWIDTH—(—data-area—)] [SERVSTATUS—(—cvda—)] [SESSIONTYPE—(—cvda—)]
| [SCRNWD—(—data-area—)]
▶ [SIGNONSTATUS—(—cvda—)] [TASKID—(—data-area—)] [TERMMODEL—(—data-area—)]
▶ [TRANSACTION—(—data-area—)] [TTISTATUS—(—cvda—)] [USERAREA—(—pointer—)]
▶ [USERAREALEN—(—data-area—)] [USERID—(—data-area—)] [USERNAME—(—data-area—)]

```

Note: All options apply to either form of the command.

Condition: TERMIDERR

Forms of command

This command has two forms: EXEC CICS INQUIRE TERMINAL and EXEC CICS INQUIRE NETNAME.

The EXEC CICS INQUIRE TERMINAL and EXEC CICS INQUIRE NETNAME commands retrieve information about the named terminal definition.

Options

ACQSTATUS(cvda)

returns a CVDA value indicating whether CICS/400 is in session with the logical unit represented by the named terminal. CVDA values are:

ACQUIRED CICS/400 is in session with the terminal.

RELEASED CICS/400 is not in session with the terminal.

ATISTATUS(cvda)

returns a CVDA value indicating whether the terminal is available for use by transactions that are automatically initiated by the CICS/400 control region or, if the terminal is an ISC session, by transactions that are using this session as an alternate facility to communicate with another system. CVDA values are:

ATI The terminal is available for use by transactions that are automatically initiated. If the terminal is an ISC session, the terminal can be used by transactions that are using this session as an alternate facility to communicate with another system.

NOATI

The terminal is not available for use by transactions that are automatically initiated.

DEVICE(cvda)

returns a CVDA value indicating the CICS terminal or session type. CVDA values are returned as follows:

Terminal type	CVDA
5250	T3277R
3270	T3277R
3270P	T3284L
SCS	T3790SCSP
3270J	T3277R
3270JP	T3284L
ASCII	T3277R

GCHARS(data-area)

returns a halfword binary field indicating the graphic character set global identifier (GCSGID). This is a registered number in the range 1 through 65 535 that indicates the set of graphic characters that can be input or output at this CICS terminal.

The term “coded graphic character set identifier” is commonly used when referring to GCHARS and GCODES together.

GCODES(data-area)

returns a halfword binary field indicating the code page global identifier (CPGID). This is a registered number in the range 1 through 65 535 that indicates the EBCDIC code page that defines the code points for the characters that can be input or output at the CICS terminal.

The term “coded graphic character set identifier” is commonly used when referring to GCHARS and GCODES together.

MODENAME(data-area) (APPC only)

returns an 8-character string representing the name of a group of parallel sessions (to which the named CICS terminal belongs), that have similar characteristics. The name is passed to the network as the LOGMODE name.

NETNAME(data-area)

returns an 8-character string representing the name of the logical unit in the network that is associated with the terminal named on an INQUIRE TERMINAL command.

NETNAME(name)

specifies, on an INQUIRE NETNAME command, the netname for which information is being requested. The name may be up to 8 characters long.

The first option used as an input field must be either TERMINAL or NETNAME. The value that is supplied is used as the search argument. Whichever is specified first, the other can be specified if required as the second option, which is always an output field.

For parallel sessions the NETNAME is not unique, so CICS/400 returns the name of the first session that it finds into the TERMINAL field.

NEXTTRANSID(data-area)

returns a 4-character string representing the name of the next transaction to be run after an EXEC CICS RETURN command. If there is no next transaction, blanks (X'40404040') are returned.

REMOTENAME(data-area)

returns a 4-character string representing the name that this terminal has in the remote system.

REMOTESYSTEM(data-area)

returns a 4-character string representing the name of the remote system, if the CICS terminal is a session or remote CICS terminal.

SCRNHT or SCREENHEIGHT(data-area)

returns a halfword binary field indicating the height of the current screen. This value depends on the mode of the terminal (default or alternate) at the time of the inquiry.

SCRNWD or SCREENWIDTH(data-area)

returns a halfword binary field indicating the width of the current screen. This value depends on the mode of the CICS terminal (default or alternate) at the time of the inquiry.

SERVSTATUS(cvda)

returns a CVDA value indicating whether the CICS terminal is available for use. CVDA values are:

INSERVICE The terminal is available for use.

OUTSERVICE The terminal is not available for use.

SESSIONTYPE(cvda)

returns a CVDA value indicating the type of session when the "terminal" is a session with another CICS system. CVDA values are:

APPCPARALLEL

The CICS terminal is a session to another system.

NOTAPPLIC The CICS terminal is not a session.

SIGNONSTATUS(cvda)

returns a CVDA value indicating whether the CICS terminal currently has a CICS user signed on. CVDA values are:

SIGNEDON The CICS terminal currently has an associated CICS user.

SIGNEDOFF The CICS terminal currently does not have an associated CICS user.

TASKID(data-area)

returns a fullword binary field indicating the identifier for the CICS task currently being executed at the CICS terminal.

TERMINAL(data-area)

returns a 4-character string representing the name of the terminal associated with the netname specified on an INQUIRE NETNAME command.

TERMINAL(name)

is the name of the CICS terminal as defined in the CICSDEV parameter within the OS/400 ADDCICSTCT CL command. The name may be up to 4 characters long. Also see the NETNAME(name) option.

TERMMODEL(data-area)

returns a halfword binary field indicating the model number.

TRANSACTION(data-area)

returns a 4-character string representing the name of the transaction currently being executed at the terminal.

TTISTATUS(cvda)

returns a CVDA value indicating whether transactions can be started at the terminal. If not, you can start transactions using either ATI or the EXEC CICS START command. CVDA values are:

TTI Transactions can be started at the terminal.

NOTTI

Transactions cannot be started at the terminal.

USERAREA(pointer)

returns the address of the TCTUA containing the process control information (PCI) for the terminal.

USERAREALEN(data-area)

returns a halfword binary field indicating the length of the user area.

USERID(data-area)

returns a 10-character string representing the identifier of the user that is signed on at this terminal or session.

USERNAME(data-area)

returns a 50-character string representing the name, as specified in the text within the OS/400 user profile, of the user signed on at this terminal or session.

Exception Conditions

TERMIDERR

occurs if the named terminal cannot be found (RESP2=1).

Default action: Terminate the task abnormally.

INQUIRE TERMINAL (browse)

▶▶—INQUIRE TERMINAL—START—▶▶

Condition: ILLOGIC

▶▶—INQUIRE TERMINAL—(*—data-area—*)—NEXT—Other options as for INQUIRE TERMINAL—▶▶

Conditions: END, ILLOGIC

▶▶—INQUIRE TERMINAL—END—▶▶

Condition: ILLOGIC

Browse information about the terminals defined to CICS/400.

These commands allow you to browse through the terminal control table (TCT) entries. The order of browsing is strictly undefined, but continued browsing does guarantee to return all the terminal definitions. Only one browse of the TCT is allowed at any one time in a given task. See "Browsing resource definitions" on page 315 for more information.

Note: There is no INQUIRE NETNAME browse function.

Options

END

terminates the browsing operation and frees any held resources.

NEXT

retrieves the name and any requested attributes of the next (that is, first or subsequent) TCT entry to be accessed.

START

initializes the scan of the TCT in preparation for a series of INQUIRE TERMINAL NEXT commands. Note that no information about terminals is retrieved until an INQUIRE TERMINAL NEXT command is executed.

TERMINAL(*data-area*)

returns a 4-character string representing the name of the terminal for which information is retrieved on an INQUIRE TERMINAL NEXT command.

Exception Conditions

END

occurs on an INQUIRE TERMINAL NEXT command if all TCT entries have been accessed (RESP2=2).

Default action: Terminate the task abnormally.

ILLOGIC

occurs in any of the following situations (RESP2=1):

- An INQUIRE TERMINAL START command is issued when a browse of the TCT is already in progress.

SWITCHALL Automatic switching occurs as necessary until the end of the CICS control region, without the need for operator intervention.

TABLESIZE(data-area)

returns a fullword binary field indicating the number of trace entries that will be kept by CICS internal tracing.

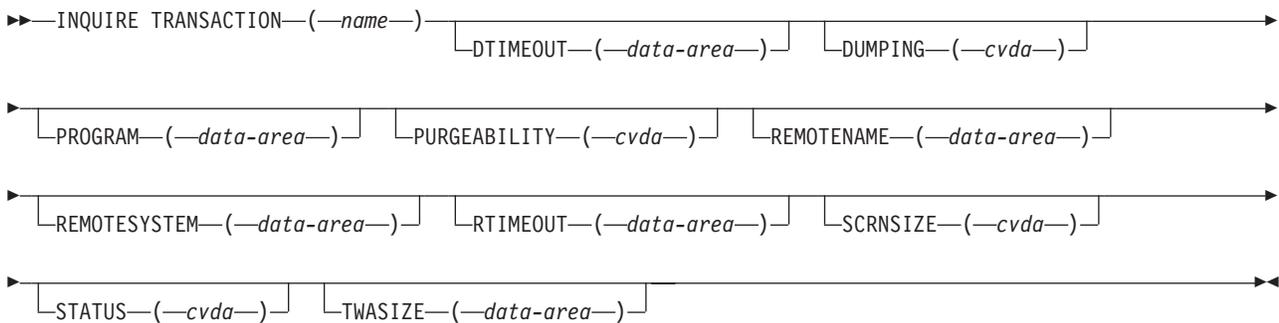
USERSTATUS(cvda)

returns a CVDA value indicating whether the CICS user tracing is to be active or not in the CICS control region. The input CVDA values are:

USERON The CICS user tracing is on.

USEROFF The CICS user tracing is off.

INQUIRE TRANSACTION



Condition: TRANSIDERR

The INQUIRE TRANSACTION command retrieves information about the named transaction definition from the program control table (PCT).

Options

DTIMEOUT(data-area)

returns a fullword binary field indicating the deadlock time-out value (in seconds) for suspended tasks associated with the transaction.

DUMPING(cvda)

returns a CVDA value indicating whether transaction dumps are taken when the transaction terminates abnormally. CVDA values are:

TRANDUMP A transaction dump will be taken when the transaction terminates abnormally.

NOTRANDUMP

A transaction dump will not be taken when the transaction terminates abnormally.

The setting of this option is ignored when an explicit DUMP TRANSACTION command is issued.

PROGRAM(data-area)

returns an 8-character string representing the name of the program to be executed when the transaction is started. The name of the program is specified in the PGMID parameter within the OS/400 ADDCICSPCT CL command.

PURGEABILITY(cvda)

returns a CVDA value indicating whether the transaction may be purged.
CVDA values are:

PURGEABLE The transaction may be purged.

NOTPURGEABLE

The transaction may not be purged.

REMOTENAME(data-area)

returns a 4-character string representing the name that this transaction has in the remote system.

REMOTESYSTEM(data-area)

returns a 4-character data string representing the name of the remote system, if the transaction is remote.

RTIMEOUT(data-area)

returns a fullword binary field indicating the read time-out value, which is the number of seconds after which a task associated with this transaction is terminated if no input is received.

SCRNSIZE(cvda)

returns a CVDA value indicating whether the alternate or the default screen size is to be used when this transaction is executed. CVDA values are:

ALTERNATE The alternate screen size is to be used when executing this transaction.

DEFAULT The default screen size is to be used when executing this transaction.

STATUS(cvda)

returns a CVDA value indicating whether the transaction is available for use.
CVDA values are:

ENABLED The transaction is available for use.

DISABLED The transaction is not available for use.

TRANSACTION(name)

is the name of the transaction as specified in the TRANSID parameter within the OS/400 ADDCICSPCT CL command. The name may be up to 4 characters long.

TWASIZE(data-area)

returns a fullword binary field indicating the size of the transaction work area (TWA) in bytes for this transaction.

Exception Conditions

TRANSIDERR

occurs if the named transaction cannot be found (RESP2=1).

Default action: Terminate the task abnormally.

INQUIRE TRANSACTION (browse)

▶▶—INQUIRE TRANSACTION—START—▶▶

Condition: ILLOGIC

▶▶—INQUIRE TRANSACTION—(—*data-area*—)—NEXT—Other options as for INQUIRE TRANSACTION—▶▶

Conditions: END, ILLOGIC

▶▶—INQUIRE TRANSACTION—END—▶▶

Condition: ILLOGIC

Browse information about the transactions defined to CICS/400.

These commands allow you to browse through the program control table (PCT) entries. The order of browsing is strictly undefined, but continued browsing does guarantee to return all the definitions. Only one browse of the PCT is allowed at any one time in a given task. See “Browsing resource definitions” on page 315 for more information.

Options

END

terminates the browsing operation and frees any held resources.

NEXT

retrieves the name and any requested attributes of the next (that is, first or subsequent) PCT entry to be accessed.

START

initializes the scan of the PCT in preparation for a series of INQUIRE TRANSACTION NEXT commands. Note that no information about transactions is retrieved until an INQUIRE TRANSACTION NEXT command is executed.

TRANSACTION(*data-area*)

returns a 4-character string representing the name of the transaction for which information is retrieved on an INQUIRE TRANSACTION NEXT command.

Exception Conditions

END

occurs on an INQUIRE TRANSACTION NEXT command if all PCT entries have been accessed (RESP2=2).

Default action: Terminate the task abnormally.

ILLOGIC

occurs in any of the following situations (RESP2=1):

- An INQUIRE TRANSACTION START command is issued when a browse of the PCT is already in progress.

Options

CONNECTION(name)

is the name of the connection (that is, the remote system, or another CICS control region) as specified in the SYSID parameter within the OS/400 ADDCICSTCS CL command.

PURGETYPE(cvda)

causes the specified CICS connection to purge associated CICS tasks. The input CVDA value is:

PURGE Transactions can be terminated only if system and data integrity can be maintained.

A task is not purged if its associated transaction definition specified *NO in the PURGE parameter in the OS/400 ADDCICSPCT CL command.

SERVSTATUS(cvda)

causes the specified CICS connection to start or stop the ability to receive and send data. The input CVDA values are:

INSERVICE Data is allowed to be sent and received.

OUTSERVICE Data is not allowed to be sent or received.

Exception Conditions

INVREQ

occurs if:

- SERVSTATUS has an incorrect CVDA value (RESP2=4).
- PURGETYPE has an incorrect CVDA value (RESP2=7).
- The EXEC CICS SET command is named on an indirect connection (RESP2=16).

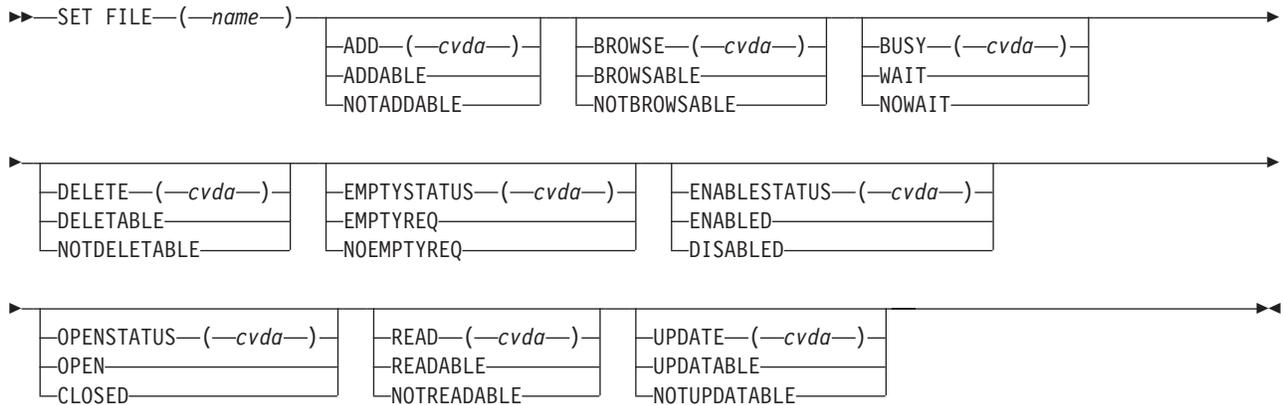
Default action: Terminate the task abnormally.

SYSIDERR

occurs if the named connection cannot be found (RESP2=9).

Default action: Terminate the task abnormally.

SET FILE



Conditions: FILENOTFOUND, INVREQ, IOERR

Note: This command replaces the EXEC CICS SET DATASET command. The parameter DATASET is supported by the translator as a synonym for FILE, but FILE should be used for all new CICS application programs.

Notes

All changes, other than to close and disable the file, require that the file be in a closed or disabled state, and they do not take effect until the file is next opened.

The requested changes are applied in the following order: CLOSED, DISABLED, ENABLED, OPEN, miscellaneous.

The EXEC CICS SET FILE command changes some of the information of a named CICS file definition. The file must not be remote.

Any combination of the options shown in the syntax can be specified on one command.

Options

ADD(cvda)

causes the specified CICS file to allow or not allow new records to be added to the CICS file. The input CVDA values are:

ADDABLE New records can be added to the CICS file.

NOTADDABLE
New records cannot be added to the CICS file.

BROWSE(cvda)

causes the specified CICS file to allow or not allow browsing of the CICS file. The input CVDA values are:

BROWSABLE The CICS file can be browsed.

NOTBROWSABLE
The CICS file cannot be browsed.

BUSY(cvda)

causes the specified file to wait or not wait, when the file is in use at the time that the EXEC CICS SET command is issued. The BUSY option is valid only for requests to set the file DISABLED or CLOSED, and is ignored for all other requests.

Note: There is no default. This differs from CICS for MVS/ESA, where WAIT is the default. The input CVDA values are:

- | | |
|---------------|--|
| WAIT | The system waits until all activity on the file has quiesced before setting the file DISABLED or CLOSED; then returns control to the application program that is issuing this command. |
| NOWAIT | The system does not wait until all activity on the file has quiesced; instead it returns control to the application program without waiting for the file to be DISABLED or CLOSED. |

DELETE(delete)

causes the specified file to allow or not allow deletion of records from the file. The input CVDA values are:

- | | |
|---------------------|--|
| DELETABLE | Records can be deleted from the file. |
| NOTDELETABLE | Records cannot be deleted from the file. |

EMPTYSTATUS(cvda)

specifies whether the file is to be emptied when it is next opened. This is valid only for files that have been defined as reusable. The input CVDA values are:

- | | |
|-------------------|--|
| EMPTYREQ | The file is made empty when it is next opened. |
| NOEMPTYREQ | The file is not made empty it is next opened. |

ENABLESTATUS(cvda)

causes the specified file to allow or not allow use by an application program. The input CVDA values are:

- | | |
|-----------------|--|
| ENABLED | The file is available for use by an application program. |
| DISABLED | The file is unavailable for use by an application program. |

FILE(name)

is the name of the file as specified in the FILEID parameter of the OS/400 ADDCICSFCT CL command.

OPENSTATUS(cvda)

causes the specified file to be opened or closed. The input CVDA values are:

- | | |
|---------------|--|
| OPEN | The file is to be opened, when used by an application program. |
| CLOSED | The file is to be closed on all CICS processes. |

READ(cvda)

causes the specified file to allow or not allow records to be read from the CICS file. The input CVDA values are:

- | | |
|--------------------|---|
| READABLE | The records can be read from the file. |
| NOTREADABLE | The records cannot be read from the file. |

UPDATE(cvda)

causes the specified file to allow or not allow records to be updated. The input CVDA values are:

UPDATABLE The records can be read, and either changed or deleted from the file.

NOTUPDATABLE

The records cannot be changed in the file.

Exception Conditions**FILENOTFOUND**

occurs if the named file cannot be found (RESP2=18).

Default action: Terminate the task abnormally.

INVREQ

occurs if:

- The named file is remote (RESP2=1)
- The named file is not closed (RESP2=2)
- The named file is not disabled (RESP2=3)
- ADD has an incorrect CVDA value (RESP2=4)
- BROWSE has an incorrect CVDA value (RESP2=5)
- BUSY has an incorrect CVDA value (RESP2=6)
- DELETE has an incorrect CVDA value (RESP2=7)
- EMPTYSTATUS has an incorrect CVDA value (RESP2=9)
- READ has an incorrect CVDA value (RESP2=12)
- UPDATE has an incorrect CVDA value (RESP2=14)
- OPENSTATUS has an incorrect CVDA value (RESP2=16)
- ENABLESTATUS has an incorrect CVDA value (RESP2=17)
- CLOSED or DISABLED has been specified, but the file is currently in use (RESP2=21)
- ENABLED was specified for a file that is currently disabling or closing (RESP2=22)
- The file name begins with AEG

Default action: Terminate the task abnormally.

IOERR

occurs if the open request has failed (RESP2=1).

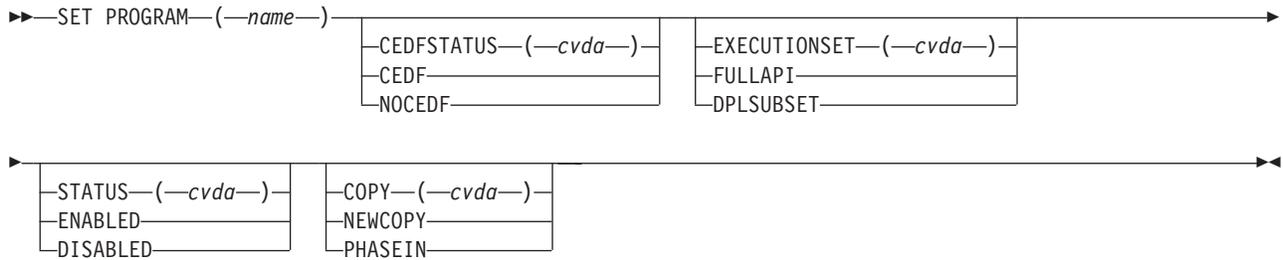
Default action: Terminate the task abnormally.

Example

```
EXEC CICS SET FILE ('FILE12')
           WAIT
           CLOSED
           DISABLED
           DELETABLE
EXEC CICS SET FILE ('FILE12')
           ENABLED
```

On the first command, the WAIT option tells CICS to allow all activity on FILE12 to quiesce before closing the CICS file, and to return control to the issuing CICS

SET PROGRAM



Conditions: INVREQ, IOERR, NOTAUTH, PGMIDERR

The EXEC CICS SET PROGRAM command changes some of the information of a named program definition.

Options

CEDFSTATUS(cvda)

causes the specified CICS program to allow or not allow the action of CEDF with this CICS program. The input CVDA value are:

- | | |
|---------------|---|
| CEDF | CEDF works normally when this CICS program is being processed. |
| NOCEDF | CEDF actions, including program initiation and termination, are suppressed when this CICS program is being processed. |

CEDFSTATUS cannot be specified for a remote program.

COPY(cvda)

specifies when to load a new copy of the program. The input CVDA values are:

- | | |
|----------------|--|
| NEWCOPY | CICS/400 will use a new copy of the program when it is subsequently referenced by, for example, a LOAD, LINK, or XCTL command. The request does not take effect unless the RESCOUNT value is zero. If you specify NEWCOPY when the RESCOUNT value is not zero, the INVREQ condition is raised. CICS loads a new version, resolving the object as defined in the PGMOBJ parameter of the PPT entry. |
| PHASEIN | CICS/400 will use a new copy of the program now. The existing version remains in storage until its RESCOUNT value reaches zero. All new LOAD, LINK, and XCTL requests are directed to the newly-loaded version. CICS/400 loads the new version, resolving the object as defined in the PGMOBJ parameter of the PPT entry. |

COPY is an incorrect option for any program currently loaded with the HOLD option, or for any program defined as remote.

EXECUTIONSET(cvda)

indicates whether CICS is to run the CICS program as if it is a linked-to CICS program running in a remote system, and subject to the CICS API restrictions of a distributed program link (DPL) program. CVDA values are:

- | | |
|------------------|--|
| DPLSUBSET | The CICS program is restricted, when it runs in the local CICS |
|------------------|--|

control region, to the same subset of the CICS API that applies when the CICS program is linked to by a DPL request.

FULLAPI The CICS program is not restricted to the DPL subset of the CICS API when it runs in the local CICS control region, and can use the full CICS API.

EXECUTIONSET cannot be specified for a remote program or map set.

PROGRAM(name)

is the name of the CICS program as specified in the PGMID parameter of the OS/400 ADDCICSPPT CL command. The name may be up to 8 characters long.

STATUS(cvda)

causes the specified CICS program to be available or unavailable for use by CICS. CICS programs beginning with "AEG" cannot be disabled. The input CVDA values are:

ENABLED The program is available for use.

DISABLED The program is unavailable for use.

Exception Conditions

INVREQ

occurs if:

- DISABLED was specified for a program with a name beginning with AEG or a program that used EXECUTIONSET(*DPLSUBSET) (RESP2=1)
- STATUS has an incorrect CVDA value (RESP2=2)
- NEWCOPY was specified and RESCOUNT is not equal to zero (RESP2=3)
- COPY has an incorrect CVDA value (RESP2=5)
- Copy was specified for a program currently loaded with the HOLD option (RESP2=6)
- CEDFSTATUS has an incorrect CVDA value (RESP2=9)
- The named program is remote (RESP2=17)
- EXECUTIONSET was specified for a BMS map set (RESP2=18)
- EXECUTIONSET has an incorrect CVDA value (RESP2=20)

Default action: Terminate the task abnormally.

IOERR

occurs if COPY was specified and the load process failed (RESP2=8).

Default action: Terminate the task abnormally.

NOTAUTH

occurs if the control region is not authorized for the program object (RESP2=101).

Default action: Terminate the task abnormally.

PGMIDERR

occurs if the named program cannot be found (RESP2=7).

Default action: Terminate the task abnormally.

SET SYSTEM



Condition: INVREQ

The EXEC CICS SET SYSTEM command changes some of the information associated with the CICS control region.

Options

DUMPING(cvda)

causes the taking of CICS control region dumps to be globally suppressed or not. The input CVDA values are:

SYSDUMP CICS control region dumps are not globally suppressed

NOSYSDUMP CICS control region dumps are globally suppressed.

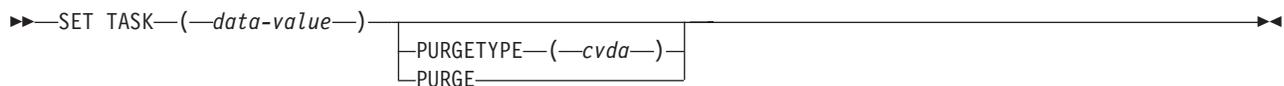
Exception Conditions

INVREQ

occurs if DUMPING has an incorrect CVDA value (RESP2=9).

Default action: Terminate the task abnormally.

SET TASK



Conditions: INVREQ, TASKIDERR

The EXEC CICS SET TASK command allows a named task to be purged.

Options

PURGETYPE(cvda)

causes the specified task to be purged. The input CVDA value is:

PURGE

The task is terminated when system and data integrity can be maintained.

A task is not purged if its associated transaction definition had specified *NO in the PURGE parameter within the OS/400 ADDCICSPCT CL command.

TASK(data-value)

is the 4-byte packed decimal sequence number that identifies the task.

Exception Conditions

INVREQ

occurs if:

- PURGETYPE has an incorrect CVDA value (RESP2=3)
- The named task is not in a valid state for purging (RESP2=5)

Default action: Terminate the task abnormally.

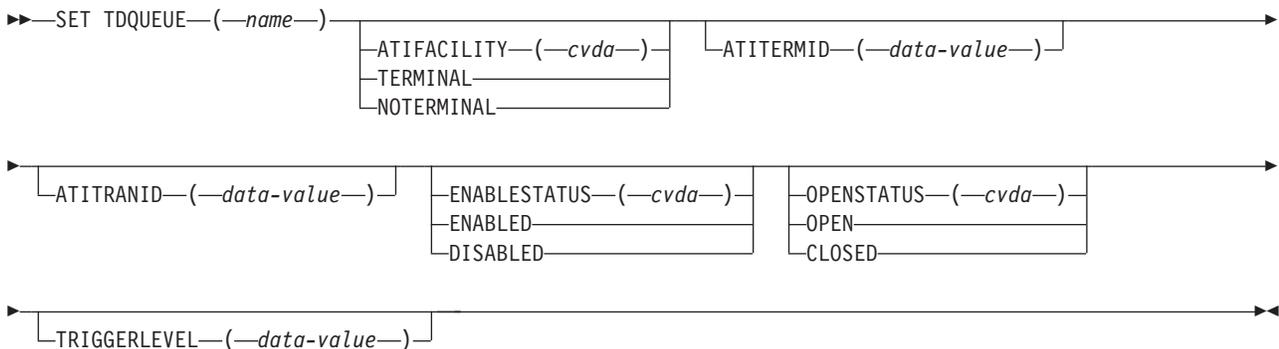
TASKIDERR

occurs if:

- The named task cannot be found (RESP2=1)
- The named task is protected by CICS; that is, it is a CICS-supplied transaction that is normally invoked from within CICS and not by an operator

Default action: Terminate the task abnormally.

SET TDQUEUE



Conditions: INVREQ, IOERR, QIDERR

The EXEC CICS SET TDQUEUE command changes some of the information of a named transient data queue definition in the control region. The queue definition must not be remote or indirect.

Options

ATIFACILITY(cvda) (intrapartition queues only)

causes the specified queue to associate, or not, a terminal (or session) with the task to be started when the TRIGGERLEVEL value is reached. The input CVDA values are:

TERMINAL A terminal (or session) is associated with the task.

NOTERMINAL

A terminal (or session) is not associated with the task.

ATITERMID(data-value) (intrapartition queues only)

causes the specified queue to use the value as the terminal (or session) that is associated with the task to be started when the TRIGGERLEVEL value is reached.

ATITRANID(data-value) (intrapartition queues only)

causes the specified queue to use the value as the transaction that is associated with the task to be started when the TRIGGERLEVEL value is reached.

ENABLESTATUS(cvda)

causes the specified queue to allow or not allow use by application programs. The input CVDA values are:

ENABLED The queue is available for use by application programs.

DISABLED The queue is unavailable for use by application programs, though it may still be OPEN. Any queue whose name begins with the letter "C" cannot be disabled, because this identifies CICS's own queues.

OPENSTATUS(cvda) (extrapartition queues only)

causes the specified queue to be opened or closed. The input CVDA values are:

OPEN The queue is to be opened.

CLOSED The queue is to be closed.

Note: A queue that is disabled cannot be opened or closed.

TDQUEUE(name)

is the name of the queue as specified in the DEST parameter within the OS/400 ADDCICSDCT CL command. The name may be up to 4 characters long.

TRIGGERLEVEL(data-value) (intrapartition queues only)

specifies as a fullword binary value the number of records that can be written to the queue before automatic transaction initiation (ATI) occurs. The number must be in the range 0 through 32 767.

Exception Conditions

INVREQ

occurs if:

- TRIGGERLEVEL was specified for an extrapartition queue (RESP2=2)
- TRIGGERLEVEL value is not in the range 0–32767 (RESP2=3)
- ATITERMID was specified for an extrapartition queue (RESP2=4)
- ATITRANID was specified for an extrapartition queue (RESP2=5)
- ATIFACILITY was specified for an extrapartition queue (RESP2=6)
- ATIFACILITY was specified with an invalid CVDA value (RESP2=7)
- OPENSTATUS was specified with an invalid CVDA value (RESP2=8)
- OPENSTATUS was specified for an intrapartition queue (RESP2=9)
- ENABLESTATUS was specified with an invalid CVDA value (RESP2=10)
- DISABLED was specified for a queue whose name begins with "C" (RESP2=11)
- The named queue is remote (RESP2=12)
- The named queue is indirect (RESP2=13)
- OPEN or CLOSED was specified for a DISABLED queue (RESP2=15)
- TRIGGERLEVEL is specified but the ATI transaction does not exist (RESP2=17)

Default action: Terminate the task abnormally.

IOERR

occurs if an error arises while opening or closing the queue (RESP2=14).

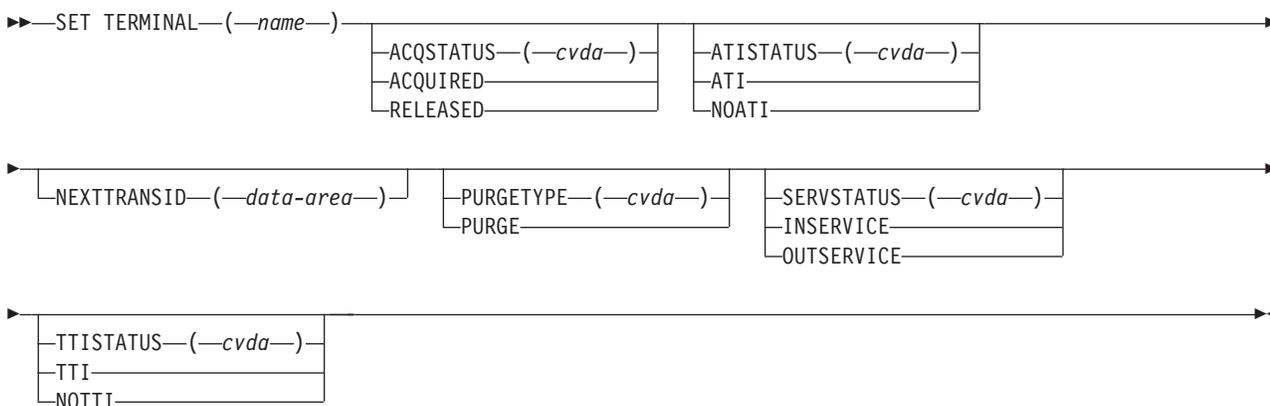
Default action: Terminate the task abnormally.

QIDERR

occurs if the named queue cannot be found (RESP2=1).

Default action: Terminate the task abnormally.

SET TERMINAL



Conditions: INVREQ, TERMIDERR

The EXEC CICS SET TERMINAL command changes some of the information of a named terminal definition.

Options

ACQSTATUS(cvda)

causes the specified terminal to be associated or not associated with the logical unit represented by the named terminal. ACQSTATUS cannot be specified for APPC sessions or TCS entries. The input CVDA values are:

ACQUIRED The terminal is associated with the logical unit represented by the named terminal.

RELEASED The terminal is not associated with the logical unit represented by the named terminal.

This happens immediately if the PURGE option is also specified. If PURGE is not specified, this happens when the current active task has finished.

ATISTATUS(cvda)

causes the specified terminal to be used or not used by transactions that are automatically initiated from within CICS. The input CVDA values are:

ATI The terminal can be used by transactions that are automatically initiated from within CICS.

NOATI

The terminal cannot be used by transactions that are automatically initiated from within CICS.

Note: A terminal cannot have both NOATI and NOTTI specified.

NEXTTRANSID(data-area)

causes the specified terminal to use this value as the next CICS transaction to be started. If this is set to blanks (X'40404040'), the next transaction about to start is nulled.

PURGETYPE(cvda)

causes the specified terminal to purge the associated task. The input CVDA value is:

PURGE The task is terminated when the CICS system and data integrity can be maintained.

A task is not purged if its associated transaction definition had specified *NO in the PURGE parameter within the OS/400 ADDCICSPCT CL command.

SERVSTATUS(cvda)

specifies whether the terminal is to be available for use. Valid CVDA values are:

INSERVICE The terminal is to be available for use.

OUTSERVICE The terminal is not to be available for use.

Unless PURGE is also specified, the current transaction is allowed to terminate normally, but no further transactions are allowed to use the terminal.

The terminal is RELEASED and the user is signed off, either immediately or when the current transaction has terminated. You cannot therefore set the terminal associated with the executing transaction to OUTSERVICE, unless it is a printer.

TERMINAL(name)

is the name of the terminal as specified in the CICSDEV parameter within the OS/400 ADDCICSTCT CL command. The name can be up to 4 characters long.

TTISTATUS(cvda)

specifies whether transactions may be started from this terminal. The input CVDA values are:

TTI Transactions may be started from the terminal.

NOTTI

Transactions may not be started from the terminal. Transactions may be started using either ATI or an EXEC CICS START command.

Note: A terminal cannot have both NOATI and NOTTI in its status.

Exception Conditions

INVREQ

occurs if:

- ACQSTATUS has an incorrect CVDA value (RESP2=2)
- ATISTATUS has an incorrect CVDA value (RESP2=4)
- ATISTATUS change would result in NOATI and NOTTI (RESP2=5)
- An attempt has been made to put the terminal associated with the executing transaction OUTSERVICE, and it is not a printer (RESP2=11)
- SERVSTATUS has an incorrect CVDA value (RESP2=13)

- TTISTATUS change would result in NOATI and NOTTI (RESP2=17)
- TTISTATUS has an incorrect CVDA value (RESP2=18)
- PURGETYPE has been specified for the transaction on this terminal (RESP2=20)
- PURGETYPE has an incorrect CVDA value (RESP2=21)
- The named terminal is a remote terminal (RESP2=24)
- ACQUIRED has been specified but the terminal is not in service (RESP2=25)
- PURGE has been specified but the target task has PURGE=NO in its associated transaction definition (RESP2=26)
- A permanent transaction has been defined for this terminal (using the TRANSID parameter in the TCT definition) (RESP2=34). See *CICS for iSeries Administration and Operations Guide* for details.
- RELEASED has been specified but the terminal is in the process of being acquired (RESP2=37)
- ACQUIRED has been specified but the terminal is in the process of being released (RESP2=38)

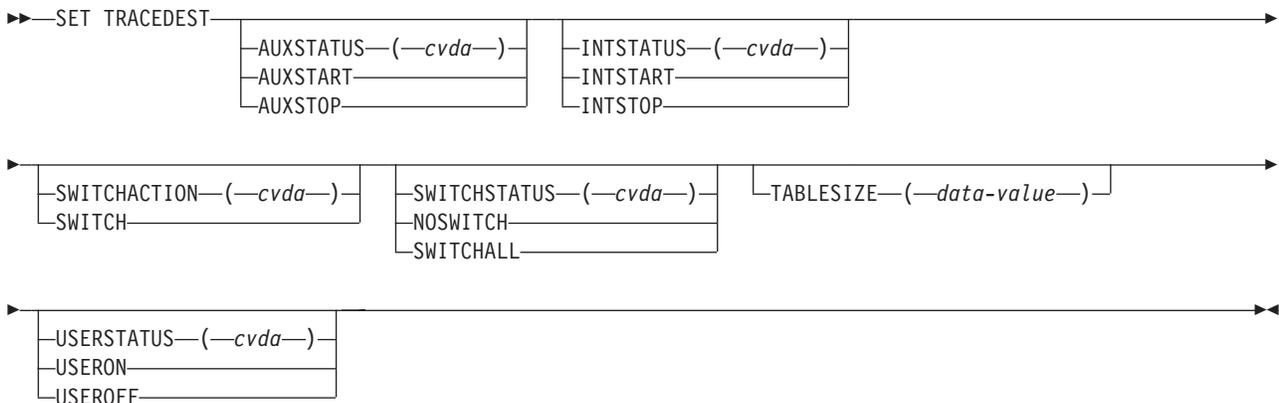
Default action: Terminate the task abnormally.

TERMIDERR

occurs if the named terminal cannot be found (RESP2=23).

Default action: Terminate the task abnormally.

SET TRACEDEST



Conditions: INVREQ, IOERR, NOSPACE

The EXEC CICS SET TRACEDEST command allows you to change some of the information about the recording of trace entries in the control region.

Options

AUXSTATUS(cvda)

specifies the auxiliary tracing status required in the CICS system. CICS/400 auxiliary trace entries are made to a pair of CICS/400 controlled user spaces. Valid CVDA values are:

AUXSTART Auxiliary tracing is to be started.

AUXSTOP Auxiliary tracing is to be stopped. A subsequent AUXSTART request causes the new trace entries to be written at the start of the user space, thereby overwriting the trace entries that were written before the AUXSTOP request.

INTSTATUS(cvda)

causes the CICS/400 internal tracing to be active or not in the CICS/400 control region. The input CVDA values are:

INTSTART The CICS/400 internal tracing is to be started.

INTSTOP The CICS/400 internal tracing is to be stopped.

SWITCHACTION(cvda)

causes the CICS/400 auxiliary trace user space to be switched. The input CVDA value is SWITCH.

SWITCH Switches immediately from the current CICS/400 auxiliary trace user space to the alternate CICS/400 auxiliary trace user space.

SWITCHSTATUS(cvda)

causes the CICS/400 auxiliary trace to allow or not allow automatic user space switching when the current CICS/400 auxiliary trace user space becomes full in the CICS/400 control region.

The input CVDA values are:

NOSWITCH

Switching is not allowed without operator intervention.

SWITCHALL

Automatic switching is allowed as necessary until the end of the CICS/400 control region, without the need for operator intervention.

TABLESIZE(data-value)

causes the CICS/400 internal trace table to allow this value as the maximum number of entries that can be kept in the table.

If you specify a value that is different from the current trace table size, CICS/400 internal tracing stops while the change is made; a new table of the requested size is obtained, and the old one is freed. Data that was in the old table is lost.

USERSTATUS(cvda)

causes the CICS/400 user tracing to be active or not in the CICS/400 control region. Valid CVDA values are:

USERON The CICS/400 user tracing is on.

USEROFF The CICS/400 user tracing is off.

Exception Conditions

INVREQ

occurs if:

- INTSTATUS has an incorrect CVDA value (RESP2=1)
- A TABLESIZE value outside the range 125-10 000 has been specified (RESP2=2)
- AUXSTATUS has an incorrect CVDA value (RESP2=3)
- SWITCHSTATUS has an incorrect CVDA value (RESP2=4)
- SWITCHACTION has an incorrect CVDA value (RESP2=11)

- USERSTATUS has an incorrect CVDA value (RESP2=12)

Default action: Terminate the task abnormally.

IOERR

occurs if:

- A SWITCH request or an EXEC CICS SET TRACEDEST AUXSTART request resulted in an OPEN error for the trace user space (RESP2=10)
- An INTSTART request resulted in an OPEN error for the trace user space (RESP2=13)

Default action: Terminate the task abnormally.

NOSPACE

occurs if there is insufficient space for the new trace table (RESP2=7).

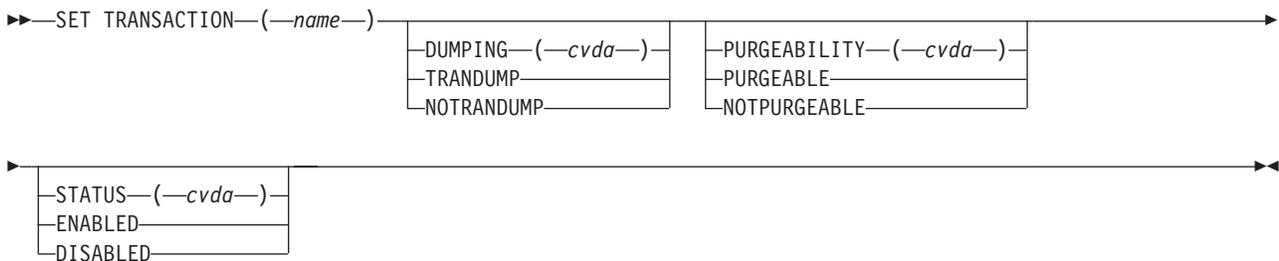
Default action: Terminate the task abnormally.

Example

```
EXEC CICS SET TRACEDEST
        SWITCH
        NOSWITCH
END-EXEC.
```

The SWITCH option tells the CICS control region to switch now from the current CICS auxiliary trace user space (which is not necessarily full) to the alternate CICS auxiliary trace user space. The NOSWITCH option states that there is to be no automatic switching of CICS auxiliary trace user space when it fills up.

SET TRANSACTION



Conditions: INVREQ, TRANSIDERR

The EXEC CICS SET TRANSACTION command allows you to change some of the information of a named transaction definition.

Options

DUMPING(cvda)

causes the specified transaction to allow or not allow a CICS/400 transaction dump when the transaction terminates abnormally. The setting of this option is ignored when an explicit DUMP TRANSACTION command is issued. The input CVDA values are:

TRANDUMP A CICS/400 transaction dump occurs when the transaction terminates abnormally.

NOTRANDUMP

A CICS transaction dump does not occur when the transaction terminates abnormally.

PURGEABILITY(cvda)

specifies whether the transaction is to be treated as purgeable or nonpurgeable in system stall conditions. Valid CVDA values are:

PURGEABLE The transaction can be purged.

NOTPURGEABLE

The transaction cannot be purged.

STATUS(cvda)

specifies whether the transaction is to be available for use. Valid CVDA values are:

ENABLED The transaction is available for use.

DISABLED The transaction is not available for use. CICS transactions beginning with the letter "C" are supplied by CICS and cannot be disabled.

TRANSACTION(name)

is the name of the transaction specified in the TRANSID parameter of the OS/400 ADDCICSPCT CL command. The name may be up to 4 characters long.

Exception Conditions

INVREQ

occurs if:

- PURGEABILITY has an incorrect CVDA value (RESP2=2)
- STATUS has an incorrect CVDA value (RESP2=3)
- An attempt has been made to disable a CICS-supplied transaction (RESP2=4)
- DUMPING has an incorrect CVDA value (RESP2=8)
- The PCT profile is not available (RESP2=10)

Default action: Terminate the task abnormally.

TRANSIDERR

occurs if the named transaction cannot be found (RESP2=1).

Default action: Terminate the task abnormally.

Part 8. Appendixes

Appendix A. EXEC interface block

This appendix describes the fields of the EXEC interface block (EIB). An application program can read all the fields in the EIB of the associated task by name, but must not change the contents of any of them.

For each field, the contents and format, for each of the supported application programming languages, are given. All fields contain zeros (X'00') in the absence of meaningful information. Fields are listed in alphabetic order.

EIB fields

EIBAID

contains the attention identifier (AID) associated with the last terminal control or basic mapping support (BMS) input operation from a display device such as a terminal. See "Attention identifier constants, DFHAID" on page 548.

```
COBOL: PIC X(1).  
C:      char eibaid;
```

EIBATT

indicates that the request/response unit (RU) contains attach header data (X'FF').

```
COBOL: PIC X(1).  
C:      char eibatt;
```

EIBCALEN

contains the length of the communication area that has been passed to the application program from the last program, using the COMMAREA and LENGTH options. If no communication area is passed, this field contains zeros.

```
COBOL: PIC S9(4) BINARY.  
C:      short int eibcalen;
```

EIBCOMPL

indicates, on a terminal control RECEIVE command, that the data is complete (X'FF'). If the NOTRUNCATE option has been used on the RECEIVE command, CICS retains data in excess of the amount requested via the LENGTH or MAXLENGTH option. EIBRECV is set indicating that further RECEIVE commands are required. EIBCOMPL is not set until the last of the data has been retrieved.

EIBCOMPL is always set when a RECEIVE command without the NOTRUNCATE option is executed.

```
COBOL: PIC X(1).  
C:      char eibcompl;
```

EIBCONF

indicates that a CONFIRM request has been received on an APPC conversation (X'FF').

```
COBOL: PIC X(1).  
C:      char eibconf;
```

EIBCPASN

contains the cursor address (position) associated with the last terminal control or basic mapping support (BMS) input operation from a display device such as a terminal.

EIB fields

```
COBOL: PIC S9(4) BINARY.  
C:      short int eibcposn;
```

EIBDATE

contains the date the task is started; this field is updated by the ASKTIME command. The date is in packed decimal form (0CYYDDD+) where C shows the century with values 0 for the 1900s and 1 for the 2000s.

```
COBOL: PIC S9(7) PACKED-DECIMAL.  
C:      char eibdate[4];
```

EIBDS

contains the symbolic identifier of the last file referred to in a file control request.

```
COBOL: PIC X(8).  
C:      char eibds[8];
```

EIBEOC

is always set to X'FF' after a RECEIVE command, (to indicate an end-of-chain) for compatibility with existing CICS programs.

```
COBOL: PIC X(1).  
C:      char eibeoc;
```

EIBERR

indicates that an error has been received on an APPC conversation (X'FF'). The error code is in EIBERRCD.

```
COBOL: PIC X(1).  
C:      char eiberr;
```

EIBERRCD

when EIBERR is set, contains the error code that has been received.

The values that can be returned in the first two bytes of EIBERRCD are given in *CICS for iSeries Intercommunication*.

See *SNA Format Protocol Reference Architecture Logic for LU Type 6.2, SC30-3269*, for information about other EIBERRCD values that can occur.

```
COBOL: PIC X(4).  
C:      char eiberrcd[4];
```

EIBFMH

indicates that the user data just received or retrieved contains an FMH (X'FF').

```
COBOL: PIC X(1).  
C:      char eibfmh;
```

EIBFN

contains a code that identifies the last CICS command to be issued by the task; this field is updated when the requested function has been completed.

```
COBOL: PIC X(2).  
C:      char eibfn[2];
```

EIBFN code sequence

This list presents EIBFN codes in numeric sequence by hexadecimal value; it is followed by a list in alphabetic sequence by command name.

Hex code	Command
0202	ADDRESS
0204	HANDLE CONDITION
0206	HANDLE AID

Hex code	Command
0208	ASSIGN
020A	IGNORE CONDITION
020C	PUSH HANDLE
020E	POP HANDLE
0402	RECEIVE
0404	SEND
0406	CONVERSE
0418	ISSUE ERASEAUP
041E	ISSUE SIGNAL
0420	ALLOCATE
0422	FREE
042C	WAIT CONVID
042E	EXTRACT PROCESS
0430	ISSUE ABEND
0432	CONNECT PROCESS
0434	ISSUE CONFIRMATION
0436	ISSUE ERROR
043E	EXTRACT ATTRIBUTES
0602	READ
0604	WRITE
0606	REWRITE
0608	DELETE
060A	UNLOCK
060C	STARTBR
060E	READNEXT
0610	READPREV
0612	ENDBR
0614	RESETBR
0802	WRITEQ TD
0804	READQ TD
0806	DELETEQ TD
0A02	WRITEQ TS
0A04	READQ TS
0A06	DELETEQ TS
0C02	GETMAIN
0C04	FREEMAIN
0E02	LINK
0E04	XCTL
0E06	LOAD
0E08	RETURN
0E0A	RELEASE
0E0C	ABEND
0E0E	HANDLE ABEND
1002	ASKTIME
1004	DELAY
1006	POST
1008	START
100A	RETRIEVE
100C	CANCEL
1202	WAIT EVENT
1204	ENQ
1206	DEQ
1208	SUSPEND
1402	WRITE JOURNALNUM

EIB fields

Hex code	Command
1404	WAIT JOURNALNUM
1602	SYNCPOINT
1802	RECEIVE MAP
1804	SEND MAP
1806	SEND TEXT
1812	SEND CONTROL
2002	BIF DEEDIT
4204	INQUIRE AUTINSTMODEL
4210	DISCARD AUTINSTMODEL
4802	ENTER TRACENUM
4A02	ASKTIME ABSTIME
4A04	FORMATTIME
4C02	INQUIRE FILE
4C04	SET FILE
4C10	DISCARD FILE
4E02	INQUIRE PROGRAM
4E04	SET PROGRAM
4E10	DISCARD PROGRAM
5002	INQUIRE TRANSACTION
5004	SET TRANSACTION
5010	DISCARD TRANSACTION
5202	INQUIRE TERMINAL
5204	SET TERMINAL
5402	INQUIRE SYSTEM
5404	SET SYSTEM
5602	SPOOLOPEN OUTPUT
5606	SPOOLWRITE
5610	SPOOLCLOSE
5802	INQUIRE CONNECTION
5804	SET CONNECTION
5C02	INQUIRE TDQUEUE
5C04	SET TDQUEUE
5E02	INQUIRE TASK
5E04	SET TASK
6002	INQUIRE JOURNALNUM
6004	SET JOURNALNUM
7602	PERFORM SHUTDOWN
7802	INQUIRE TRACEDEST
7804	SET TRACEDEST
7E02	DUMP TRANSACTION

Command name sequence

This list presents EIBFN codes in alphabetic sequence by command name.

Hex	code
0E0C	ABEND
0202	ADDRESS
0420	ALLOCATE
1002	ASKTIME
4A02	ASKTIME ABSTIME
0208	ASSIGN

Hex	code
100C	CANCEL
0432	CONNECT PROCESS
0406	CONVERSE
1004	DELAY
0608	DELETE
0806	DELETEQ TD
0A06	DELETEQ TS
1206	DEQ
4210	DISCARD AUTINSTMODEL
4C10	DISCARD FILE
4E10	DISCARD PROGRAM
5010	DISCARD TRANSACTION
7E02	DUMP TRANSACTION
0612	ENDBR
1204	ENQ
4802	ENTER TRACENUM
043E	EXTRACT ATTRIBUTES
042E	EXTRACT PROCESS
4A04	FORMATTIME
0422	FREE
0C04	FREEMAIN
0C02	GETMAIN
0E0E	HANDLE ABEND
0206	HANDLE AID
0204	HANDLE CONDITION
020A	IGNORE CONDITION
4202	INQUIRE AUTINSTMODEL
5802	INQUIRE CONNECTION
4C02	INQUIRE FILE
6002	INQUIRE JOURNALNUM
4E02	INQUIRE PROGRAM
5402	INQUIRE SYSTEM
5E02	INQUIRE TASK
5C02	INQUIRE TDQUEUE
5202	INQUIRE TERMINAL
7802	INQUIRE TRACEDEST
5002	INQUIRE TRANSACTION
0430	ISSUE ABEND
0434	ISSUE CONFIRMATION
0418	ISSUE ERASEAUP
0436	ISSUE ERROR
0438	ISSUE PREPARE
041E	ISSUE SIGNAL
0E02	LINK
0E06	LOAD
7602	PERFORM SHUTDOWN
020E	POP HANDLE
1006	POST
020C	PUSH HANDLE
0602	READ
060E	READNEXT
0610	READPREV
0804	READQ TD
0A04	READQ TS

EIB fields

Hex	code
0402	RECEIVE
1802	RECEIVE MAP
0E0A	RELEASE
0614	RESETBR
100A	RETRIEVE
0E08	RETURN
0606	REWRITE
0404	SEND
1812	SEND CONTROL
1804	SEND MAP
1806	SEND TEXT
5804	SET CONNECTION
4C04	SET FILE
6004	SET JOURNALNUM
4E04	SET PROGRAM
5404	SET SYSTEM
5E04	SET TASK
5C04	SET TDQUEUE
5204	SET TERMINAL
7804	SET TRACEDEST
5004	SET TRANSACTION
5610	SPOOLCLOSE
5602	SPOOLOPEN OUTPUT
5606	SPOOLWRITE
1008	START
060C	STARTBR
1208	SUSPEND
1602	SYNCPOINT
060A	UNLOCK
042C	WAIT CONVID
1202	WAIT EVENT
1404	WAIT JOURNALNUM
0604	WRITE
1402	WRITE JOURNALNUM
0802	WRITEQ TD
0A02	WRITEQ TS
0E04	XCTL

EIBFREE

indicates that the application program cannot continue using the facility (X'FF'). The application program should either free the facility explicitly or terminate, so that the facility is freed by CICS.

```
COBOL: PIC X(1).  
C:      char eibfree;
```

EIBNODAT

indicates that no data has been sent by the remote application (X'FF'). A message has been received from the remote system that conveyed only control information. For example, if the remote application executed a SEND command with the WAIT option, or a WAIT CONVID command, any data would be sent across the link. If the remote application then executed a SEND INVITE command without using the FROM option to transmit data at the same time, it would be necessary to send the INVITE instruction across the

link by itself. In this case, the receiving application finds EIBNODAT set. The use of this field is restricted to application programs holding conversations across APPC links only.

```
COBOL: PIC X(1).
C:      char eibnodat;
```

EIBRCODE

contains the CICS response code returned after the function requested by the last CICS command to be issued by the task has been completed.

Note: Each possible value of EIBRESP relates directly to a specific exception condition, no matter which command caused the condition to be raised. This is not true for EIBRCODE values; in general, both the value and the byte of EIBRCODE in which it is set depend on which command was issued.

For a normal response, this field contains hexadecimal zeros (6X'00').

```
COBOL: PIC X(6).
C:      char eibrancode[6];
```

The following list shows the EIBRCODE values corresponding to the exception conditions that can occur for each group of commands (as indicated by byte 0 of EIBFN), together with the names of the associated conditions.

For some conditions, further information is provided in EIBRCODE and this is indicated by a note following the list of values.

Note: For some commands (for example, DISCARD, INQUIRE and SET), byte 3 of EIBRCODE contains the hexadecimal equivalent of the value in EIBRESP; the remaining bytes are set to X'00'. Any further information relating to conditions occurring on these commands can be found in EIBRESP2 rather than EIBRCODE. The RESP2 values are given in the descriptions of the individual commands.

EIBFN	EIBRCODE	Condition
02 ..	E0	INVREQ
04 ..	D0	SYSIDERR ¹
04 ..	D3	SYSBUSY ²
04 ..	D5	NOTALLOC
04 ..	E0	INVREQ ³
04 ..	E1	LENGERR ⁴
04 ..	E5	SIGNAL
04 ..	F1	TERMERR
04 20	EOC
06 ..	01	FILENOTFOUND
06 ..	02	ILLOGIC ⁵
06 ..	08	INVREQ
06 ..	0C	NOTOPEN
06 ..	0D	DISABLED
06 ..	0F	ENDFILE
06 ..	02	IOERR ⁵
06 ..	81	NOTFND
06 ..	82	DUPREC
06 ..	83	NOSPACE
06 ..	84	DUPKEY
06 ..	D0	SYSIDERR ¹
06 ..	D1	ISCINVREQ
06 ..	D6	NOTAUTH

EIB fields

EIBFN	EIBRCODE	Condition
06 ..	E1	LENGERR
08 ..	01	QZERO
08 ..	02	QIDERR
08 ..	04	IOERR
08 ..	08	NOTOPEN
08 ..	10	NOSPACE
08 ..	C0	QBUSY
08 ..	D0	SYSIDERR ¹
08 ..	D1	ISCINVREQ
08 ..	D6	NOTAUTH
08 ..	D7	DISABLED
08 ..	E0	INVREQ
08 ..	E1	LENGERR
0A ..	01	ITEMERR
0A ..	02	QIDERR
0A ..	04	IOERR
0A ..	08	NOSPACE
0A ..	20	INVREQ
0A ..	D0	SYSIDERR ¹
0A ..	D1	ISCINVREQ
0A ..	D6	NOTAUTH
0A ..	E1	LENGERR
0C ..	E1	LENGERR
0C ..	E2	NOSTG
0E ..	01	PGMIDERR
0E ..	D6	NOTAUTH
0E ..	E0	INVREQ
10 ..	01	ENDDATA
10 ..	04	IOERR
10 ..	11	TRANSIDERR
10 ..	12	TERMIDERR
10 ..	20	EXPIRED
10 ..	81	NOTFND
10 ..	D0	SYSIDERR ¹
10 ..	D1	ISCINVREQ
10 ..	D6	NOTAUTH
10 ..	E1	LENGERR
10 ..	E9	ENVDEFERR
10 ..	FF	INVREQ
12 ..	32	ENQBUSY
12 ..	E0	INVREQ
12 ..	E1	LENGERR
14 ..	01	JIDERR
14 ..	05	NOTOPEN
14 ..	06	LENGERR
14 ..	07	IOERR
14 ..	09	NOJBUFSP
14 ..	D6	NOTAUTH
16 ..	01	ROLLEDBACK
18 ..	01	INVREQ
18 ..	04	MAPFAIL
18 ..	08	INVMPSZ ⁶
42 10 ..	INVREQ
42 15 ..	ILLOGIC

EIBFN	EIBRCODE	Condition
42 53 ..	END
42 5F ..	MODELIDERR
48	INVREQ
48	LENGERR
4A 10 ..	INVREQ
4C 0C ..	FILENOTFOUND
4C 10 ..	INVREQ
4C 11 ..	IOERR
4C 15 ..	ILLOGIC
4C 53 ..	END
4E 10 ..	INVREQ
4E 11 ..	IOERR
4E 15 ..	ILLOGIC
4E 1B ..	PGMIDERR
4E 46 ..	NOTAUTH
4E 53 ..	END
50 10 ..	INVREQ
50 15 ..	ILLOGIC
50 1C ..	TRANSIDERR
50 53 ..	END
52 0B ..	TERMIDERR
52 10 ..	INVREQ
52 15 ..	ILLOGIC
52 53 ..	END
54 10 ..	INVREQ
56 0D ..	NOTFND
56 10 ..	INVREQ
56 13 ..	NOTOPEN
56 16 ..	LENGERR
56 57 ..	OPENERR
56 58 ..	SPOLBUSY
56 59 ..	SPOLERR
58 10 ..	INVREQ
58 15 ..	ILLOGIC
58 35 ..	SYSIDERR
58 53 ..	END
5C 10 ..	INVREQ
5C 11 ..	IOERR
5C 15 ..	ILLOGIC
5C 2C ..	QIDERR
5C 53 ..	END
5E 10 ..	INVREQ
5E 5B ..	TASKIDERR
60 10 ..	INVREQ
60 11 ..	IOERR
60 15 ..	ILLOGIC
60 2B ..	JIDERR
60 53 ..	END
76 10 ..	INVREQ
78 10 ..	INVREQ
78 12 ..	IOERR
78 12 ..	NOSPACE
7E	INVREQ

Notes:

1. When SYSIDERR occurs, further information is provided in bytes 1 and 2 of EIBRCODE as follows:

```

.. 04 00 .. .. . requested function not valid
.. 04 04 .. .. . no session available and
                    NOQUEUE was specified
.. 04 08 .. .. . mode name not founda
.. 04 0C .. .. . mode name not valida
.. 04 10 .. .. . task canceled or timed out
                    during allocationa
.. 04 14 .. .. . mode group out of servicea
.. 04 18 .. .. . close - DRAIN=ALLa
.. 08 .. .. . sysid out of service
.. 0C xx .. .. . sysid definition error
.. 0C 00 .. .. . name not that of TCTSE
.. 0C 04 .. .. . name not that of remote
                    TCTSE
.. 0C 08 .. .. . mode name not found
.. 0C 0C .. .. . profile not found

```

^a Applies to APPC only.

2. If SYSBUSY occurs on an ALLOCATE command that attempts to acquire a session to an APPC terminal or system, byte 3 of EIBRCODE indicates where the error condition was detected:

```

.. . . . 00 .. . the request was for a
                    session to a connected
                    terminal or system
.. . . . 01 .. . the request was for a
                    session to a remotely
                    connected terminal or
                    system, and the error
                    occurred in the terminal-
                    owning region (TOR) or
                    an intermediate system
.. . . . 02 .. . the request was for a
                    session to a remotely
                    connected terminal or
                    system, and the error
                    occurred in the
                    application-owning
                    region (AOR)

```

3. When INVREQ occurs during APPC conversations, further information is provided in byte 3 of EIBRCODE as follows:

```

.. . . . 04 .. . ALLOCATE command -
                    conversation
                    already allocated
.. . . . 08 .. . FREE command -
                    conversation in
                    wrong state
.. . . . 0C .. . CONNECT PROCESS
                    command - sync-
                    level 2 has been
                    requested, but
                    cannot be supported
                    on the conversation
                    in use
.. . . . 14 .. . SEND command -
                    CONFIRM option has
                    been specified,
                    but conversation
                    is not sync-level
                    1 or 2
.. . . . 1C .. . an incorrect
                    command has been

```

- issued for the terminal or logical unit in use
 20 an incorrect command has been issued for the APPC conversation type in use
4. When LENGERR occurs during terminal control operations, further information is provided in byte 1 of EIBRCODE as follows:
- .. 00 input data is overlong and has been truncated
 .. 04 on output commands, an incorrect (FROM)LENGTH has been specified, either less than zero or greater than 32767
 .. 08 on input commands, an incorrect (TO)LENGTH has been specified, greater than 32767
5. When ILOGIC or IOERR occurs during file control operations, the OS/400 data management routines provide CICS with the OS/400 message number if there is one.

CICS converts the message number into a format acceptable in EIBRCODE. The conversion is as follows:

- Byte 0 contains the EIBRCODE; 02 - ILOGIC or 80 - IOERR.
- Bytes 1 and 2 contain the message number, if any, in hexadecimal.
- Bytes 3 and 4 both contain:
 - X'00' for CPF
 - X'01' for MCH
 - X'02' for other
- Byte 5 is not used.

For example, if an I/O error is trapped and the system sends out the message CPF5006, this would appear in EIBRCODE as follows:

```
xx 50 06 00 00 ..
```

where byte 0 contains the EIBRCODE for the abend, bytes 1 and 2 contain the message number X'5006', bytes 3 and 4 both contain X'00', designating CPF, and byte 5 is not used.

ILOGIC can occur when you attempt to access an *ESDS file by RRN or an *REL file by RBA.

6. When INVMPSZ occurs during BMS operations, byte 3 of EIBRCODE contains the terminal code:

```
.. .. . xx .. .. terminal code
```

These are the same as the map set suffixes shown in Table 16 on page 560.

EIBRECV

indicates that the application program is to continue receiving data from the facility by executing RECEIVE commands (X'FF').

EIB fields

```
COBOL: PIC X(1).  
C:      char eibrecv;
```

EIBREQID

contains the request identifier assigned to an interval control command by CICS; this field is not used when a request identifier is specified in the application program.

```
COBOL: PIC X(8).  
C:      char eibreqid[8];
```

EIBRESP

contains a fullword binary number corresponding to the condition that has occurred. These numbers are listed below (in decimal) for the conditions that can occur on local requests during execution of EXEC CICS commands. **The abend code (if any) associated with each condition is also presented.**

```
COBOL: PIC S9(8) BINARY.  
C:      long int eibresp;
```

EIBRESP code sequence

This list presents conditions and their associated abend codes (if any) in EIBRESP code sequence; it is followed by a list in abend code sequence.

No.	Condition	Abend
00	NORMAL	
01	ERROR	AEIA
06	EOC	
11	TERMIDERR	AEIK
12	FILENOTFOUND	AEIL
13	NOTFND	AEIM
14	DUPREC	AEIN
15	DUPKEY	AEIO
16	INVREQ	AEIP
17	IOERR	AEIQ
18	NOSPACE	AEIR
19	NOTOPEN	AEIS
20	ENDFILE	AEIT
21	ILLOGIC	AEIU
22	LENGERR	AEIV
23	QZERO	AEIW
24	SIGNAL	
25	QBUSY	
26	ITEMERR	AEIZ
27	PGMIDERR	AEI0
28	TRANSIDERR	AEI1
29	ENDDATA	AEI2
31	EXPIRED	
36	MAPFAIL	AEI9
38	INVMPSZ	AEYB
42	NOSTG	ASCP
43	JIDERR	AEYG
44	QIDERR	AEYH
45	NOJBUFSP	A17G
53	SYSIDERR	AEYQ
54	ISCINVREQ	AEYR
55	ENQBUSY	

No.	Condition	Abend
56	ENVDEFERR	AEYT
59	SYSBUSY	
61	NOTALLOC	AEYY
70	NOTAUTH	AEY7
81	TERMERR	ATNI
82	ROLLEDBACK	AEXJ
83	END	AEXK
84	DISABLED	AEXL
87	OPENERR	
88	SPOLBUSY	
89	SPOLERR	
91	TASKIDERR	AEXX
95	MODELIDERR	AEX3

Abend code sequence

This list presents conditions and their associated abend codes in abend code sequence.

Conditions without corresponding abend codes are not shown.

No.	Condition	Abend
01	ERROR	AEIA
11	TERMIDERR	AEIK
12	FILENOTFOUND	AEIL
13	NOTFND	AEIM
14	DUPREC	AEIN
15	DUPKEY	AEIO
16	INVREQ	AEIP
17	IOERR	AEIQ
18	NOSPACE	AEIR
19	NOTOPEN	AEIS
20	ENDFILE	AEIT
21	ILLOGIC	AEIU
22	LENGERR	AEIV
23	QZERO	AEIW
26	ITEMERR	AEIZ
27	PGMIDERR	AEI0
28	TRANSIDERR	AEI1
29	ENDDATA	AEI2
36	MAPFAIL	AEI9
82	ROLLEDBACK	AEXJ
83	END	AEXK
84	DISABLED	AEXL
91	TASKIDERR	AEXX
95	MODELIDERR	AEX3
38	INVMPSZ	AEYB
43	JIDERR	AEYG
44	QIDERR	AEYH
53	SYSIDERR	AEYQ
54	ISCINVREQ	AEYR
56	ENVDEFERR	AEYT

EIB fields

No.	Condition	Abend
61	NOTALLOC	AEYY
70	NOTAUTH	AEY7
42	NOSTG	ASCP
81	TERMERR	ATNI
45	NOJBUFSP	A17G

EIBRESP2

contains more detailed information that may help explain why the condition indicated by EIBRESP has occurred. The relevant values are documented with each command as applicable. For requests to remote files, EIBRESP2 contains zeros.

```
COBOL: PIC S9(8) BINARY.  
C:      long int eibresp2;
```

EIBRLDBK

indicates that the remote transaction has sent SYNCPOINT ROLLBACK in response to a SYNCPOINT request (X'FF').

```
COBOL: PIC X(1).  
C:      char eibrldb;
```

EIBRSRCE

contains the symbolic identifier of the resource being accessed by the last CICS command to be issued by the task:

Type of command	Resource	No of chars
BMS	Map name	7
File control	File name	8
Interval control	Transaction name	4
Journal control	Journal number	H
Program control	Program name	8
Temporary storage control	TS queue name	8
Terminal control	Terminal identifier or APPC conversation identifier	4
Transient data control	TD queue name	4

Identifiers less than eight characters long are padded on the right with blanks.

```
COBOL: PIC X(8).  
C:      char eibrsrce[8];
```

EIBSIG

indicates that the conversation partner has issued an ISSUE SIGNAL command (X'FF').

```
COBOL: PIC X(1).  
C:      char eibsig;
```

EIBSYNC

indicates that the application program must either issue a SYNCPOINT command or terminate (X'FF'). Before either is done, the application program must ensure that any other facilities it owns are put into the send state or are freed.

```
COBOL: PIC X(1).  
C:      char eibsync;
```

EIBSYNC is set when an EXEC CICS RECEIVE command detects that the partner has issued a SYNCPOINT request.

EIBSYNRB

indicates that the application program must issue a SYNCPOINT ROLLBACK command (X'FF'). This field is set only in application programs holding a conversation across an APPC link.

```
COBOL: PIC X(1).
C:      char eibsynrb;
```

EIBSYNRB is set when an EXEC CICS RECEIVE command detects that the partner has issued a SYNCPOINT ROLLBACK request.

EIBTASKN

contains the task number assigned to the task by CICS. This number appears in trace entries generated while the task is in control. The format of the field is packed decimal.

```
COBOL: PIC S9(7) PACKED-DECIMAL.
C:      char eibtaskn[4];
```

EIBTIME

contains the time at which the task is started; this field is updated by the ASKTIME command. The time is in packed decimal form (0HHMMSS+).

```
COBOL: PIC S9(7) PACKED-DECIMAL.
C:      char eibtime[4];
```

EIBTRMID

contains the symbolic terminal identifier of the principal facility (terminal or logical unit) associated with the task.

```
COBOL: PIC X(4).
C:      char eibtrmid[4];
```

EIBTRNID

contains the symbolic transaction identifier of the task.

```
COBOL: PIC X(4).
C:      char eibtrnid[4];
```

EIB fields

Appendix B. BMS-related constants

This appendix lists the BMS-related standard field attributes and printer control characters, and includes a bit map of attributes to help you generate combinations other than those listed. It also lists the attention identifier (AID) constants, which are applicable to BMS and terminal control input operations.

Field attribute and printer control characters

The standard list, DFHBMSCA, simplifies the provision of field attributes and printer control characters. Table 13 lists the symbolic names for the various combinations of attributes and control characters. If you need combinations other than those listed, you must generate them separately. Table 14 on page 547 shows a bit map of attributes to help you do this. Attributes and orders can be set only by their constant names or by their associated graphic values (for example, "A" for unprotected and modified data tag set).

The value of an attribute constant can be determined by referring to the *3274 Control Unit Reference Summary*.

You can get the standard attribute and printer character control list by including the DFHBMSCA copybook or ILE C header file in your application program as appropriate.

- For COBOL users, it consists of a set of PICTURE statements that must be copied into the working-storage section as follows:

```
COPY DFHBMSCA.CBL
```
- For ILE C users, it consists of a series of defined constants that are included in the application program as follows:

```
#include <dfhbmsca.h>
```

Note: IBM 5250 screen attributes are converted from 3270 attributes. For an explanation, see "IBM 5250 Information Display System" on page 141.

You must use the symbolic name DFHDFT in the application structure to override a map attribute with the default. On the other hand, to specify default values in a set attribute (SA) sequence in text build, you should use the symbolic names DFHDFCOL or DFHDFHI.

Table 13. Standard attribute and printer control character list, DFHBMSCA

Constant	Meaning
DFHBMPPEM	Printer end-of-message
DFHBMPNL	Printer new-line
DFHBMASK	Autoskip
DFHBMUNP	Unprotected
DFHBMUNN	Unprotected and numeric
DFHBMPRO	Protected
DFHBMBRY	Bright
DFHBMDAR	Dark
DFHBMFSE	MDT set
DFHBMPRF	Protected and MDT set
DFHBMAFSE	Autoskip and MDT set

BMS-related constants

Table 13. Standard attribute and printer control character list, DFHBMSCA (continued)

Constant	Meaning
DFHBMASB	Autoskip and bright
DFHBMPSO	shift-out value X'0E'
DFHBMPSI	shift-in value X'0F'
DFHBMEOF	Field erased
DFHBMFLG	Flags (COBOL only)
DFHSA ¹	Set attribute (SA) order
DFHERROR	Error code
DFHCOLOR ¹	Color
DFHHLT ¹	Highlight
DFH3270 ¹	Base 3270 field attribute
DFHVAL	Validation
DFHOUTLN	Field outlining attribute code
DFHALL ¹	Reset all to defaults
DFHDFT	Default
DFHDFCOL ¹	Default color
DFHBLUE	Blue
DFHRED	Red
DFHPINK	Pink
DFHGREEN	Green
DFHTURQ	Turquoise
DFHYELLO	Yellow
DFHNEUTR	Neutral
DFHDFHI ¹	Normal
DFHBLINK	Blink
DFHREVRS	Reverse video
DFHUNDLN	Underscore
DFHMFIL ²	Mandatory fill
DFHMENT ²	Mandatory enter
DFHMFEE	Mandatory fill and mandatory enter
DFHUNNOD	Unprotected, nondisplay, nonprint, MDT
DFHUNIMD	Unprotected, intensify, MDT
DFHUNNUM	Unprotected, numeric, MDT
DFHUNINT	Unprotected, numeric, intensify, MDT
DFHUNNON	Unprotected, numeric, nondisplay, nonprint, MDT
DFHPROTI	Protected, intensify
DFHPROTN	Protected, nondisplay, nonprint
DFHDFFR	Default outline
DFHUNDER	Underline
DFHRIGHT	Right vertical line
DFHOVER	Overline
DFHLEFT	Left vertical line
DFHBOX	Underline and right vertical and overline and left vertical
DFHSOSI	SOSI=yes
Note:	
¹	For text processing only. Use for constructing embedded set attribute orders in user text.
²	Cannot be used in set attribute orders.

BMS-related constants

Table 14. Bit map for attributes (continued)

prot	a/n	hi	spd ¹	ndp	mdt	EBCDIC	ASCII	char
U = Unprotected			N = Numeric			H = High		
P = Protected			S = Autoskip			Y = Yes		
Note:								
¹ spd is not supported by CICS/400.								

Attention identifier constants, DFHAID

The standard attention identifier list, DFHAID, simplifies testing the contents of the EIBAID field after a BMS or terminal control input operation associated with a display device. Table 15 shows the symbolic name for each attention identifier (AID) and the corresponding 3270 function.

You can get the standard attention identifier list by including the DFHAID COBOL copybook or ILE C header file in your application program as appropriate.

- For COBOL users, it consists of a set of PICTURE statements that must be copied into the working-storage section.
- For ILE C users, it consists of a series of defined constants that are included in the application program as follows:

```
#include <dfhaid.h>
```

Note: IBM 5250 screen attributes are converted from 3270 attributes. For an explanation, see “IBM 5250 Information Display System” on page 141.

Table 15. Standard attention identifier constants list, DFHAID

Constant	Meaning
DFHENTER	ENTER key
DFHCLEAR	CLEAR key
DFHPA1–DFHPA3	PA1–PA3 keys
DFHPPF1–DFHPPF24	PF1–PF24 keys

Appendix C. Terminal control

This appendix gives general information that applies to all terminals and logical units. For more detail, see the command descriptions.

Commands and options for terminals and logical units

Fullword lengths

For all terminal control commands, fullword length options can be used instead of halfword length options. In particular, where the following options are used in an EXEC CICS CONVERSE, RECEIVE, or SEND command, the corresponding alternative can be specified instead.

Option	Alternative
LENGTH	FLENGTH
TOLENGTH	TOFLENGTH
FROMLENGTH	FROMFLENGTH
MAXLENGTH	MAXFLENGTH

Application programs must be consistent in their use of fullword and halfword options on terminal control commands. The maximum value that can be specified as the argument on any length option is 32767; but see "LENGTH options" on page 312 for some general advice.

Read from terminal or logical unit (EXEC CICS RECEIVE)

The EXEC CICS RECEIVE command is used to read data from a terminal or logical unit. The INTO option is used to specify the area into which the data is to be placed. Alternatively, a pointer reference can be specified in the SET option. CICS acquires an area large enough to hold the data and sets the pointer reference to the address of that data.

The contents of this area are available to the task until the next terminal I/O command. However, the area does not belong to the task and is released by CICS while processing the next request. Therefore, this area cannot be passed back to CICS for further processing.

The application can use the MAXLENGTH option to specify the maximum length of data that the program accepts. If the MAXLENGTH option is omitted on an EXEC CICS RECEIVE command for which the INTO option is specified, the maximum length of data that the program accepts can be specified in the LENGTH option. If the MAXLENGTH option is omitted on an EXEC CICS RECEIVE command for which the SET option is specified, CICS acquires enough storage to hold all the available data.

If the data exceeds the specified maximum length and the NOTRUNCATE option is specified, the remaining data is made available to satisfy subsequent EXEC CICS RECEIVE commands. If NOTRUNCATE is not specified, the data is truncated and the LENGERR condition occurs. In this event, if the LENGTH option is specified, the named data area is set to the actual data length (before truncation occurs) when data has been received.

Terminal control

The first EXEC CICS RECEIVE command in a task started by a terminal does not issue a terminal control read but simply copies the input buffer, even if the data length is zero. A second EXEC CICS RECEIVE command must be issued to cause a terminal control read.

Write to terminal or logical unit (EXEC CICS SEND)

The EXEC CICS SEND command is used to write data to a terminal or logical unit. The FROM and LENGTH options specify respectively the data area from which the data is to be taken and the length (in bytes) of the data. For a transaction started by automatic transaction initiation (ATI), an EXEC CICS SEND command must always precede the first EXEC CICS RECEIVE in a transaction.

WAIT option of the EXEC CICS SEND command

Unless the WAIT option is specified also, the transmission of the data associated with the EXEC CICS SEND command is deferred until a later event, such as a syncpoint, occurs. This deferred transmission reduces the flows of data by allowing data-flow controls to be transmitted with the data.

A wait is always carried out for an EXEC CICS RECEIVE command. A wait may cause execution of a task to be suspended. Execution of the task is resumed when the operation is completed.

Even if the WAIT option is not specified in a SEND command, CICS ensures that the operation is completed before executing a subsequent EXEC CICS RECEIVE or SEND command.

Converse with terminal or logical unit (EXEC CICS CONVERSE)

For most terminals or logical unit types, a conversational mode of communication can be used. The EXEC CICS CONVERSE command is used for this purpose. In general, the EXEC CICS CONVERSE command can be considered as a combination of a SEND command with the WAIT option followed immediately by an EXEC CICS RECEIVE command. However, not all options of the EXEC CICS SEND and RECEIVE commands are valid for the EXEC CICS CONVERSE command. The TOLength option of the CONVERSE command is equivalent to the LENGTH option of the EXEC CICS RECEIVE command, and the FROMLength option is equivalent to the LENGTH option of the EXEC CICS SEND command.

Display device operations

In addition to the standard terminal control commands for sending and receiving data, several commands and lists are provided for use with display devices.

The commands are:

- Erase all unprotected fields (EXEC CICS ISSUE ERASEAUP). See below.
- Receive input without data (EXEC CICS RECEIVE with no options). See below.
 - Handle attention identifiers (EXEC CICS HANDLE AID). See Chapter 14, "Terminal control," on page 169.

The lists are:

- Standard attention identifier list (DFHAID). See "Attention identifier constants, DFHAID" on page 548.
- Standard attribute and printer control character list (DFHBMSCA). See "Field attribute and printer control characters" on page 545.

For devices with switchable screen sizes, the size of the screen that can be used, and the size to be used for a given transaction, are defined in CICS tables. These values can be obtained by means of the ASSIGN command, described on page 327.

The ERASE option should always be included in the first EXEC CICS SEND command, to clear the screen. This first EXEC CICS SEND command also formats the screen according to the transmitted data, and selects the screen size to be used, as specified in the PCT and TCT. If ERASE is omitted, the screen size is the same as its previous setting, which may be incorrect.

Use of the CLEAR key outside a transaction sets the screen to its default size.

Erase all unprotected fields (EXEC CICS ISSUE ERASEAUP)

The EXEC CICS ISSUE ERASEAUP command is used to erase all unprotected fields of a 3270 buffer, by the following actions:

1. Clearing all unprotected fields to nulls (X'00')
2. Resetting the modified data tags (MDTs) in each unprotected field to zero
3. Positioning the cursor to the first unprotected field
4. Restoring the keyboard

Input operation without data (EXEC CICS RECEIVE)

The EXEC CICS RECEIVE command with no options causes input to take place and the EXEC interface block (EIB) to be updated. However, data received by CICS is not passed on to the application program and is lost. A wait is implied. Two of the fields in the EIB that are updated are described below:

Cursor position (EIBCPOSN)

For every terminal control (or BMS) input operation associated with a display device, the screen cursor address (position) is placed in the EIBCPOSN field of the EIB. The cursor address is in the form of a halfword binary value and remains unaltered until updated by a new input operation.

EIBCPOSN is also updated at task initiation for non-ATI tasks.

Attention identifier (EIBAID)

For every terminal control (or BMS) input operation associated with a display device, an attention identifier (AID) is placed in the EIBAID field of the EIB. The AID indicates the method that the terminal operator has used to initiate the transfer of information from the device to CICS; for example, the ENTER key, a program function key, and so on. The field contents remain unaltered until updated by a new input operation.

EIBAID can be tested after each terminal control (or BMS) input operation to determine further processing, and a standard attention identifier list (DFHAID) is provided for this purpose. Alternatively, for COBOL programs only, the HANDLE AID command can be used to pass control to specified labels when the AIDs are received.

EIBAID is also updated at task initiation for non-ATI tasks.

Appendix D. BMS macro summary

This appendix contains the syntax of the BMS definition macros as supported by CICS/400. The purpose and format of each macro and its operands are described.

For more information about BMS, see Chapter 13, “CICS/400 basic mapping support (BMS),” on page 141.

Defining map sets, maps, and fields

You must ensure that the names of maps and the names of fields within a map set (or within multiple map sets that are copied into one application program) are unique. Maps and map sets must not have the same name.

Before CICS can load a physical map, you must define the maps by using the ADDCICSPT CL command with the CICSMAP parameter set to *YES. See the *CICS for iSeries Administration and Operations Guide* for details of this command.

You use the Create CICS MAP (CRTCICSMAP) CL command to generate the physical map and symbolic description map (see “CRTCICSMAP” on page 301 for details).

Map set definition macro (DFHMSD)

The DFHMSD macro defines a map set. A map set contains one or more maps.

Map definition macro (DFHMDI)

The DFHMDI macro defines a map within the map set defined by the previous DFHMSD macro. A map contains zero or more fields.

Field definition macro (DFHMDF)

The DFHMDF macro defines a field within a map defined by the previous DFHMDI macro.

Ending a map set definition

A map set definition ends with a macro of the form:

“mapset” is optional, but if used it must be the same as that on the DFHMSD

```
DFHMSD—TYPE=FINAL
```

macro that began the map set definition.

Defining field groups

Often an output data display field has to contain several subfields, all sharing the same display attributes. Each of these subfields might have to be modified separately. At output, subfields that have not been modified by the program can adopt default data values from the output map. For example, a display can include a date field with a “day” subfield, “month” subfield, and “year” subfield. The contents of the year subfield remain constant over a relatively long period; its

BMS macros

value can safely be taken from a map. However, the day value and month value must be updated more frequently. Similarly, on input, the terminal operator can enter data in each subfield separately.

You use the GRPNAME operand to define a group of subfields that combine to produce a field. The start of the group is indicated by a DFHMDF macro with the GRPNAME operand. This operand defines the first subfield, and specifies the attributes and name of the group. It is followed by other DFHMDF macros, one for each of the other subfields. Each of these must specify the group name, but cannot specify attribute values. The definition of the group is terminated by a DFHMDF macro that specifies a different group name, by one that specifies no group name, or by a DFHMDI or DFHMSD macro.

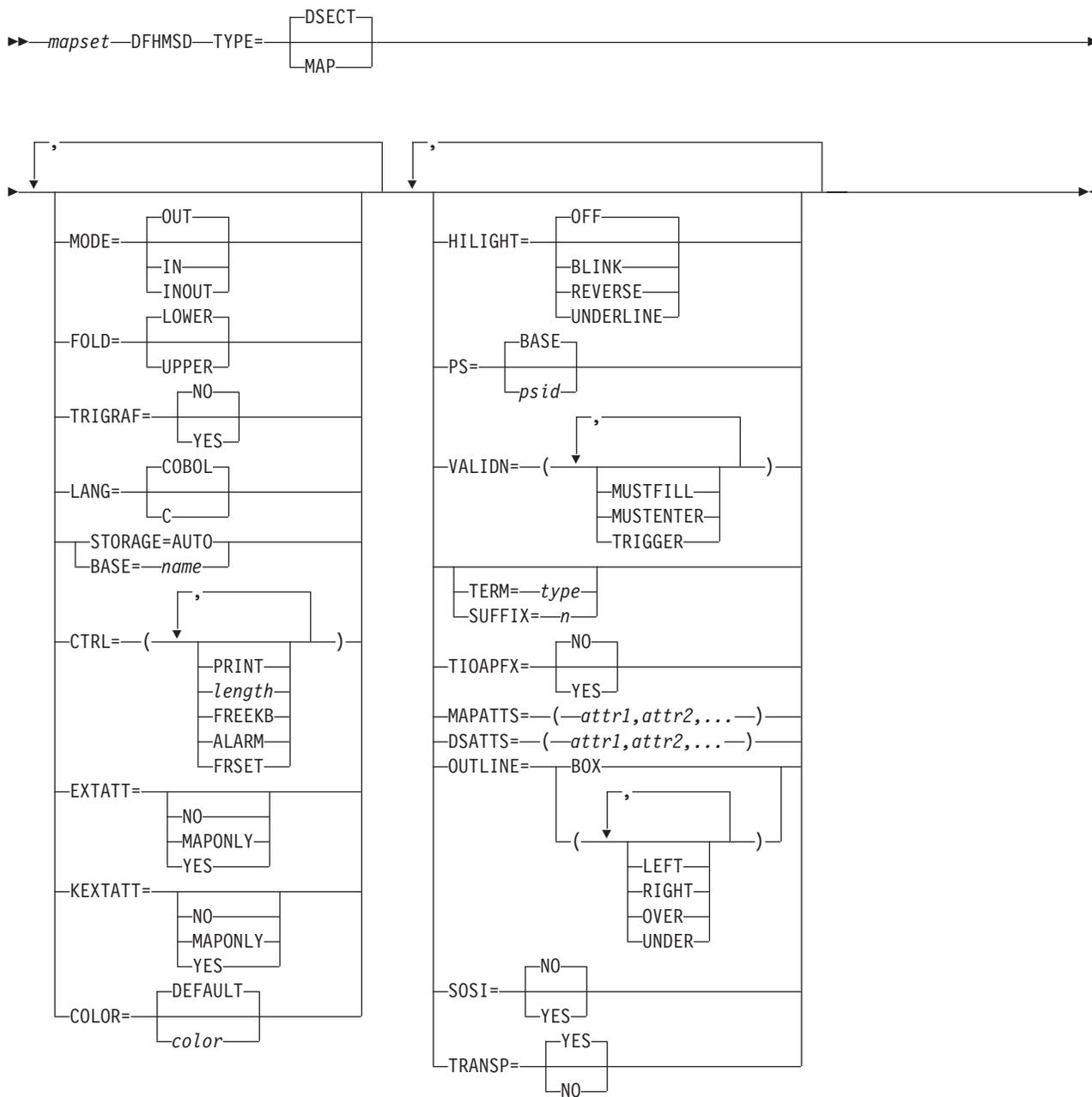
Briefly, a group of fields in a map would appear as follows in the map definition:

```
MAPSET DFHMSD....
      .
      .
MAP    DFHMDI....
      .
      .
DD     DFHMDF GRPNAME=DATE, POS=40,
          LENGTH=2, ATTRB=...
MM     DFHMDF GRPNAME=DATE, POS=46,
          LENGTH=2
YY     DFHMDF GRPNAME=DATE, POS=52,
          LENGTH=2
FIELD  DFHMDF LENGTH=5, COLOR=GREEN, ...
      DFHMSD TYPE=FINAL
```

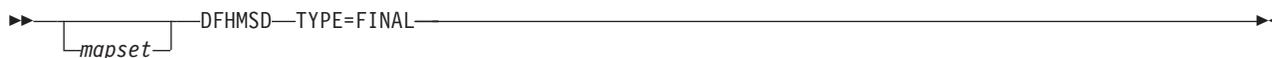
The POS operand specifies the position of the attribute byte of the field even though subfields of a group (other than the first) do not have attributes. If the subfields are positioned contiguously with no intervening blanks, the POS of the second and succeeding subfields must specify the last character of the previous subfield in each case.

DFHMSD

BMS—map set definition



or



A DFHMSD macro **defines a map set**; it begins:
 mapset DFHMSD TYPE=MAP (or TYPE=DSECT)

and ends:

```
[mapset] DFHMSD TYPE=FINAL
```

“mapset” is the name (1 through 7 characters) of the map set. For TYPE=FINAL, “mapset” is optional; but if used it must be the same as that on the DFHMSD macro that began the map set definition.

A DFHMSD macro contains one or more map definition macros, each of which contains zero or more field definition macros.

Options

BASE=name

specifies that the same storage base is used for the symbolic description maps from more than one map set. The same name is specified for each map set that is to share the same storage base. Because all map sets with the same base describe the same storage, data related to a previously used map set may be overwritten when a new map set is used. Furthermore, different maps within the same map set also overlay one another.

For example, assume that the following macros are used to generate symbolic description maps for two map sets:

```
MAPSET1 DFHMSD TYPE=DSECT,  
        LANG=COBOL,  
        BASE=DATAREA1,MODE=IN
```

```
MAPSET2 DFHMSD TYPE=DSECT,  
        LANG=COBOL,  
        BASE=DATAREA1,MODE=OUT
```

The symbolic description maps of this example might be referred to in a COBOL application program as follows:

```
WORKING-STORAGE SECTION.  
01 DATAREA1 PIC X(1920).  
01 name COPY MAPSET1.  
01 name COPY MAPSET2.
```

```
      .  
      .
```

MAPSET1 and MAPSET2 both redefine DATAREA1; only one 02 statement is needed to establish addressability. However, the program can only use the fields in one of the symbolic description maps at a time.

If BASE=DATAREA1 is deleted from this example, an additional 02 statement is needed to establish addressability for MAPSET2; the 01 DATAREA1 statement is not needed. The program could then refer to fields concurrently in both symbolic description maps.

COLOR

specifies the color to be used for all maps in the map set. This color can be overridden for individual maps by the COLOR operand of the DFHMDI macro, which is in turn overridden for individual fields by the COLOR operand of the DFHMDF macro. If this operand is omitted, the default color for the output device is used.

If COLOR is specified when EXTATT=NO, a warning is issued and the operand ignored.

The COLOR operand is ignored if the terminal does not support color.

color specifies the basic color to be used for all maps in the map set. The valid colors are BLUE, RED, PINK, GREEN, TURQUOISE, YELLOW, and NEUTRAL.

DEFAULT specifies that the default color for the output device is to be used as the basic color for all maps in the map set.

CTRL

defines characteristics of IBM 5250 and 3270 terminals. Use of *any* of the control options in the SEND MAP command overrides *all* control actions specified in the DFHMDI macro, which in turn override *all* control actions specified in the DFHMSD macro.

PRINT must be specified if the printer is to be started; if omitted, the data is sent to the printer buffer but is not printed.

length indicates the line length on the printer; length can be specified as L40, L64, L80, or HONEOM. L40, L64, and L80 force a new line after 40, 64, or 80 characters, respectively. HONEOM causes the default printer line length to be used. If the length value is omitted, BMS sets the line length from the TCT.

FREEKB causes the keyboard to be unlocked after the map is written. If FREEKB is not specified, the keyboard remains locked; data entry from the keyboard is inhibited until this status is changed.

ALARM activates the audible alarm on a display device.

FRSET specifies that the modified data tags (MDTs) of all fields currently in the 3270 buffer are to be reset to a not-modified condition (that is, field reset) before map data is written to the buffer. This allows the DFHMDF macro with the ATTRB operand to control the final status of any fields written or rewritten in response to a BMS command.

DSATTS

specifies the attribute types to be included in the symbolic description map. These types can be one or more of the following: COLOR, HILIGHT, OUTLINE, PS, SOSI, TRANSP, and VALIDN. Any type included in DSATTS must also be included in MAPATTS.

Specifying an attribute has no effect if the device receiving the map does not support the attribute.

EXTATT

This operand is supported for compatibility with early releases of mainframe CICS. Each of the extended attributes can be defined individually. For new maps, use the operands DSATTS and MAPATTS instead.

NO is equivalent to neither of the operands DSATTS and MAPATTS being specified.

YES is equivalent to:
MAPATTS=(COLOR,HILIGHT,PS,VALIDN)
DSATTS=(COLOR,HILIGHT,PS,VALIDN)

MAPONLY is equivalent to:
MAPATTS=(COLOR,HILIGHT,PS,VALIDN)

FOLD

specifies the case of characters in variable names in ILE C programs. This option is available only for programs written in ILE C.

LOWER Lowercase characters are generated.

UPPER Uppercase characters are generated.

HIGHLIGHT

specifies the default highlighting attribute for all fields in all maps in a map set. This can be overridden for individual maps by the HIGHLIGHT operand of the DFHMDI macro, which is in turn overridden for individual fields by the HIGHLIGHT operand of the DFHMDF macro.

OFF is the default and indicates that no highlighting is used.

BLINK

specifies that the field must blink. For 5250 devices, this attribute works only if the field color is red.

REVERSE

specifies that the character or field is displayed in reverse video.

UNDERLINE

specifies that the field is underlined.

If HIGHLIGHT is specified when EXTATT=NO, a warning is issued and the operand ignored.

The HIGHLIGHT operand is ignored if the terminal does not support highlighting.

KEXTATT

This operand is supported for compatibility with early releases of mainframe CICS. It enables you to define the SOSI and OUTLINE attributes. For new maps, use the operands DSATTS and MAPATTS instead.

NO is equivalent to neither of the operands DSATTS and MAPATTS being specified.

YES is equivalent to:
MAPATTS=(SOSI,OUTLINE)
DSATTS=(SOSI,OUTLINE)

MAPONLY is equivalent to:
MAPATTS=(SOSI,OUTLINE)

LANG

specifies the source language of the application programs into which the symbolic description maps in the map set are copied. This operand need only be coded for DFHMSD TYPE=DSECT. If this operand is omitted, COBOL is assumed. For application portability, however, you should always code the LANG operand. If a map set is to be used by more than one program, and the programs are not all written in the same source language, a separate version of the map set must be defined for each programming language.

MAPATTS

specifies the attribute types to be included in the physical map. These types can be one or more of the following: COLOR, HIGHLIGHT, PS, OUTLINE, TRANSP, SOSI, and VALIDN. This list must include all the attribute types to be specified for individual fields in the map (DFHMDF macro).

Where possible, the attribute types are deduced from operands already specified in the DFHMSD macro. For example, if COLOR=BLUE is specified, MAPATTS is assumed to include COLOR.

When the MAPATTS operand is specified, the EXTATT and KEXTATT operands are ignored.

MODE

specifies whether the map is to be used for input, output, or both.

OUTLINE

allows lines to be included above, below, to the left, or to the right of a field. You can use these lines in any combination to construct boxes around fields or groups of fields.

The OUTLINE operand is ignored if the terminal or printer does not support outlining. For example, 5250 terminal types do not support outlining.

PS specifies that programmed symbols are to be used. This can be overridden for individual maps by the PS operand of the DFHMDI macro, which is in turn overridden for individual fields by the PS operand of the DFHMDF macro.

BASE specifies that the base symbol set is to be used.

psid specifies a single character, or an EBCDIC hexadecimal code of the form X'nn', that identifies the set of programmed symbols to be used.

Note: Only a psid value of 8 is supported.

If PS is specified when EXTATT=NO, a warning is issued and the operand ignored.

The PS operand is ignored if the terminal does not support programmed symbols.

SOSI

specifies that the field may contain a mixture of EBCDIC and DBCS data. The DBCS subfields within an EBCDIC field are delimited by SO (shift-out) and SI (shift-in) characters. SO and SI both occupy a single screen position (normally displayed as a blank). They can be included in any non-DBCS field on output if they are correctly paired. The terminal user can transmit them inbound if they are already present in the field, but can add them to an EBCDIC field only if the field has the SOSI attribute.

STORAGE=AUTO

The meaning of this operand depends on the language in which application programs are written, as follows:

- For **COBOL** programs, STORAGE=AUTO specifies that the symbolic description maps in the map set are to occupy separate (that is, not redefined) areas of storage. This operand is used when the symbolic description maps are copied into the working-storage section and the storage for the separate maps in the map set is to be used concurrently.
- For **ILE C** programs, STORAGE=AUTO specifies that the symbolic description maps are to be defined as having the automatic storage class. If STORAGE=AUTO is not specified, they are declared as pointers. If STORAGE=AUTO is specified and TIOAPFX is not, TIOAPFX=YES is assumed.

You cannot specify both BASE=name and STORAGE=AUTO for the same map set.

SUFFIX

specifies a 1-character, user-defined, device-dependent suffix for this map set, as an alternative to the suffix generated by the TERM operand. The suffix specified by this operand must match the value of the ALTSUFFIX parameter

specified on the definition for the terminal. Use a numeric value to avoid conflict with unsuffixed map set names.

TERM

specifies the type of terminal or logical unit (LU) associated with the mapset. If no terminal type or LU is specified, 3270 is assumed. The terminal types and LUs you can specify, together with their generated suffixes, are shown in Table 16.

Table 16. BMS terminal types

TYPE	Suffix
3270-2	M
3270	blank
5250	Z
ALL (all the above)	blank

3270 is the default. Specifying 3270 has the same effect as specifying ALL and should be used when there is no need to distinguish between models.

For 3270 model 5's, specify the terminal type as 3270 and set the alternate screen size to 27x132 in the TCT entry.

If ALL is specified, ensure that device-dependent characters are not included in the map set and that format characteristics such as page size are suitable for all input/output operations (and all terminals) in which the map set is applied.

BMS support for device-dependent map sets can be bypassed by specifying the NODDS parameter in the ADDCICSSIT command. For more information, see the *CICS for iSeries Administration and Operations Guide*.

TIOAPFX

is a 12-byte filler that command-level CICS for MVS/ESA uses for internal processing. CICS/400 maintains the 12-byte filler for migration purposes only but does not use it.

TRANSP

determines whether the background of an alphanumeric field is transparent or opaque, that is, whether an underlying (graphic) presentation space is visible between the characters.

TRIGRAF (ILE C only)

specifies that, for those characters in the ILE C character set that are not available on the keyboard, a sequence of three characters, known as a trigraph, may be used instead.

NO The symbolic description maps do not contain trigraph sequences.

YES The symbolic description maps do contain trigraph sequences.

Characters and the corresponding trigraph sequences are:

Character	Trigraph
{	??<
}	??>
[??(
]	??)

TYPE

when either DSECT or MAP is specified, generates respectively a symbolic description map or a physical map. It is not necessary to code either of these options. The default action is to generate both types of map. Symbolic description maps must be copied into the source program before it is translated and compiled. Physical maps must be cataloged in the CICS program library before an application program can use them.

VALIDN

specifies that validation is to be used on a 5250 terminal. This can be overridden for individual maps by the VALIDN operand of the DFHMDDI macro, which is in turn overridden for individual fields by the VALIDN operand of the DFHMDF macro.

MUSTFILL specifies that the field must be filled completely with data. An attempt to move the cursor from the field before it has been filled, or to transmit data from an incomplete field, raises the "inhibit input" condition.

MUSTENTER specifies that data must be entered into the field, but need not fill it. An attempt to move the cursor from an empty field raises the "inhibit input" condition.

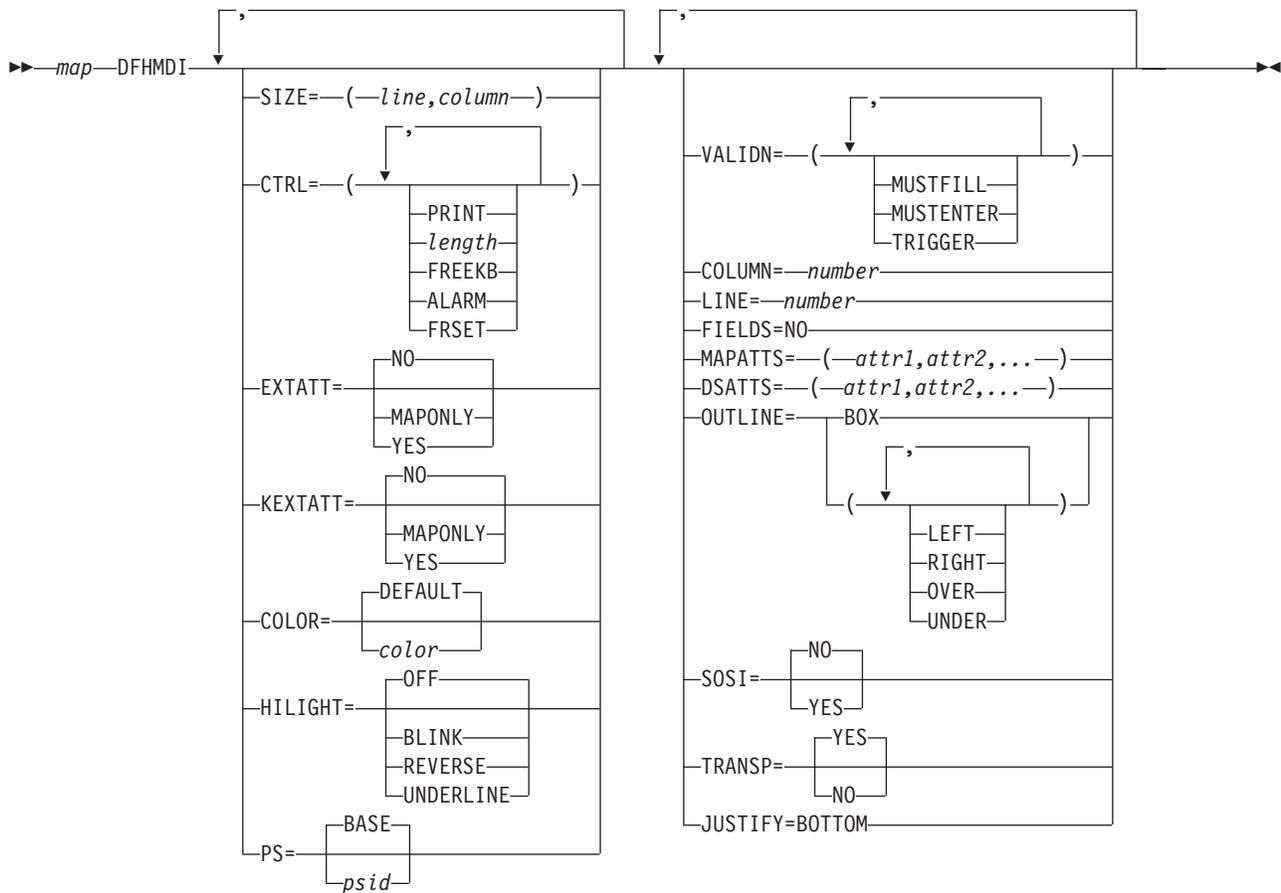
TRIGGER specifies, on other CICS platforms, that this field is a trigger field. In CICS/400 this option has no effect and is supported only for compatibility.

The VALIDN operand is ignored if the terminal does not support validation.

If VALIDN is specified when EXTATT=NO, a warning is issued and the operand is ignored.

DFHMDI

BMS—map definition



The DFHMDI macro **defines a map** within the map set defined by the previous DFHMSD macro. A map set contains one or more maps.

“map” is the name (1 through 7 characters) of the map.

Note for COBOL users: If the maps are for use in a COBOL program, and STORAGE=AUTO has not been specified in the DFHMSD macro, they must be specified in descending size sequence. (Size refers to the generated 01-level data areas and not to the size of the map on the screen.)

Options

COLOR

specifies the color to be used for the named map. This overrides the COLOR operand of the DFHMSD macro, and is in turn overridden for individual fields by the COLOR operand of the DFHMDF macro.

If COLOR is specified when EXTATT=NO, a warning is issued and the operand ignored.

The COLOR operand is ignored if the terminal does not support color.

color specifies the basic color to be used for this map. The valid colors are BLUE, RED, PINK, GREEN, TURQUOISE, YELLOW, and NEUTRAL.

DEFAULT specifies that the default color for the output device is to be used as the basic color for this map.

COLUMN

specifies the column in a line at which the map is to be placed; that is, it establishes the left map margin. The columns between the specified map margin and the page margin are not available for subsequent use on the page for any lines included in the map.

number

is the column from the left page margin where the left map margin is to be established.

CTRL

defines characteristics of IBM 5250 and 3270 terminals. Use of *any* of the control options in the SEND MAP command overrides *all* control actions specified in the DFHMDI macro, which in turn override *all* control actions specified in the DFHMSD macro.

PRINT

must be specified if the printer is to be started; if omitted, the data is sent to the printer buffer but is not printed.

length indicates the line length on the printer; length can be specified as L40, L64, L80, or HONEOM. L40, L64, and L80 force a new line after 40, 64, or 80 characters, respectively. HONEOM causes the default printer line length to be used. If the length value is omitted, BMS sets the line length from the TCT.

FREEKB

causes the keyboard to be unlocked after the map is written. If FREEKB is not specified, the keyboard remains locked; data entry from the keyboard is inhibited until this status is changed.

ALARM

activates the audible alarm on the terminal device.

FRSET

specifies that the modified data tags (MDTs) of all fields currently in the 3270 buffer are to be reset to a not-modified condition (that is, field reset) before map data is written to the buffer. This allows the DFHMDF macro with the ATTRB operand to control the final status of any fields written or rewritten in response to a BMS command.

DSATTS

specifies the attribute types to be included in the symbolic description map. These types can be one or more of the following: COLOR, HILIGHT, OUTLINE, PS, SOSI, TRANSP, and VALIDN. Any type included in DSATTS must also be included in MAPATTS.

Specifying an attribute has no effect if the device receiving the map does not support the attribute.

EXTATT

This operand is supported for compatibility with early releases of mainframe CICS. Each of the extended attributes can be defined individually. For new maps, use the operands DSATTS and MAPATTS instead.

NO is equivalent to neither of the operands DSATTS and MAPATTS being specified.

YES is equivalent to:

MAPATTS=(COLOR,HILIGHT,PS,VALIDN)

DSATTS=(COLOR,HILIGHT,PS,VALIDN)

MAPONLY

is equivalent to:

MAPATTS=(COLOR,HILIGHT,PS,VALIDN)

FIELDS=NO

specifies that the map contains no fields. If you specify FIELDS=NO, you create a null map that defines a “hole” in BMS’s view of the screen. BMS cannot change the contents of such a hole after it has created it by sending a null map.

HILIGHT

specifies the default highlighting attribute for all fields in the named map. This overrides the HILIGHT operand of the DFHMSD macro, and is in turn overridden for individual fields by the HILIGHT operand of the DFHMDF macro.

OFF is the default and indicates that no highlighting is used.

BLINK specifies that the field must blink. For 5250 devices, this attribute works only if the field color is red.

REVERSE specifies that the character or field is displayed in reverse video. For example, on a 3278, black characters are displayed on a green background.

UNDERLINE specifies that the field is underlined.

If HILIGHT is specified when EXTATT=NO, a warning is issued and the operand ignored.

The HILIGHT operand is ignored if the terminal does not support highlighting.

JUSTIFY=BOTTOM

specifies that the map is to be positioned at the bottom of the screen. This operand applies to SEND MAP and RECEIVE MAP commands, provided that the number of lines in the map is specified in the SIZE operand; otherwise, it is ignored.

JUSTIFY=BOTTOM is equivalent to specifying

LINE=(screendepth - mapdepth + 1)

on the map definition, but it allows the same map to be used for different screen sizes. If JUSTIFY=BOTTOM and the LINE operand are both specified, the value specified in LINE is ignored.

Note: If a field is initialized by an output map or contains data from any other source, data that is entered as input overwrites only the equivalent length of existing data; any surplus existing data remains in the field and could cause unexpected interpretation of the new data.

KEXTATT

This operand is supported for compatibility with early releases of mainframe

CICS. It allows you to specify the SOSI and OUTLINE attributes. For new maps, use the operands DSATTS and MAPATTS instead.

NO is equivalent to neither the DSATTS operand nor the MAPATTS operand being specified.

YES is equivalent to:
 MAPATTS=(SOSI,OUTLINE)
 DSATTS=(SOSI,OUTLINE)

MAPONLY
 is equivalent to:
 MAPATTS=(SOSI,OUTLINE)

LINE

specifies the starting line on a page in which data for a map is to be formatted.

number

is a value in the range 1 through 27, specifying a starting line number.

MAPATTS

specifies the attribute types to be included in the physical map. These types can be one or more of the following: COLOR, HIGHLIGHT, OUTLINE, PS, SOSI, TRANSP, and VALIDN. This list must include all the attribute types to be specified for individual fields in the map (DFHMDF macro).

Where possible, the attribute types are deduced from operands already specified in the DFHMDS and DFHMDSI macros. For example, if COLOR=BLUE is specified, MAPATTS is assumed to include COLOR.

When the MAPATTS operand is specified, the EXTATT and KEXTATT operands are ignored.

OUTLINE

allows lines to be included above, below, to the left, or to the right of a field. You can use these lines in any combination to construct boxes around fields or groups of fields.

The OUTLINE operand is ignored if the terminal or printer does not support outlining. For example, 5250 terminal types do not support outlining.

PS specifies that programmed symbols are to be used. This overrides the PS operand of the DFHMDS macro, and is in turn overridden for individual fields by the PS operand of the DFHMDF macro.

BASE specifies that the base symbol set is to be used.

psid specifies a single character, or an EBCDIC hexadecimal code of the form X'nn', that identifies the set of programmed symbols to be used.

Note: Only a psid value of 8 is supported.

If PS is specified when EXTATT=NO, a warning is issued and the operand ignored.

The PS operand is ignored if the terminal does not support programmed symbols.

SIZE

specifies the size of a map.

line is a value in the range 1 through 240, specifying the depth of a map as a number of lines.

BMS macros

column

is a value in the range 1 through 240, specifying the width of a map as a number of columns.

This operand is required in the following cases:

- An associated DFHMDF macro with the POS operand is used.
- The map is to be used when referring to input data from other than a 3270 terminal in a RECEIVE MAP command.

The map dimensions specified in the SIZE operand of the DFHMDF macro defining a map may be smaller than the actual page size or screen size as defined for the terminal.

TRANSP

determines whether the background of an alphanumeric field is transparent or opaque, that is, whether an underlying (graphic) presentation space is visible between the characters.

VALIDN

specifies that validation is to be used on a 5250 terminal. This overrides the VALIDN operand of the DFHMDF macro, and is in turn overridden for individual fields by the VALIDN operand of the DFHMDF macro.

MUSTFILL specifies that the field must be filled completely with data. An attempt to move the cursor from the field before it has been filled, or to transmit data from an incomplete field, raises the “inhibit input” condition.

MUSTENTER specifies that data must be entered into the field, but need not fill it. An attempt to move the cursor from an empty field raises the “inhibit input” condition.

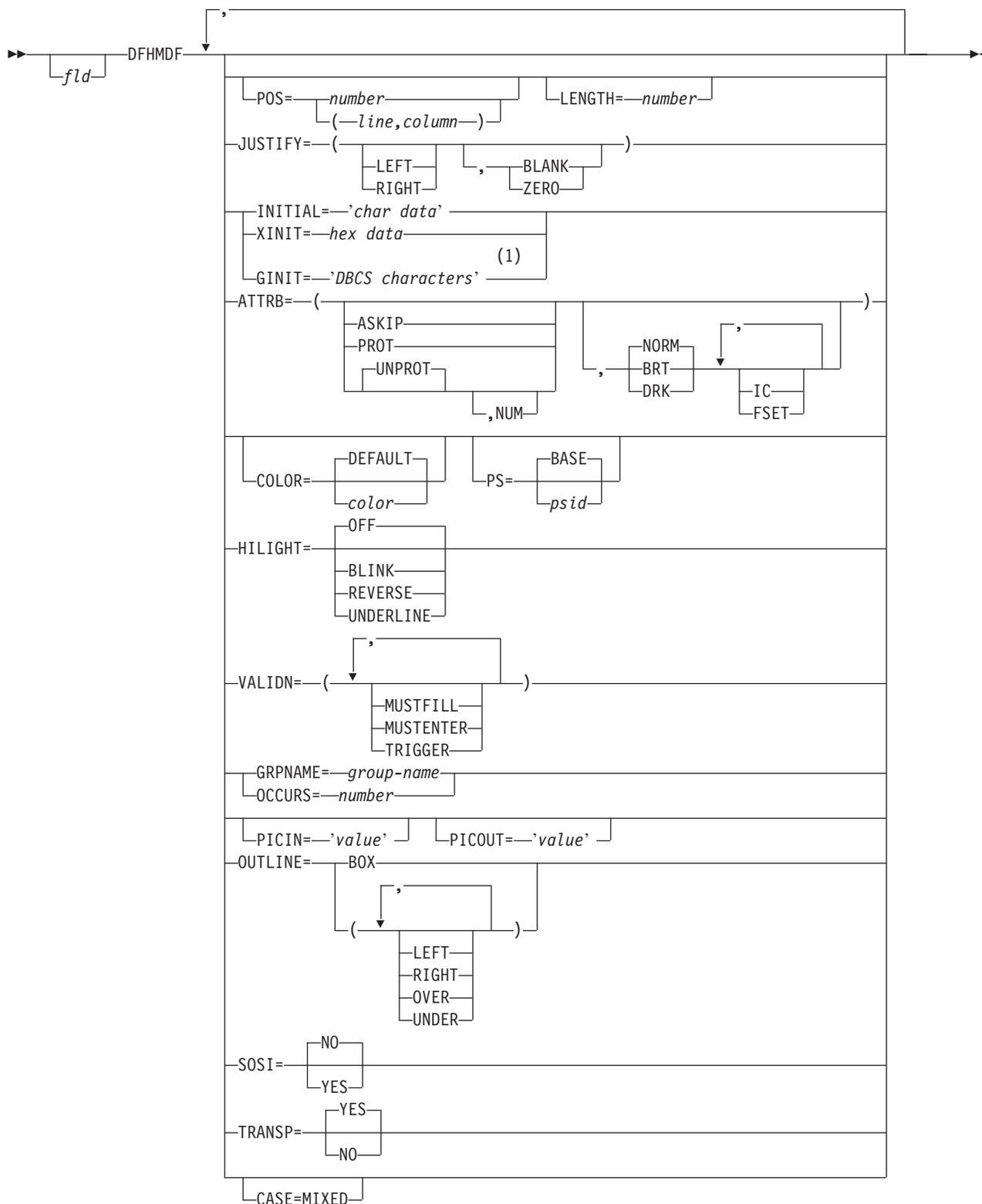
TRIGGER specifies, on other CICS platforms, that this field is a trigger field. In CICS/400 this option has no effect and is supported only for compatibility.

The VALIDN operand is ignored if the terminal does not support validation.

If VALIDN is specified when EXTATT=NO, a warning is issued and the operand is ignored.

DFHMDF

BMS—field definition



Notes:

- 1 DBCS character strings start with a Shift-out character X'0E' and end with a Shift-in character X'0F'. The DFHMDI macro **defines a field** within a map defined by the previous DFHMDI macro. A map contains zero or more fields.

“fld” is the name (1 through 29 characters) of the field.

If “fld” is omitted, application programs cannot access the field to change its attributes or alter its contents. For an output map, omitting the field name may be appropriate when the INITIAL operand is used to specify the contents of a field. If a field name is specified and the map that includes the field is used in a mapping operation, nonnull data supplied by the user overlays data supplied by initialization (unless default data only is being written).

The performance of mapping operations is optimized if DFHMDF macros are arranged in numeric order of the POS operand.

You cannot define more than 256 fields in a map for 5250 devices. You cannot define more than 1023 named fields for a COBOL input/output map.

Options

ATTRB

specifies how a field is displayed on a device. It can also cause the field to be returned to the application, or even protect the field from user input.

If ATTRB is specified within a group of fields, it must be specified in the first field entry. (A group of fields appears as one field to the 3270.) The ATTRB specification refers to all of the fields in a group as one field rather than as individual fields.

It specifies device-dependent characteristics and attributes, such as the capability of a field to receive data, or the intensity to be used when the field is output. It could, however, be used for making an input field nondisplay for secure entry of a password from a screen. For input map fields, DET is the only valid attribute; all others are ignored.

When defining a field to be displayed on a 3270, either of two sets of defaults may apply if not all attributes are specified. If no ATTRB values are specified, ASKIP and NORM are assumed. If any value is specified, UNPROT and NORM are assumed for that field unless overridden by a specified value.

ASKIP

specifies that data cannot be keyed into the field and causes the cursor (current location pointer) to skip over the field.

PROT specifies that data cannot be keyed into the field.

If data is to be copied from one device to another attached to the same 3270 control unit, the first position (address 0) in the buffer of the device to be copied from must not contain an attribute byte for a protected field. Therefore, when preparing maps for 3270s, ensure that the first map of any page does not contain a protected field starting at position 0.

UNPROT

specifies that data can be keyed into the field.

NUM ensures that the data entry keyboard is set to numeric shift for this field unless the operator presses the alpha shift key. It also prevents entry of nonnumeric data if the Keyboard Numeric Lock feature is installed.

BRT specifies that a high-intensity display of the field is required.

NORM

is the default and specifies that the field intensity is to be normal.

DRK specifies that the field is both nonprint and nondisplay.

IC specifies that the cursor is to be placed in the first position (after the attribute byte) of the field. The IC attribute for the last field for which it is specified in a map is the one that takes effect. If not specified for any fields in a map, the default location is zero. Specifying IC with ASKIP or PROT causes the cursor to be placed in an unkeyable field.

ATTRB=IC can be overridden by the CURSOR option of the SEND MAP command that causes the write operation.

FSET specifies that the modified data tag (MDT) for this field is to be set when the field is sent to a terminal.

Specification of FSET causes the 3270 to treat the field as though it has been modified. On a subsequent read from the terminal, this field is read, whether or not it has been modified. The MDT remains set until the field is rewritten without ATTRB=FSET or until an output mapping request causes the MDT to be reset.

CASE=MIXED

specifies that the field contains both uppercase and lowercase data that is to be converted to uppercase if FEATURE=KATAKANA has been included in the terminal definition. Do not specify this option if the field may contain valid Katakana characters.

COLOR

specifies the color to be used for the named field. This overrides the COLOR operands of the DFHMDF and DFHMDFI macros.

The COLOR operand is ignored if the terminal does not support color.

color specifies the color to be used for this field. The valid colors are BLUE, RED, PINK, GREEN, TURQUOISE, YELLOW, and NEUTRAL.

DEFAULT

specifies that the default color for the output device is to be used as the color for this field.

GINIT1	DFHMDF POS=(02,10),LENGTH=10,PS=8, GINIT='<D1D2D3D4D5>'	*
GINIT2	DFHMDF POS=(03,10),LENGTH=12,PS=8, GINIT='<D1D2D3D4D5D6D7D8>'	*
GINIT3	DFHMDF POS=(04,10),LENGTH=40,PS=8,GINIT='<D1D2D3D4D5D6D7D8>***** <D9D0D1D2D3D4D5D6D7D8D9D0>'	

GINIT

specifies constant or default data for an output field, in DBCS character strings. This parameter is for use with pure DBCS strings.

Data defined with this operand is checked as follows:

- The complete constant must be enclosed by a SO/SI pair.
- Each DBCS character must be either a DBCS blank (X'4040'), or each byte must have a value in the range X'41' through X'FE'.

- No SBCS meaning is taken from a DBCS character.
- The length value must be a multiple of 2 in the range 2 through 256, which allows from 1 to 128 DBCS characters. The SO and SI are not included in the length calculation.
- When necessary, data is truncated starting from the rightmost character position.
- When necessary, data is padded from the right with DBCS blank characters.

In the first example above, a 'DBCS only' field is defined containing five DBCS characters which occupy 10 positions in the device buffer. The length is coded as 10 to reflect this. The shift in and shift out characters are not included in the length value.

In the second example, a 'DBCS only' field is defined containing eight DBCS characters which would require sixteen positions in the device buffer. However, the LENGTH value is coded as 12, and this overrides the implicit length of the data supplied with the GINIT keyword. This results in truncation of the data and the rightmost two DBCS characters are not stored in the physical map.

In the third example, a 'DBCS only' field is defined containing twenty DBCS characters which require forty positions in the device buffer. Accordingly, the LENGTH value is coded as 40. However, the GINIT keyword and data are too long to fit on one line. To overcome this, you can split the data by placing a shift in character after the data characters coded on the first line, and a shift out character in the continuation column (column 16) of the next line.

These extra SI and SO characters, although required by the BMS translator, are considered to be *redundant* because they are removed from the data string before it is stored in the physical map.

This example also demonstrates the use of *extended continuation characters*. The continuation character (an asterisk) coded in column 72 is propagated back toward the data string. Scanning backward from column 72, the BMS translator takes the first character having a different value to the continuation character as the last data character on this line.

GRPNAME

specifies a name (1 through 7 characters) used to generate symbolic storage definitions and to combine specific fields under one group name. The same group name must be specified for each field that is to belong to the group.

If this operand is specified, the OCCURS operand cannot be specified.

The fields in a group must follow on; there can be gaps between them, but not other fields from outside the group. A field name must be specified for every field that belongs to the group, and the POS operand must also be specified to ensure that the fields follow each other. All the DFHMDF macros defining the fields of a group must be placed together, and in the correct order (ascending numeric order of the POS value).

For example, the first 20 columns of the first six lines of a map can be defined as a group of six fields, so long as the remaining columns on the first five lines are not defined as fields.

The ATTRB operand specified on the first field of the group applies to all of the fields within the group.

A display field cannot extend beyond the right-hand edge of a map. The length of the display field built by a group of subfields is thus limited to the width of the map.

For more information about field groups, see “Defining field groups” on page 553..

HIGHLIGHT

specifies the highlighting attribute for the named field. This overrides the HIGHLIGHT operands of the DFHMSD and DFHMDI macros.

OFF is the default and indicates that no highlighting is used.

BLINK specifies that the field must blink. For 5250 devices, this attribute works only if the field color is red.

REVERSE specifies that the character or field is displayed in reverse video. For example on a 3278, black characters are displayed on a green background.

UNDERLINE specifies that the field is underlined.

INIT1	DFHMDI	POS=(02,23),LENGTH=23,SOSI=YES, INITIAL='Single<.A.B.C.D.E>bytes'	*
INIT2	DFHMDI	POS=(03,23),LENGTH=23,SOSI=YES,INITIAL='Single<.A.B>**** <.C.D.E>bytes'	

The HIGHLIGHT operand is ignored if the terminal does not support highlighting.

INITIAL

specifies, in character form, constant or default data for an output field. Only one of INITIAL, XINIT, and GINIT may be specified.

The number of characters that you can specify in the INITIAL operand is restricted to the continuation limitation of the assembler to be used or to the value specified in the LENGTH operand (whichever is the smaller). With Assembler H, you can change the continuation column by using the Input Format Control (ICTL) instruction.

Data defined with this operand is checked as follows:

- The data string may contain one or more DBCS subfields, but on each line SO and SI characters must be matched.
- Each DBCS character must be either a DBCS blank (X'4040'), or each byte must have a value in the range X'41' through X'FE'.
- No SBCS meaning is taken from a DBCS character.
- The length value must be in the range 1 through 256. SO and SI characters are included in the length calculation.
- When the LENGTH value is less than the length of the INITIAL data, only SBCS data can be truncated.
- When necessary, data is padded from the right with SBCS blank characters.

In the first example above, a DBCS subfield is embedded in SBCS data characters. SOSI=YES is defined to indicate that this field is a mixed data field. Each SBCS character has a length of 1, each of the five DBCS characters has a length value of 2, and the SO and SI each have a length value of 1. Hence the total length is 23. The BMS translator retains the SO and SI characters

embedded in the data, and stores them in the physical map, because these characters must be transmitted to the device.

The second example is the same as the first, in that it has the same effect on the terminal display. However, here the *redundant SO/SI* and *extended continuation character* techniques are used to continue an INITIAL operand specification across lines.

If this definition were defined with LENGTH=20, this would be acceptable and the last three characters (tes) would be removed from the data string. However, if LENGTH=17 (or a lesser value) were defined, this would be an error because truncation into DBCS data is being requested. A length greater than 23 would result in SBCS blank characters being appended to the end of the data string.

JUSTIFY

specifies the field justifications for input operations.

LEFT specifies that data in the input field is left-justified.

RIGHT

specifies that data in the input field is right-justified.

BLANK

specifies that blanks are to be inserted in any unfilled positions in an input field.

ZERO specifies that zeros are to be inserted in any unfilled positions in an input field.

LEFT and RIGHT are mutually exclusive, as are BLANK and ZERO. If certain values are specified for JUSTIFY but others are not, assumptions are made as follows:

Specified	Assumed
LEFT	BLANK
RIGHT	ZERO
BLANK	LEFT
ZERO	RIGHT

If JUSTIFY is omitted, but the NUM attribute is specified, RIGHT and ZERO are assumed. If JUSTIFY is omitted, but attributes other than NUM are specified, LEFT and BLANK are assumed.

Note: If a field is initialized by an output map or contains data from any other source, data that is entered as input overwrites only the equivalent length of existing data; any surplus existing data remains in the field and could cause unexpected interpretation of the new data.

LENGTH

specifies the length (in the range 1 through 256 bytes) of the field. This length should be the maximum length required for application program data to be entered into the field; it should not include the 1-byte attribute indicator appended to the field by CICS for use in subsequent processing. The sum of the lengths of the fields within a group must not exceed 256 bytes. LENGTH can be omitted if PICIN or PICOUT is specified, but is required otherwise. You can specify a length of zero only if you omit the label (field name) from the DFHMDF macro. That is, the field is not part of the application data structure and the application program cannot modify the attributes of the field. You can use a field with zero length to delimit an input field on a map.

If the LENGTH specification in a DFHMDF macro causes the map-defined boundary on the same line to be exceeded, the field on the output screen is continued by wrapping.

OCCURS

specifies that the indicated number of entries for the field are to be generated in a map, and that the map definition is to be generated in such a way that the fields are addressable as entries in a matrix or an array. This permits several data fields to be addressed by the same name (subscripted) without generating a unique name for each field. OCCURS and GRPNAME are mutually exclusive; that is, OCCURS cannot be used when fields have been defined under a group name.

OUTLINE

allows lines to be included above, below, to the left, or to the right of a field. You can use these lines in any combination to construct boxes around fields or groups of fields.

The OUTLINE operand is ignored if the terminal or printer does not support outlining.

PICIN (COBOL only)

specifies a picture to be applied to an input field in an IN or INOUT map; this picture serves as an editing specification that is passed to the application program, thus permitting the user to exploit the editing capabilities of COBOL. BMS checks that the specified characters are valid picture specifications for the language of the map.

However, the validity of the input data is not checked by BMS or the high-level language when the map is used, so any desired checking must be performed by the application program. The length of the data associated with "value" should be the same as that specified in the LENGTH operand if LENGTH is specified. If both PICIN and PICOUT (see below) are used, an error message is produced if their calculated lengths do not agree; the shorter of the two lengths is used. If neither PICIN nor PICOUT is coded for the field definition, a character definition of the field is automatically generated regardless of other operands that are coded, such as ATTRB=NUM.

As an example, assume that the following map definition is created for reference by a COBOL application program:

```
MAPX DFHMSD TYPE=DSECT,
        LANG=COBOL,
        MODE=INOUT
MAP DFHMDI LINE=1,COLUMN=1,
        SIZE=(1,80)
F1 DFHMDF POS=0,LENGTH=30
F2 DFHMDF POS=40,LENGTH=10,
        PICOUT='$$$,$$0.00'
F3 DFHMDF POS=60,LENGTH=6,
        PICIN='9999V99',
        PICOUT='ZZ9.99'
DFHMSD TYPE=FINAL
```

This generates the following DSECT:

```
01 MAPI.
02 F1L PIC S9(4) BINARY.
02 F1F PIC X.
02 FILLER REDEFINES F1F.
03 F1A PIC X.
02 F1I PIC X(30).
02 F2L PIC S9(4) BINARY.
02 F2F PIC X.
02 FILLER REDEFINES F2F.
```

BMS macros

```
03 F2A PIC X.
02 F2I PIC X(10).
02 F3L PIC S9(4) BINARY.
02 F3F PIC X.
02 FILLER REDEFINES F3F.
03 F3A PIC X.
02 F3I PIC 9999V99.
01 MAPO REDEFINES MAPI.
02 FILLER PIC X(3).
02 F10 PIC X(00030).
02 FILLER PIC X(3).
02 F20 PIC $$$,$$0.00.
02 FILLER PIC X(3).
02 F30 PIC ZZ9.99.
```

Note: The valid picture values for COBOL input maps are:

A P S V X 9 / (and)

PICOUT (COBOL only)

is similar to PICIN, except that a picture to be applied to an output field in the OUT or INOUT map is generated.

Note: The valid picture values for COBOL output maps are:

A B P S V X Z 0 9 , . + - \$
CR DB / (and)

POS

specifies the location of a field. This operand specifies the individually addressable character location in a map at which the attribute byte that precedes the field is positioned.

number specifies the displacement (relative to zero) from the beginning of the map being defined.

(line,column) specify lines and columns (relative to one) within the map being defined.

The location of data on the output medium is also dependent on DFHMDI operands.

The first position of a field is reserved for an attribute byte. When supplying data for input mapping from non-3270 devices, the input data must allow space for this attribute byte. Input data must not start in column 1, but may start in column 2.

The POS operand always contains the location of the first position in a field, which is normally the attribute byte when communicating with a 3270 device. For the second and subsequent fields of a group, the POS operand points to an assumed attribute byte position, ahead of the start of the data, even though no actual attribute byte is necessary. If the fields follow on immediately from one another, the POS operand must point to the last character position of the previous field in the group. See "Defining field groups" on page 553.

When a position number is specified that represents the last character position in the 3270 device, two special rules apply:

- ATTRB=IC must not be coded. The cursor can be set to location zero by using the CURSOR option of the SEND MAP, SEND CONTROL, or SEND TEXT command.

- If the field is to be used in an output mapping operation with the DATAONLY option on the SEND MAP command, an attribute byte for that field must be supplied in the symbolic map data structure by the application program.

PS specifies that programmed symbols are to be used. This overrides the PS operands of the DFHMSD and DFHMDI macros.

BASE is the default and specifies that the base symbol set is to be used.

psid specifies a single character, or an EBCDIC hexadecimal code of the form X'nn', that identifies the set of programmed symbols to be used.

Note: Only a psid value of 8 is supported.

The PS operand is ignored if the terminal does not support programmed symbols.

SOSI

specifies that the field may contain a mixture of EBCDIC and DBCS data. The DBCS subfields within an EBCDIC field are delimited by SO (shift-out) and SI (shift-in) characters. SO and SI both occupy a single screen position (normally displayed as a blank). They can be included in any non-DBCS field on output if they are correctly paired. The terminal user can transmit them inbound if they are already present in the field, but can add them to an EBCDIC field only if the field has the SOSI attribute.

TRANSP

determines whether the background of an alphanumeric field is transparent or opaque, that is, whether an underlying (graphic) presentation space is visible between the characters.

VALIDN

specifies that validation is to be used on a 5250 terminal. This overrides the VALIDN operands of the DFHMSD and DFHMDI macros.

MUSTFILL specifies that the field must be filled completely with data. An attempt to move the cursor from the field before it has been filled, or to transmit data from an incomplete field, raises the "inhibit input" condition.

MUSTENTER specifies that data must be entered into the field, but need not fill it. An attempt to move the cursor from an empty field raises the "inhibit input" condition.

TRIGGER specifies that this field is a trigger field.

The VALIDN operand is ignored if the terminal does not support validation.

```
XINIT1 DFHMDF POS=(02,23),LENGTH=23,SOSI=YES,
XINIT='E289958793850E42C142C242C342C442C50F82A8A3A5A2' *
```

XINIT

specifies, in hexadecimal form, constant or default data for an output field. Only one of XINIT, GINIT, and INITIAL may be specified.

Hexadecimal data is written as an even number of hexadecimal digits, for example, XINIT=C1C2. If the number of valid characters is smaller than the field length, the data is padded to the right with blanks. For example, if LENGTH=3, XINIT=C1C2 results in an initial field of "AB".

If hexadecimal data is specified that corresponds with line or format control characters, the results are unpredictable. The XINIT operand should therefore be used with care. In older versions of CICS, using XINIT was the only method of defining DBCS data in BMS maps, and consequently there are many existing maps in the mainframe environment which use the XINIT keyword for DBCS data. However, if you are coding a new map, (rather than porting an existing map from another CICS platform), you are strongly recommended to avoid the use of the XINIT keyword for DBCS data, and to use either the GINIT or the INITIAL keywords.

When the BMS translator is executing in DBCS mode, the XINIT data is processed as follows:

- If SOSI=YES is defined for the field, the data must conform to the rules for INITIAL data. If the LENGTH value exceeds the data length, the data will be padded out with SBCS blanks, not nulls.
- If PS=8 is defined for the field, the data must conform to the rules for GINIT data, except that SO and SI characters must not occur in the data. (This is to maintain compatibility with CICS for MVS/ESA maps).
- If neither SOSI=YES nor PS=8 is defined, the XINIT data is processed in the usual (non-DBCS) way.

The *redundant SO/SI* and *extended character continuation* techniques do not apply to the XINIT operand.

In the XINIT example, a DBCS subfield is embedded in SBCS data characters, that is, between the SO (X'0E') and SI (X'0F') characters. SOSI=YES is defined to indicate that this field is a mixed data field. Each SBCS character has a length of 1, each of the five DBCS characters has a length value of 2, and the SO and SI each has a length value of 1. Hence the total length is 23. The BMS translator retains the SO and SI characters embedded in the data and stores them in the physical map, because these characters must be transmitted to the device.

This example defines the same data string as the examples given for the INITIAL operand.

Sample map with DBCS data definitions

Figure 62 on page 577 shows a complete map definition for a map using DBCS characters.

```

CONTMAP DFHMSD
        TYPE=MAP,
        LANG=COBOL,
        MODE=INOUT,
        MAPATTS=(PS,SOSI),
        DSATTS=(PS,SOSI),
        TERM=3270-2,
        TIOAPFX=YES,
*
MAP1 DFHMDI
        SIZE=(24,80),
        LINE=1,
        COLUMN=1,
INIT1 DFHMDF
        POS=(03,01),
        LENGTH=78,
        INITIAL='abc<.F.R.E.D>defghijklmnopqrstuvwxyz0123456789A*
        BCD<.B.I.L.L>EFG'
INIT2 DFHMDF
        POS=(05,01),
        LENGTH=78,
        INITIAL='abc<.A.B.C.D>defghijklmnopqrstuvwxyz012<.M.A.N>*
        <.F.R.E.D>EFG'
INIT3 DFHMDF POS=(07,01),LENGTH=78,INITIAL='abcdefghijklmnop*****
        <.F.R.E.D>EFG'
INIT4 DFHMDF POS=(09,01),LENGTH=78,INITIAL='abcdefghijklmnop<.M.A.N>**
        <.F.R.E.D>EFG'
GINIT1 DFHMDF POS=(11,01),LENGTH=78,GINIT='<.H.U.R.S.L.E.Y .R.U.L>***
        <.E.S .O.K>',PS=8
        DFHMSD
        TYPE=FINAL
END

```

Figure 62. Sample map with DBCS data definitions

Appendix E. CICS-value data areas supported by CICS/400

This appendix lists the CICS-value data area (CVDA) symbolic names that are supported by CICS/400, together with their numeric values.

If a CVDA symbolic name that CICS/400 recognizes but does not support is passed as an argument to the DFHVALUE built-in function, the translator substitutes the correct numeric value regardless.

If an argument passed to DFHVALUE is not recognized by CICS/400, the translator generates an error message.

The CVDA tables are:

- A list of the supported CVDAs in alphanumeric sequence by symbolic name
- A list in ascending order by associated numeric values
- Values returned by the INQUIRE TERMINAL|NETNAME DEVICE command

For more information about CVDAs, see “CICS-value data areas (CVDAs)” on page 309.

CVDAs by symbolic name

Symbolic name	Value
ACQUIRED	69
ACTIVE	181
ADDABLE	41
ALLOCATED	81
ALTERNATE	197
APPCPARALLEL	374
ATI	75
AUXSTART	312
AUXSTOP	314
BROWSABLE	39
CEDF	370
CLOSED	19
CLOSING	21
COBOL	151
COLDSTART	266
CONFFREE	82
CONFRECEIVE	83
CONFSEND	84
DEFAULT	198
DELETABLE	43
DEST	235
DISABLED	24
DISABLING	25
DISK1	252
DISK2	253
DPLSUBSET	383
EMERGENCY	268
EMPTY	210

CVDAs

Symbolic name	Value
EMPTYREQ	31
ENABLED	23
ESDS	5
EXTRA	221
FIRSTQUIESCE	182
FIXED	12
FREE	85
FULL	212
FULLAPI	384
GOINGOUT	172
HOLD	163
C	149
INDIRECT	122
INPUT	226
INSERVICE	73
INTRA	222
INTSTART	310
INTSTOP	311
KSDS	6
LOGICAL	216
LUW	246
MAP	155
NEWCOPY	167
NOATI	76
NOCEDF	371
NOEMPTYREQ	32
NOHOLD	164
NOSWITCH	285
NOSYSDUMP	185
NOTADDABLE	42
NOTAPPLIC	1
NOTBROWSABLE	40
NOTDELETABLE	44
NOTEMPTY	211
NOTERMINAL	214
NOTINIT	376
NOTPURGEABLE	161
NOTRANDUMP	187
NOTREADABLE	36
NOTRECOVERABLE	30
NOTTI	78
NOTUPDATABLE	38
NOWAIT	341
OPEN	18
OPENING	20
OPENOUTPUT	257
OUTPUT	227
OUTSERVICE	74
PENDFREE	86
PENDRECEIVE	87
PHASEIN	168
PROGRAM	154
PURGE	236
PURGEABLE	160

Symbolic name	Value
READABLE	35
RECEIVE	88
READBACK	209
RECOVERABLE	29
RELEASED	70
REMOTE	4
RRDS	7
SEND	90
SIGNEDOFF	245
SIGNEDON	244
SWITCHALL	287
SYSDUMP	184
TASK	233
TERM	234
TERMINAL	213
TRANDUMP	186
TTI	77
T3277R	145
T3284L	155
T3790SCSP	182
UPDATABLE	37
USEROFF	322
USERON	321
VARIABLE	13
VSAM	3
VTAM	60
WAIT	340
WARMSTART	267

CVDAs by numeric value

Value	Symbolic name
1	NOTAPPLIC
3	VSAM
4	REMOTE
5	ESDS
6	KSDS
7	RRDS
12	FIXED
13	VARIABLE
18	OPEN
19	CLOSED
20	OPENING
21	CLOSING
23	ENABLED
24	DISABLED
25	DISABLING
29	RECOVERABLE
30	NOTRECOVERABLE
31	EMPTYREQ
32	NOEMPTYREQ
35	READABLE

CVDAs

Value	Symbolic name
36	NOTREADABLE
37	UPDATABLE
38	NOTUPDATABLE
39	BROWSABLE
40	NOTBROWSABLE
41	ADDABLE
42	NOTADDABLE
43	DELETABLE
44	NOTDELETABLE
60	VTAM
69	ACQUIRED
70	RELEASED
73	INSERVICE
74	OUTSERVICE
75	ATI
76	NOATI
77	TTI
78	NOTTI
81	ALLOCATED
82	CONFFREE
83	CONFRECEIVE
84	CONFSEND
85	FREE
86	PENDFREE
87	PENDRECEIVE
88	RECEIVE
90	SEND
122	INDIRECT
145	T3277R
149	C
151	COBOL
154	PROGRAM
155	MAP
155	T3284L
160	PURGEABLE
161	NOTPURGEABLE
163	HOLD
164	NOHOLD
167	NEWCOPY
168	PHASEIN
172	GOINGOUT
181	ACTIVE
182	FIRSTQUIESCE
182	T3790SCSP
184	SYSDUMP
185	NOSYSDUMP
186	TRANDUMP
187	NOTRANDUMP
197	ALTERNATE
198	DEFAULT
209	READBACK
210	EMPTY
211	NOTEMPTY
212	FULL

Value	Symbolic name
213	TERMINAL
214	NOTERMINAL
216	LOGICAL
221	EXTRA
222	INTRA
226	INPUT
227	OUTPUT
233	TASK
234	TERM
235	DEST
236	PURGE
244	SIGNEDON
245	SIGNEDOFF
246	LUW
252	DISK1
253	DISK2
257	OPENOUTPUT
266	COLDSTART
267	WARMSTART
268	EMERGENCY
285	NOSWITCH
287	SWITCHALL
310	INTSTART
311	INTSTOP
312	AUXSTART
314	AUXSTOP
321	USERON
322	USEROFF
340	WAIT
341	NOWAIT
370	CEDF
371	NOCEDF
374	APPCPARALLEL
376	NOTINIT
383	DPLSUBSET
384	FULLAPI

CVDAs returned by the INQUIRE TERMINALINETNAME DEVICE command

CVDA sequence CVDA	Value
T3277R	145
T3284L	155
T3790SCSP	182

CVDAs

Appendix F. Notices

This information was developed for products and services offered in the U.S.A.

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 give you any license to these patents. You can send license inquiries, in writing, to:

| IBM Director of Licensing
| IBM Corporation
| 500 Columbus Avenue
| Thornwood, NY 10594-1785
| U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or region or send inquiries, in writing, to:

| IBM World Trade Asia Corporation
| Licensing
| 2-31 Roppongi 3-chome, Minato-ku
| Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country or region where such provisions are inconsistent with local law: 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 states 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 Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply 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 Corporation
Software Interoperability Coordinator, Department 49XA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

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 information 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.

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. 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 IBM's application programming interfaces.

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_. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming Interface Information

This *CICS for iSeries Application Programming Guide* documents intended Programming Interfaces that allow the customer to write programs to obtain the services of CICS/400.

Trademarks

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

AD/Cycle
AS/400
CICS
CICS/400
CICS/ESA
COBOL/400
CUA
Distributed Relational Database Architecture
DRDA
e (logo)
IBM
iSeries
MVS
Operating System/400
OS/390
OS/400
S/370
SAA
System/36
System/370
System/38
System/390
Systems Application Architecture
VSE/ESA
VTAM
WebSphere
zSeries
z/OS

Other company, product, and service names may be trademarks or service marks of others.

Glossary

This glossary defines special CICS terms used in this book; and words used differently from their everyday meaning.

If you can't find the term you are looking for, try the glossary in one of the books mentioned in the bibliography, or the *IBM Dictionary of Computing*, New York: McGraw-Hill, 1994.

A

Advanced Program-to-Program Communication (APPC). The general term for LUTYPE6.2 protocol under Systems Network Architecture (SNA).

AID. See *automatic initiate descriptor*.

alternate index. An index based on an alternate key. It allows the file to be processed in a secondary key order.

API. See *application programming interface*.

API commands. CICS commands supported for EXEC CICS statements. CICS/400 includes a subset of commands supported by CICS on larger systems.

API translator. See *translator*.

APPC. See *Advanced Program-to-Program Communication*.

application programming interface (API). The facility that allows application programmers to access CICS facilities by including EXEC CICS commands in their programs.

asynchronous processing. Processing in which a local transaction is allowed to continue while having a remote request processed; for example, a remote transaction can be started and data stored for it by the local transaction.

ATI. See *automatic transaction initiation*.

attention identifier (AID) keys. The ENTER key, the CLEAR key, the PF keys, and the PA keys are all known as attention identifier (AID) keys. The ENTER key and the PF keys transmit data from the terminal's buffer to the system; the others just give an indication of which key was pressed.

attribute. Attributes control the field's appearance and operation. Color or highlighting can be chosen for some terminals. Fields can be either unprotected (used for data entry) or protected (used for display only).

attribute byte. For standard attributes in BMS, a single byte that occupies the character location on the screen that is immediately before the field to which it applies.

automatic initiate descriptor (AID). When an ICE expires for a timer-related task, it then becomes an AID. If all its required resources are available it becomes an enabled AID; if it is waiting for a resource to become free, it is a suspended AID. See also *interval control element*.

automatic transaction initiation (ATI). When data is sent to an intrapartition destination and the number of entries reaches a predefined trigger level, it can be specified that a transaction be started automatically to process the data in the intrapartition destination queue. A transaction can also start automatically another transaction at a specified terminal. See *control interval*.

auxiliary storage TS queue. A temporary storage queue that is in a physical file managed by CICS. Auxiliary storage should be used to store large amounts of data, or data needed for a long period of time. Contrast with *main storage TS queue*.

auxiliary trace. An option whereby trace entries are written to an external file. Any or all of these trace entries can then be printed to help you search for a problem. Contrast with *internal trace*.

B

base key. A partial key definition beginning with a known base (for example, SAMPLE-SAMPLE1, SAMPLE2, SAMPLE3 would be found on sequential reads). Used in VSAM.

basic mapping support (BMS). A facility that handles data stream input and output from a terminal. Its use provides device and format independence for application programs.

batch shell. A shell started to handle interval control timer requests. The batch shell is transparent to the user; each user's program runs under its own user shell. Contrast with *user shell*.

BMS. See *basic mapping support*.

BMS, support provided. Support is provided for 3270 displays and printers only. In CICS/400, BMS supports extended attributes and large screens. It does not support cumulative mapping, terminal operator paging, routing, or message switching.

bottleneck. A symptom that characterizes a performance problem. It can be due to a task failing to start, failing to continue after starting, or taking a long time to complete.

bracket protocol. Use of this protocol prevents the interruption of a transaction between CICS and a logical unit. Bracket protocol is used when CICS communicates with specific logical units.

browse. Sequential reading of a file or temporary storage queue, beginning with a specified record.

C

CCSID. A coded character set identifier used to convert one character or graphic value to another.

CEBR. A supplied transaction that allows the user to browse temporary storage (TS) queues from a CICS user shell environment.

CECI. The command-level interpreter transaction. This transaction allows application programmers to check syntax interactively, and test their API commands before incorporating them into CICS application programs.

CECS. A transaction that allows syntax checking of API commands.

CEDA. The resource definition transaction. A supplied transaction used to handle the manipulation of the CICS table definitions.

CEDE. The transaction used to start the execution diagnostic facility (EDF). See *execution diagnostic facility* for a more detailed description.

CEMT. The master terminal transaction. This transaction allows the system administrator to inquire about or change the status of resources (including programs, transactions, files, queues, and terminals).

centralized processing. Processing in which the application is processed on a central processor, which users access using nonintelligent terminals. Contrast with *distributed processing*.

CESF. Supplied transaction to sign off from CICS. This transaction signs the user off from the user shell.

CICS-value data area (CVDA). A CICS-supplied value for certain data options on EXEC CICS commands.

CL. See *control language*.

client/server. The model of interaction in distributed data processing in which a program at one site sends a request to a program at another site and awaits a response. The requesting program is called a client; the answering program is called a server.

COBOL/400. The AD/Cycle COBOL compiler available for OS/400. This is one of the two supported languages for CICS/400 application development.

cold start. One of the ways in which temporary storage and transient data queues are recovered when a CICS control region is started. A cold start indicates that these resources are cleared. Contrast with *warm start* and *emergency start*.

command. In an OS/400 environment usually refers to a CL command. Each CL command corresponds to a specific operation. Use of a CL command is usually quicker than the corresponding menu selections. In CICS, an instruction similar in format to a high-level language instruction; the command statement begins with EXEC CICS.

COMMAREA. A communication area used for passing data between programs within a transaction or between transactions from the same terminal.

commit. Changes made to resources are written or committed at a syncpoint.

common work area (CWA). A work area that can be accessed by any transaction in the CICS system.

control block. A specialized storage area in shared system storage that is used by CICS to pass parameters between service modules.

control language (CL). The primary interface to the OS/400 operating system. Each command name refers to a command processing program in the system that performs the actions indicated by the command.

control region (CR). The control region provides the control, scheduling, and work management mechanisms necessary to coordinate all the shared resources in CICS/400. A control region is started using the STRCICS CL command and ended using the ENDCICS CL command.

conversation. A sequence of exchanges over a session, delimited by SNA brackets.

CR. See *control region*.

CRTE. A supplied transaction used for routing transactions to another CICS system.

cursor position, normal. You can specify a 2-byte cursor position on the BMS SEND commands. This enables you to specify the absolute value of the cursor position on the screen. Contrast with *cursor position, symbolic*.

cursor position, symbolic. If the length of a field is set to -1, CICS places the cursor at the first of these fields. Often used by the programmer when data entry fields are in error. Contrast with *cursor position, normal*.

CVDA. See *CICS-value data area*.

CWA. See *common work area*.

D

data description specification (DDS). Traditionally data attributes were described in the programs themselves. With OS/400 it is possible to describe data external to the application program that processes the files. Data descriptions specifications can be used to describe either physical or logical files.

data link protocol. A set of rules for data communication over a data link, in terms of a transmission code, a transmission mode, and control and recovery procedures.

data path. A path is an alternate way of extracting data from a file. Usually in key order, but not the major key. Used for an alternate view of a file.

database administrator. The person responsible for the design, development, integrity, and maintenance of databases.

DBCS. See *double byte character set*.

DCT. See *destination control table*.

DDS. See *data description specification*.

deadlock. A contention for resources, where two programs are attempting to use the same resource (for example, update the same record) at the same time.

deferred work element (DWE). The deferred work element is the catalyst used to invoke "event driven" services controlled within CICS/400. DWE's cause a unit of work to be scheduled later, normally either at task termination or before or after syncpoint.

dequeue. CICS/400 release a resource held for exclusive use.

destination control table (DCT). A table describing each of the transient data destinations used in CICS. This table contains an entry for each extrapartition, intrapartition, and indirect destination.

DFHAID. A COBOL copybook containing the symbolic names for all the AID keys.

DFHBMSCA. Attribute and control character list, for use with COBOL programs. A copybook containing the symbolic names for the various combinations of attributes and control characters.

DFHCOMMAREA. The communication area in the linkage section of an OS/400 COBOL program. See also *COMMAREA*.

DFHMDF. The BMS macro that defines a field within a map.

DFHMDI. The BMS macro that defines a map within a map set.

DFHMDS. The BMS macro that defines a map set.

distributed processing. Processing in which application transaction programs, distributed among interconnected processors, cooperate to perform applications for end users of a network. Contrast with *centralized processing*.

distributed program link (DPL). This enables an application program running on one CICS system to link to another application program running in another CICS system.

distributed transaction processing (DTP). Processing in which a CICS transaction communicates with a transaction running in another system.

double byte character set (DBCS). The graphic character set used in Asian countries or regions such as Japan, China, Taiwan, and Korea.

DPL. See *distributed program link*.

DTP. See *distributed transaction processing*.

dump control. This facility handles program controlled dumping of the application program. In CICS/400, also referred to as *serviceability and dump*.

DWE. See *deferred work element*.

E

EDF. See *execution diagnostic facility*.

EIB. See *EXEC interface block*.

emergency start. One of the ways in which temporary storage and transient data queues are recovered when a CICS control region is started following an abnormal shutdown of the region. An emergency start affects these resources as defined in the system initialization table. This may result in one or more queues being cleared or recovered, that is, returned to their state prior to shutdown. Contrast with *cold start* and *warm start*.

emulation. An imitation of one computer or product by another so that both accept the same data, and achieve similar results.

enqueue. Hold a user-defined resource for exclusive use.

entry-sequenced data set (ESDS). One of the file organizations supported to emulate VSAM. Each record is identified by its relative byte address (RBA). Records are held in the order they were first entered, with new records added at the end.

error message. Under OS/400, error messages (abends) from failed routines are displayed at the terminal, and additional help is available by using the PF1 key.

ESDS. See *entry-sequenced data set*.

EXEC (EXECUTE) statement. An instruction similar in format to a high-level language instruction. Begins with EXEC CICS, lists the command and options, and ends with END-EXEC.

EXEC interface block (EIB). A control block associated with a CICS task, used for direct communication between CICS and command-level application programs. Several fields in the EIB, such as the RESP and RESP2 fields, are often checked by the programmer to determine whether a CICS command was executed as expected.

execution diagnostic facility (EDF). A facility that helps the application programmer to debug an application by stepping through its CICS commands. The programmer can change values in the application while it is running. To start EDF, you need to use the CEDF transaction.

extrapartition destination. A type of transient data queue. Extrapartition destinations can be accessed either within the CICS environment or outside CICS/400; they can be defined as either input or output.

F

FCT. See *file control table*.

file control. The facility for managing basic operations against a file (ADD, READ, DELETE, REWRITE, and BROWSE).

file control table (FCT). A table containing the characteristics of files accessed by file control.

function shipping. The process in which CICS accesses resources when those resources are actually held on another CICS system.

G

GETMAIN area. A requested area of transaction storage that is outside the working-storage section of your program.

I

ICE. See *interval control element*.

ILE. See *integrated language environment*.

ILE C. An IBM licensed program that is a Systems Application Architecture platform C programming language. The ILE C program uses the ILE model on the iSeries system.

indirect destination. A type of transient data destination that points to another destination within the DCT. See *destination control table*.

initialization. The preparation of a system, device, or program for operation. An operating system is initialized on startup or after a system failure.

initialization stall. A wait that occurs during initialization when a CICS system appears to be running normally but is not actually progressing through the various stages of initialization.

installation verification procedure (IVP). This provides a sample online and maintenance application to verify a successful installation of CICS/400.

integrated language environment (ILE). A set of constructs and interfaces that provides a common run-time environment and run-time bindable application program interfaces for all ILE-conforming high-level languages.

interception point. In the execution diagnostic facility (EDF), a point at which execution is interrupted to allow the display of program information (see *execution diagnostic facility*).

internal trace. An option whereby trace entries are written to an internal control region table. The table, which can be specified to wrap when full, is most appropriate if you do not need to capture a large number of trace entries. Contrast with *auxiliary trace*.

intersystem communication (ISC). This facility provides inbound and outbound support for communication from other computer systems.

interval control. The primary task of this facility is the handling, synchronization, and initiation of tasks requested by user application programs and CICS internal service routines. Other functions include obtaining the formatted time for the user.

interval control element (ICE). An entry under interval control that is waiting in an unexpired state. Its defined date and time (to become current) is still in the future. When an ICE expires it becomes an AID. See also *automatic initiate descriptor*.

intrapartition destination. A type of transient data queue used subsequently as input data to another task within CICS.

ISC. See *intersystem communication*.

IVP. See *installation verification procedure*.

J

JCT. See *journal control table*.

journal control. Provides the CICS user with the ability to write CICS journal records when required by the application for auditing purposes.

journal control table (JCT). Describes the CICS user journals along with their access characteristics.

K

key-sequenced data set (KSDS). One of the file organizations supported to emulate VSAM. Each record in the file is identified by a key within a predefined position of the record. Each key must be unique.

KSDS. See *key-sequenced data set*.

L

limited capability. The use of certain OS/400 commands can be restricted by setting a user's profile to limited capability.

linkage section. The section of a COBOL program that allows the programmer to access areas outside of working-storage.

local. Pertaining to the system, program, or device being described, as opposed to other systems, programs, or devices which are on other computer systems. Contrast with *remote*.

logical unit (LU). A port through which a user gains access to the services of networks.

logical unit of work (LUW). The work processed between syncpoints. In CICS/400, an implicit syncpoint occurs at normal task completion.

LU. See *logical unit*.

LUW. See *logical unit of work*.

M

macro. An assembler-language instruction. Except for the macros that BMS uses, CICS/400 supports command-level instructions only.

main storage TS queue. A dynamic storage area managed by CICS under the temporary storage facility. Data in main storage does not survive from one CICS run to the next. Contrast with *auxiliary storage TS queue*.

map. Screen data defined as fields using BMS macros that are generated into two forms: a physical map for display on part or all of a device, and a symbolic map referring to just the named input and output fields that

are needed by the program processing the screen data. Maps are grouped together in a *map set*.

map set. Defines the related screens used for one transaction. Each map set is a set of assembler-language macros that are used to create a physical map set, and a symbolic map set. See both *physical map set* and *symbolic map set*.

MAPFAIL. When using BMS, a MAPFAIL condition occurs (on input only) if the data to be mapped has a length equal to zero. This happens if a PA key or the CLEAR key is pressed and the modified data tag is turned off.

MDT. See *modified data tag*.

message file. The file holding the text of all CICS messages.

modified data tag (MDT). The modified data tag is turned on whenever fields are modified by the terminal operator. This tag may also be turned on by the programmer to assure that the field is returned with the next inbound transmission.

multithread test. This type of test involves several concurrently active transactions. Whether the new function can coexist with other related functions is tested. Contrast with *single-thread test*.

N

national language support (NLS). A feature that allows the user to communicate with the system in the national language chosen by the user.

network. The hardware equipment (for example, terminals and lines) that supplies the connections between terminals and hosts or other systems.

NLS. See *national language support*.

nonrecoverable (requests). Requests on a queue or in a file that are lost in the event of a transaction or system failure. Contrast with *recoverable*.

nonshared storage. Storage areas created for and used exclusively by individual user shells.

O

object oriented system. Under OS/400, everything is treated as an object; that is, files, programs, user spaces, and data queues are all treated as objects.

P

panel. The screen layout of such items as the initial cursor position, entry fields, selection fields, and list fields.

PCT. See *program control table*.

PF keys. Program function keys used within CICS and OS/400 to perform certain preprogrammed functions. For example, PF3 is normally set up to take you back to a previous screen.

physical map set. The facility that BMS uses to map data between the program and the terminal.

PPT. See *processing program table*.

precompiler. A program that manages preparation of source code for compilation. See also *translator*. Often used as synonyms.

printer spooling. This CICS/400 facility provides support for writing data to OS/400 print spools. Only *printed* output is supported by CICS/400.

processing program table (PPT). A table defining the application programs and BMS maps that can be run under CICS.

program check. This indicates that an error in a program has caused the CICS transaction to terminate abnormally under the control of CICS, which issues appropriate messages.

program control. This facility handles the flow of control among application programs.

program control table (PCT). A table defining the transactions that can be processed by the system. Each transaction identifier is paired with the name of the program CICS executes when the transaction is invoked.

protected attribute. If this attribute is chosen, the terminal operator cannot key data into this field; the field is display only.

protocol, data link. A set of rules for data communication over a data link, in terms of a transmission code, a transmission mode, and control and recovery procedures.

pseudoconversational programming. A programming technique in which every SEND or SEND MAP must be followed by a RETURN TRANSID command. The program exits after sending data to the terminal, rather than waiting for an operator response.

Q

queue. A line or list formed by items in a system waiting for service; for example, tasks to be performed, or messages to be transmitted.

R

RDO. See *resource definition online*.

recoverable (requests). Requests on a queue or in a file that can be restored to their previous state at the exit of the last task or syncpoint after a transaction or system failure.

reentrant code. When a program is reentrant it is serially reusable. Each time you enter the program a fresh copy of working-storage is provided. If any values need to be saved, you must save them in other storage areas or files.

regression test. A complete test of the existing system along with any changes.

relative record data set (RRDS). One of the file organizations supported to emulate VSAM. Records are of fixed length and identified by a relative record number (RRN). When you add a new record, you can either specify the RRN or let VSAM assign the next sequential number.

remote. Pertaining to a system, program, or device that is accessed through a telecommunication line. Contrast with *local*.

request unit (RU). The unit of data from a logical unit. One or more of these RUs can be grouped together and handled as a chain. This is often done to transmit data over a line.

resource definition online (RDO). This facility lets you define certain CICS resources interactively while CICS is running. Specifically RDO lets you define terminals, programs, and transactions interactively. Its use is normally restricted to the system administrator.

resource management. The facility that keeps track of what system resources are being used by mapping the CICS identification name to the underlying system resources.

rollback. The process of canceling changes made by a transaction to recoverable resources, following the failure of that transaction.

routing transaction. A supplied transaction (CRTE) that allows an operation at a terminal owned by one CICS system to sign on to another CICS system connected by an APPC link.

RRDS. See *relative record data set*.

RU. See *request unit*.

runaway task. A transaction that features instructions that are executed repeatedly without executing a condition to terminate the loop.

S

SBA. See *set buffer address*.

screen design aid (SDA). A utility that generates data description specifications for screens. It allows you to see the display you are creating or modifying.

SDA. See *screen design aid*.

security. A mechanism to ensure that resources are protected from unauthorized access.

server program. A program executing in a remote system that is linked to by a program in the client system using a remote link request. See also *distributed program link*.

server system. The CICS system to which a client system ships a link request.

serviceability and dump. See *dump control*.

set buffer address (SBA). A data stream control field used in both inbound and outbound data streams. Indicated by X'11'. Outbound data streams contain SBAs to describe the position in which user data is placed. SBAs in the inbound data stream describe to BMS what fields were entered by the user.

shared storage. Storage shared between a control region and other processes.

shell. The CICS facility that provides the work management mechanism to build and refresh the application programming environment needed to run CICS transactions. A shell is started using the STRCICSUSR CL command and ended using the ENDCICSUSR CL command. See also *user shell*.

shutdown. The process of terminating a CICS control region in a controlled way by using the CEMT transaction or as a result of a system failure.

single-thread test. Test of a single application or transaction running by itself. Contrast with *multi-thread tests*.

SIT. See *system initialization table*.

SNA. See *Systems Network Architecture*.

space objects. System objects that manage storage requests for internal CICS and user application requirements, that is, local space, system space, and user space.

SQL. See *structured query language*.

stop condition. A specified condition for stopping the execution of a transaction which is being debugged by the execution diagnostic facility (EDF).

storage control. This facility controls requests for main storage to provide intermediate work areas not automatically provided by CICS. The use of these areas is requested with a GETMAIN command.

structured query language (SQL). A language that is used to access data in a relational database by using simple conversation type statements.

supplied transactions. General purpose transactions supplied by CICS/400 to perform general CICS functions required by many users, such as debugging (CEDF).

switchable file. In the CICS/400 journal control facility, a new CICS journal file is created and opened when an out-of-space condition occurs during a CICS journal write operation.

symbolic map set. A logical representation of a set of BMS map definitions in either COBOL copybook or C header file format. The symbolic maps define the format of the screen's data, and provide reference names for each field and its attributes.

syncpoint. The commitment or backout point of OS/400 recoverable resources within a task's logical unit of work. A syncpoint may be either explicitly requested, or implicitly initiated at task termination.

system administrator. Any person authorized to use or modify resources or use the CEMT transaction.

system initialization table (SIT). A table containing parameters used to start the control region's system initialization process.

system storage. These storage areas are used specifically for internal CICS requirements. Contrast with *user storage space*.

systems information access. This facility provides the OS/400 system administrator and application developers with the ability to inquire and modify certain CICS runtime resource definitions.

Systems Network Architecture (SNA). The description of the logical structure, formats, protocols, and operational sequences for transmitting information units through, and controlling the configuration and operation of, networks.

T

tables. Resources used in the CICS/400 system are stored in resource tables and used to control the interaction between the different components in the system, for example, programs and terminals.

task. The execution of an application program (or perhaps several application programs) for a specific user. Several users may invoke the same transaction, but each execution of that transaction is treated as a separate task.

task control. This facility controls resources at task syncpoint, and at normal or abnormal termination.

TCS. See *terminal control system table*.

TCT. See *terminal control table*.

TCTUA. See *terminal control table user area*.

TD. See *transient data*.

temporary storage (TS). The facility that allows application programs to store data in a temporary storage queue for later retrieval.

temporary storage table (TST). A table describing temporary storage queues. For temporary storage data to be recoverable by CICS, if the system terminates abnormally, data identifiers for the temporary storage queue must be specified in the TST.

terminal. In CICS, a device equipped with a keyboard and a display, and capable of sending and receiving information over a communication channel.

terminal control. This facility handles addressing, and transmission error detection and correction for terminals (both displays and printers) associated with the local CICS system. It also handles the intercommunication data queue used by CICS.

terminal control system table (TCS). This table defines the connections between CICS systems.

terminal control table (TCT). A table describing the terminals and logical units within a CICS network.

terminal control table user area (TCTUA). An area used to pass information between application programs, but only if the same terminal is associated with the application programs involved.

timer-related event. An interval control function that is used to support events that are delayed, suspended, or restarted after a time interval.

trace. A debugging aid for system and applications programmers. This facility produces trace entries at set system points or in response to trace commands.

transaction. A transaction is identified to CICS by a 4-character code called a transid. It is often invoked when an operator enters a transid in column 1 on line 1 of a terminal.

transaction routing. The facility that provides support for inbound and outbound terminal requests from another CICS system connected by an APPC link.

transaction work area (TWA). A work area which exists only for the duration of a transaction. Consequently, you can use it to pass data among programs executed in the same transaction, but not between transactions.

transient data (TD). This facility provides the user with the ability to read and write data in sequential queues.

translator. A program that converts (translates) EXEC CICS commands found in source code into host language statements, including variable assignments and a call to the CICS EXEC interface program. See also *precompiler*.

trigger level. The number of requests for output to a CICS transient data queue that must be reached before automatic transaction initiation occurs.

TS. See *temporary storage*.

TST. See *temporary storage table*.

TWA. See *transaction work area*.

two-phase commit. The protocol that permits updates to protected resources to be committed or rolled back as a unit. During the first phase, partners in a conversation are asked if they are ready to commit. If all partners respond positively, they are asked to commit their updates. Otherwise, they are asked to roll back their updates.

U

user-based pricing. A pricing option that provides the capability for the customer to pay for a licensed program on the basis of the number of users.

user liaison. The department responsible for communication between data processing functions and end users.

user shell. An interactive shell initiated by a user by issuing a STRCICSUSR CL command. The user's CICS application programs run directly under this CICS facility. See also *shell*.

user storage space. These storage areas are obtained and managed by CICS, but are used by application modules requesting main storage. Contrast with *system storage*.

user trace. An option whereby an application program can write trace entries containing user-defined trace areas. User trace entries can be written to either internal or auxiliary trace.

V

virtual storage access method (VSAM). In CICS, support to access all three types of emulated VSAM files is provided: KSDS, ESDS and RRDS. Refer to separate types for more detailed information.

VSAM. See *virtual storage access method*.

W

wait. A state in which a task's execution has been suspended. One common cause, is a task waiting for a resource to become available.

warm start. One of the ways in which temporary storage and transient data queues are recovered when a CICS control region is started following a normal shutdown of the region. A warm start affects these resources as defined in the system initialization table. This may result in one or more queues being cleared or recovered, that is, returned to their state prior to shutdown. Contrast with *cold start* and *emergency start*.

Index

Numerics

- 3270 Information Display System
 - attribute characters 144
 - color select 145
 - field concepts 143
 - input operations 141
 - screen sizes 147
- 5250 Information Display System
 - color select 145
 - emulation of 3270 141
 - input operations 141

A

- ABCODE option
 - ABEND command 323
 - ASSIGN command 328
- abend codes, table of 540
- ABEND command 323
- ABEND exit, reactivating 369
- abend support commands 319
- abend user task, EDF 235
- abends 102
- abnormal termination
 - recovery 109
 - task 369
- ABSTIME option
 - ASKTIME command 327
 - FORMATIME command 362
- access to system information
 - ADDRESS command 324
 - ASSIGN command 327
 - CICS storage areas 324
 - EXEC interface block (EIB) 211
- ACCESSMETHOD option
 - INQUIRE CONNECTION command 481
 - INQUIRE FILE command 484
- ACQSTATUS option
 - INQUIRE CONNECTION command 482
 - INQUIRE TERMINAL command 501
 - SET TERMINAL command 520
- activation groups 53
- activation unit 25
- ADD option
 - INQUIRE FILE command 485
 - SET FILE command 511
- adding records 123
- ADDRESS command 324
- address, cursor 551
- AFTER option
 - START command 449
- AID (attention identifier)
 - 3270 input operation 142
 - EIBAID field 162
 - HANDLE AID command 163
- ALARM option
 - SEND CONTROL command 435
 - SEND MAP command 437
- ALARM option (*continued*)
 - SEND TEXT command 439
- ALARM value
 - CTRL operand, DFHMDI macro 563
 - CTRL operand, DFHMSD macro 557
- ALLOCATE command 325
- alternate screen size 147
- ALWBLK parameter
 - CRTCICSC CL command 297
 - CRTCICSCBL CL command 279
- ALWCPYDATA parameter
 - CRTCICSC CL command 296
 - CRTCICSCBL CL command 278
- ANY option
 - GETMAIN command 367
- ANYKEY option 163
- HANDLE AID command 371
- APPC logical unit
 - acquiring session to 325
 - initiating conversation with 335
 - returning mapped sessions to CICS 364
 - sending and receiving 337
- APPC mapped conversation
 - commands 319
 - ensuring transmission of accumulated data 461
 - receiving data 410
 - sending data 430
- APPC mapped conversations
 - abending 374
 - extracting attributes of 359
 - informing partner of error 377
 - issuing a positive response 375
 - requesting change of direction 379
 - retrieving values from attach header 360
 - returning sessions to CICS 364
- application programs
 - asynchronous processing 185
 - compiling 16, 40
 - design considerations 61
 - distributed program link 176
 - distributed transaction processing 186
 - efficiency 67
 - function shipping 176
 - intercommunication considerations 175
 - logical levels 27, 53, 200
 - performance considerations 82, 83
 - transaction routing 175
- application shell facility
 - application shell environment 191
 - transaction scheduling 191
 - work flow
 - initialization 192
 - processing 192
 - shutdown 192
- APPLID option
 - ASSIGN command 328
- architecture, base and towers 139
- argument values
 - COBOL programs 308
 - ILE C programs 309
- ASA option
 - SPOOLOPEN OUTPUT command 442
- ASIS option
 - basic mapping support 161
 - RECEIVE (5250 or 3270 logical) command 413
 - RECEIVE MAP command 415
- ASKIP value
 - ATTRB operand, DFHMDF macro 568
- ASKTIME command 193
 - request current time of day 326
- ASSIGN command 327
- assigning CVDA values 310
- asynchronous page build 167
- asynchronous processing 4, 185
- AT option
 - START command 449
- ATI (automatic transaction initiation) 216
- ATIFACILITY option
 - INQUIRE TDQUEUE command 496
 - SET TDQUEUE command 518
- ATISTATUS option
 - INQUIRE TERMINAL command 501
 - SET TERMINAL command 520
- ATTTERMID option
 - INQUIRE TDQUEUE command 496
 - SET TDQUEUE command 518
- ATTTRANID option
 - INQUIRE TDQUEUE command 496
 - SET TDQUEUE command 519
- attention identifier (AID) 371
 - EIBAID field 551
 - HANDLE AID command 550
 - list of constants, DFHAID 548, 550
 - terminal control 551
- attention identifier list 548
- ATTRB operand
 - DFHMDF macro 568
- attributes
 - characters 144, 154
 - control character list, DFHBMSCA 545
- attributes, extended 165
- AUTINSTMODEL option
 - DISCARD command 477
 - INQUIRE AUTINSTMODEL NEXT command 481
 - INQUIRE command 480
- automatic transaction initiation (ATI) 216
- autoskip field 144
- AUXILIARY option
 - WRITEQ TS command 472
- auxiliary temporary storage 73

- auxiliary temporary storage data 219
- auxiliary trace 67
- AUXSTATUS option
 - INQUIRE TRACEDEST command 505
 - SET TRACEDEST command 522

B

- back out to a syncpoint 458
- background transparency 146
- backout of resources 106
- base and towers architecture 139
- base color 145
- base file, accessing 116
- BASE operand
 - DFHMSD macro 556
- BASE value
 - PS operand, DFHMDF macro 575
 - PS operand, DFHMDI macro 565
 - PS operand, DFHMSD macro 559
- basic mapping support (BMS)
 - BMS support 141
 - cataloging maps 148
 - commands 319
 - coordinating BMS and another screen manager 159
 - creating maps 148
 - cursor position 159
 - data streams 77
 - device control options 158
 - exception conditions 162
 - field data format 153
 - field definition macro, DFHMDF 148, 553, 567
 - incorrect data 155
 - input field suffix 153
 - map definition macro, DFHMDI 147, 553, 562
 - map set definition macro, DFHMSD 147, 553, 555
 - map set suffixing 149
 - map set termination 148
 - map sets 136
 - mapping input data 160, 415
 - mapping output data 155
 - maps 151
 - output field suffixes 153
 - physical map 149
 - pregenerated versions 139
 - related constants 545
 - sending data to a display 155
 - sending text data (SEND TEXT) 164
 - symbolic map 148, 152
- batch shells 191
- BELOW option
 - GETMAIN command 367
- BIF DEEDIT command 332
- blank fields 78
- blank lines and printers 168
- BLANK value
 - JUSTIFY operand, DFHMDF macro 572
- BLINK value
 - HIGHLIGHT operand, DFHMDF macro 571

- BLINK value (*continued*)
 - HIGHLIGHT operand, DFHMDI macro 564
 - HIGHLIGHT operand, DFHMSD macro 558
- BOTTOM command, CEBR transaction 246
- BOX value
 - OUTLINE operand, DFHMDF macro 573
 - OUTLINE operand, DFHMDI macro 565
 - OUTLINE operand, DFHMSD macro 559
- bracket protocol, LAST option 172
- bright intensity field 144
- browse operation
 - ending 353
 - files 76
 - INQUIRE AUTINSTMODEL command 480
 - INQUIRE CONNECTION command 483
 - INQUIRE FILE command 487
 - INQUIRE JOURNALNUM command 489
 - INQUIRE PROGRAM command 492
 - INQUIRE TDQUEUE command 499
 - INQUIRE TERMINAL command 504
 - INQUIRE TRANSACTION command 508
 - read next record 395
 - read previous record 400
 - records 119
 - reset starting point 418
- BROWSE option
 - INQUIRE FILE command 485
 - SET FILE command 511
- BROWSE TEMP STORAGE option, CEDF 235
- browsing
 - PROGRAM entries 490
 - resource definitions 315
 - TERMINAL entries 500
 - TRANSACTION entries 506
- BRT value
 - ATTRB operand, DFHMDF macro 568
- BTRANS option
 - ASSIGN command 328
- BUFFER option
 - RECEIVE (5250 or 3270 logical) command 413
- built-in function 319
- BUSY option
 - SET FILE command 512

C

- CANCEL command 193, 333
- CANCEL option
 - ABEND command 323
 - HANDLE ABEND command 370
- CASE operand
 - DFHMDF macro 569
- CBLGENLVL parameter
 - CRTCICSCBL CL command 281

- CBLOPT parameter
 - CRTCICSCBL CL command 275
- CCSID (coded character set identifier) 15
- CEBR transaction 243, 248
 - body 244
 - BOTTOM command 246
 - browse transaction 243
 - CEBR initiation 243
 - COLUMN command 246
 - command area 244
 - displays 244
 - FIND command 246
 - GET command 246
 - header 244
 - initiation 243
 - LINE command 247
 - message line 245
 - PF keys 244
 - PURGE command 247
 - PUT command 247
 - QUEUE command 247
 - security considerations 248
 - temporary storage browse 243
 - TERMINAL command 247
 - TOP command 247
 - transient data 248
- CECI (command-level interpreter) transaction
 - about to start command 252
 - ampersand (&) 256
 - body 253
 - command completed 252
 - command input 250
 - command input line 250
 - command line 250
 - command syntax check 251
 - EIB 257
 - ENTER key 254
 - expanded area 255
 - information area 253
 - introduction 249
 - invoking 249
 - making changes 258
 - message line 254
 - messages display 257
 - PF key values area 254
 - program control 259
 - screen layout 250
 - security considerations 261
 - status area 251
 - terminal sharing 259
 - variables 255
- CECS transaction 249
- CEDF (execution diagnostic facility) 229
- CEDFSTATUS option
 - INQUIRE PROGRAM command 490
 - SET PROGRAM command 515
- chaining of data 171
- character attribute 165
- checkout, program 229
- CICS COBOL restrictions 20
- CICS translator 12
- CICS-value data area (CVDA) 309, 579
- CICSGENLVL parameter
 - CRTCICSC CL command 293
 - CRTCICSCBL CL command 281

CICS MAP parameter
 CRTICSMAP CL command 302
 CICS OPT parameter
 CRTICSC CL command 292
 CRTICSCBL CL command 273
 CICS STATUS option
 INQUIRE SYSTEM command 493
 CL commands
 CRTBNDC 289
 CRTICSC 286
 CRTICSCBL 266
 CRTICSMAP 301
 CRTCMOD 289
 CRTPGM 289
 STRCICSUSR 191
 CLASS option
 SPOOL OPEN OUTPUT
 command 443
 CLEAR key 142, 144, 163
 CLEAR option 163
 HANDLE AID command 371
 client system 176
 closing spool files 223
 CLOSQCSR parameter
 CRTICSC CL command 298
 CRTICSCBL CL command 280
 CNOTCOMPL option
 SEND (SCS) command 432
 COBOL
 addressing CICS data areas 18
 CALL statement 20, 22
 calling programs 22
 calling subprograms 20
 program segments 18
 COBOL variations
 argument values 308
 translated code 14
 COBOL TYPE option
 INQUIRE PROGRAM command 490
 coded character set identifier
 (CCSID) 15
 coding CICS statements in ILE C
 applications 37
 COLOR operand
 DFHMDI macro 569
 DFHMDI macro 562
 DFHMSD macro 556
 COLOR option
 ASSIGN command 328
 color, extended 145
 COLUMN command, CEBR
 transaction 246
 COLUMN operand
 DFHMDI macro 563
 commands
 argument values 307
 command input area 244
 format 305
 reference information 323
 COMMAREA option 201, 203
 ADDRESS command 324
 LINK command 70, 199, 382
 RETURN command 72, 425
 XCTL command 70, 474
 COMMIT parameter
 CRTICSC CL command 296
 CRTICSCBL CL command 272
 commitment control 105, 128
 common programming interface
 communications (CPI
 communications) 186
 common work area (CWA) 71
 compilers supported
 COBOL 14
 ILE C 38
 compiling application programs 16, 40
 conditions, exception 87
 CONFIRM option
 SEND (APPC) command 430
 CONNECT PROCESS command 335
 CONNECTION option
 INQUIRE CONNECTION
 command 482
 INQUIRE CONNECTION NEXT
 command 483
 CONNECTION, INQUIRE
 command 481
 CONNECTION, SET command 509
 CONNSTATUS option
 INQUIRE CONNECTION
 command 482
 constants
 3270 attributes 154
 AID values, DFHAID 548
 attribute values, DFHBMSCA 545
 for 3270 attributes 545
 for examining EIBAID field 162, 548
 for printer format controls 545
 printer control values,
 DFHBMSCA 545
 contention for resources 62
 control region
 shared resources 189
 work flow 189
 work management 189
 conversational programming 61, 80
 CONVERSE (5250 or 3270 logical)
 command 339
 CONVERSE (APPC) command 337
 converse with terminal or LU 550
 CONVID option
 CONNECT PROCESS command 335
 CONVERSE (APPC) command 337
 EXTRACT ATTRIBUTES (APPC)
 command 359
 EXTRACT PROCESS command 360
 FREE (APPC) command 364
 ISSUE ABEND command 374
 ISSUE CONFIRMATION
 command 375
 ISSUE ERROR command 377
 ISSUE PREPARE command 378
 ISSUE SIGNAL (APPC)
 command 379
 RECEIVE (APPC) command 410
 SEND (APPC) command 430
 WAIT CONVID command 461
 COPY option
 SET PROGRAM command 515
 copybooks
 DFHAID 548
 DFHBMSCA 154, 545
 copying symbolic description maps 152
 COUTPUT parameter
 CRTICSC CL command 291
 CPI communications 186
 CR see control region 189
 create logical file (CTRLF) 116
 creating BMS maps 148
 CRTBNDC CL command 289
 CRTICSC CL command 286
 CRTICSCBL CL command 266
 CRTICSMAP CL command 301
 CRTCMOD CL command 289
 CRTPGM CL command 289
 CSMT log 103
 CTLCHAR option
 CONVERSE (5250 or 3270 logical)
 command 339
 SEND (5250 or 3270 logical)
 command 433
 CTRL operand
 DFHMDI macro 563
 DFHMSD macro 557
 CURAUXDS option
 INQUIRE TRACEDEST
 command 505
 cursor address 551
 CURSOR option
 SEND CONTROL command 435
 SEND MAP command 437
 SEND TEXT command 439
 cursor position
 basic mapping support 159
 terminal control 551
 cursor select key
 handling in program 163
 CVDA (CICS-value data area) 579
 CVDA options
 MAXLIFETIME
 DEQ 351
 ENQ 356
 STATE
 CONNECT PROCESS 336
 EXTRACT ATTRIBUTES
 (APPC) 359
 FREE (APPC) 364
 ISSUE ABEND command 374
 ISSUE CONFIRMATION 375
 ISSUE ERROR 377
 ISSUE PREPARE 378
 ISSUE SIGNAL (APPC) 379
 CVDA values
 ALLOCATED
 ISSUE PREPARE 378
 APPC commands 311
 CONF FREE
 ISSUE PREPARE 378
 CONF RECEIVE
 ISSUE PREPARE 378
 CONF SEND
 ISSUE PREPARE 378
 FREE
 ISSUE PREPARE 378
 LUW
 DEQ 351
 ENQ 356
 PEND FREE
 ISSUE PREPARE 378

- CVDA values (*continued*)
 - PENDRECEIVE
 - ISSUE PREPARE 378
 - RECEIVE
 - ISSUE PREPARE 378
 - ROLLBACK
 - ISSUE PREPARE 378
 - SEND
 - ISSUE PREPARE 378
 - SYNCFREE
 - ISSUE PREPARE 378
 - SYNCRECEIVE
 - ISSUE PREPARE 378
 - SYNCSEND
 - ISSUE PREPARE 378
 - TASK
 - DEQ 351
 - ENQ 356
 - CWA option
 - ADDRESS command 324
 - CWALENG option
 - ASSIGN command 328
- D**
- data
 - chaining 171
 - deleting
 - file control records 344
 - temporary storage queues 349
 - transient data queues 348
 - passing to new tasks 448
 - passing to other programs 201
 - reading from a display 160
 - storing within transaction 68
 - data communication 133
 - data definition services (DDS) 117
 - DATA option
 - FREEMAIN command 365
 - data storing
 - within transaction 68
 - data streams
 - BMS 77
 - compressing 79
 - inbound 78
 - data-area arguments 307
 - data-entry operations 79
 - data-value arguments 307
 - DATALENGTH option
 - LINK command 382
 - DATAONLY
 - sending changed fields 78
 - DATAONLY option
 - modifying displays 156
 - SEND MAP command 437
 - DATASET option 311
 - date field of EIB 211
 - DATE option
 - FORMATTIME command 362
 - DATEFORM option
 - FORMATTIME command 362
 - DATESEP option
 - FORMATTIME command 362
 - DATFMT parameter
 - CRTCICSC CL command 299
 - CRTCICSCBL CL command 284
 - DATSEP parameter
 - CRTCICSC CL command 300
 - CRTCICSCBL CL command 285
 - DAYCOUNT option
 - FORMATTIME command 362
 - DAYOFMONTH option
 - FORMATTIME command 362
 - DAYOFWEEK option
 - FORMATTIME command 362
 - DBCS 166, 174
 - CCSID 15
 - GRAPHIC option of CRTCICSC command 292
 - GRAPHIC option of CRTCICSCBL command 274
 - DBGVIEW parameter
 - CRTCICSC CL command 291
 - DDMMYY option
 - FORMATTIME command 362
 - DDMMYYYY option
 - FORMATTIME command 363
 - deadlock prevention 65, 128
 - debugging 229
 - default action for conditions 87
 - default screen size 147
 - DEFAULT value
 - COLOR operand, DFHMDF macro 569
 - COLOR operand, DFHMDI macro 563
 - COLOR operand, DFHMSD macro 557
 - defining map sets 147
 - definite response protocol
 - terminal control 172
 - DEFRESP option
 - CONVERSE (5250 or 3270 logical) command 339
 - SEND (5250 or 3270 logical) command 434
 - SEND (SCS) command 432
 - DELAY command 193, 341
 - delay processing, task 341
 - DELETE command 123, 344
 - delete loaded program 417
 - DELETE option
 - INQUIRE FILE command 485
 - SET FILE command 512
 - SPOOLCLOSE command 441
 - DELETEQ TD command 348
 - DELETEQ TS command 349
 - deleting records 123
 - DEQ command 350
 - dequeue from resource 350
 - dequeueing resources 197
 - design considerations for applications
 - conversational 61
 - exclusive control of resources 64
 - nonconversational 61
 - pseudoconversational 61
 - destinations
 - extrapartition 215
 - indirect 216
 - intrapartition 215
 - device control options, BMS 158
 - DEVICE option
 - INQUIRE TERMINAL command 501
 - device-dependent data stream 135
 - DFHAID 162, 548, 550
 - DFHBMSCA 154, 545
 - DFHMDF, field definition macro 148, 553, 567
 - DFHMDI, map definition macro 147, 553, 562
 - DFHMSD, map set definition macro 147, 553, 555
 - DFHRESP built-in function 87, 313
 - DFHVALUE built-in function 310
 - DFTRDBCOL parameter
 - CRTCICSC CL command 300
 - CRTCICSCBL CL command 286
 - diagnostic services commands 319
 - DISABLED condition
 - DELETE command 346
 - DELETEQ TD command 349
 - READ command 392
 - READQ TD command 405
 - STARTBR command 455
 - UNLOCK command 459
 - WRITE command 465
 - WRITEQ TD command 470
 - DISCARD commands 211, 318
 - AUTINSTMODEL 477
 - FILE 478
 - PROGRAM 478
 - TRANSACTION 479
 - discarding resources
 - resource definitions 477
 - display device operations
 - attention identifier (AID) 551
 - attention identifier list, DFHAID 548
 - cursor address 551
 - input operation without data 551
 - standard attribute and printer control character 545
 - standard attributes (DFHBMSCA) 545
 - terminal 550
 - display screens 72
 - display-device operations
 - pass control on receipt of an AID 371, 374
 - display, reading from 160
 - distributed program link (DPL) 4, 176, 199
 - client system 176, 381
 - COMMAREA option 179
 - DPL API subset 183
 - exception conditions 184
 - independent syncpoints 180
 - options 178
 - programming considerations 182
 - RMTPGMID parameter 178
 - server program 178
 - server program restrictions 381
 - server system 176, 179, 381
 - SYSID option 179
 - TRANSID option 179
 - distributed transaction processing (DTP) 4, 169, 175
 - DL/I (IMS) data base access 4
 - DLYPRP parameter
 - CRTCICSC CL command 298
 - CRTCICSCBL CL command 280

- DRK value
 - ATTRB operand, DFHMDF macro 569
- DSATTS operand
 - DFHMDF macro 563
 - DFHMSD macro 557
- DSNAME option
 - INQUIRE FILE command 485
- DTIMEOUT option
 - INQUIRE TRANSACTION command 506
- DUMP option
 - PERFORM SHUTDOWN command 509
- DUMP TRANSACTION command 352
- DUMPCODE option
 - DUMP TRANSACTION command 352
- DUMPING option
 - INQUIRE SYSTEM command 493
 - INQUIRE TRANSACTION command 506
 - SET SYSTEM command 517
 - SET TRANSACTION command 524
- DUPKEY condition
 - DELETE command 346
 - READ command 392
 - READNEXT command 398
 - READPREV command 402
- DUPREC condition
 - REWRITE command 428
 - WRITE command 465

E

- ECADDR option
 - WAIT EVENT command 462
- EDF
 - abend user task 235
 - browse temporary storage 235
 - CEDF transaction 229
 - displays 231
 - dual-screen mode 240
 - functions 230
 - invoking 229
 - modifying execution 240
 - options on function (PF) keys 235
 - overtyping displays 240
 - PF key 231
 - program labels 241
 - pseudoconversational programs 239
 - single-screen mode 238
- EDF (execution diagnostic facility) 229
- EIB option
 - ADDRESS command 324
- EIBAID field
 - contents of 529
 - copybook DFHAID 162, 548
 - examining contents of 548
- EIBATT field 529
- EIBCALEN field 529
- EIBCOMPL field 529
- EIBCONF field 529
- EIBCPOSN field 529
- EIBDATE field 530
- EIBDS field 530
- EIBEOC field 530

- EIBERR field 530
- EIBERRCD field 530
- EIBFMH field 530
- EIBFN field 530
- EIBFREE field 534
- EIBNODAT field 534
- EIBRCODE field 535
- EIBRECV field 539
- EIBREQID field 540
- EIBRESP field 540
- EIBRESP2 field 542
- EIBRLDBK field 542
- EIBRSRCE field 542
- EIBSIG field 542
- EIBSYNC field 542
- EIBSYNRB field 543
- EIBTASKN field 543
- EIBTIME field 543
- EIBTRMID field 543
- EIBTRNID field 543
- EMPTYSTATUS option
 - INQUIRE FILE command 485
 - INQUIRE TDQUEUE command 497
 - SET FILE command 512
- ENABLESTATUS option
 - INQUIRE FILE command 485
 - INQUIRE TDQUEUE command 497
 - SET FILE command 512
 - SET TDQUEUE command 519
- END condition
 - INQUIRE AUTINSTMODEL NEXT command 481
 - INQUIRE CONNECTION NEXT command 483
 - INQUIRE FILE NEXT command 488
 - INQUIRE JOURNALNUM command 489
 - INQUIRE PROGRAM NEXT command 493
 - INQUIRE TDQUEUE NEXT command 499
 - INQUIRE TERMINAL NEXT command 504
 - INQUIRE TRANSACTION NEXT command 508
- END option
 - INQUIRE AUTINSTMODEL (browse) command 481
 - INQUIRE CONNECTION (browse) command 483
 - INQUIRE FILE (browse) command 487
 - INQUIRE JOURNALNUM (browse) command 489
 - INQUIRE PROGRAM (browse) command 492
 - INQUIRE TDQUEUE (browse) command 499
 - INQUIRE TERMINAL (browse) command 504
 - INQUIRE TRANSACTION (browse) command 508
- ENDBR command 353
- ENDDATA condition
 - RETRIEVE command 423
- ENDFILE condition
 - READNEXT command 398

- ENDFILE condition (*continued*)
 - READPREV command 402
- ENQ command 355
- ENQBUSY condition
 - ENQ command 356
- enqueueing resources 197
- ENTER option 163
 - HANDLE AID command 371
- ENTER TRACENUM command 357
- entry point, trace 112
- entry-sequenced file (ESDS) 116
- ENVDEFERR condition
 - RETRIEVE command 423
- environment services commands 319
- EOC condition
 - CONVERSE (5250 or 3270 logical) command 341
 - CONVERSE (APPC) command 338
 - RECEIVE (5250 or 3270 logical) command 414
 - RECEIVE (APPC) command 411
- EQUAL option
 - READ command 390
 - RESETBR command 419
 - STARTBR command 454
- ERASE option
 - CONVERSE (5250 or 3270 logical) command 340
 - SEND (5250 or 3270 logical) command 434
 - SEND CONTROL command 435
 - SEND MAP command 437
 - SEND TEXT command 440
- ERASEAUP option
 - SEND CONTROL command 435
 - SEND MAP command 437
- ERROR condition 87
- ESDS (entry-sequenced file) 116
- event, timer 193
 - control area 387
- events, timer
 - waiting for 461
- examples
 - browsing the PPT 317
 - CICS command format 305
 - CRTICISC command 39
 - CRTICISCBL CL command 15
 - EXEC CICS READ command 305
 - getting map set storage (COBOL) 20
 - passing CDVA value (short form) 310
 - passing CVDA value (flexible form) 310
 - RECEIVE MAP command 160
 - SEND MAP command 158
 - testing CVDA values 310
 - testing RESP values 313
 - translated code (COBOL) 14
 - translated code (ILE C) 40
 - translator source command 14, 39
 - using DFHVALUE(IGNORE) 318
 - using null values 318
 - using pointer variables (COBOL) 19
 - using pointer variables (ILE C) 42
 - using the BIF DEEDIT command 333
 - using the CANCEL command 323
 - using the DELAY command 343, 344

- examples (*continued*)
 - using the DELETE command 348
 - using the DEQ command 352
 - using the DUMP TRANSACTION command 353
 - using the ENQ command 357
 - using the ENTER TRACENUM command 358
 - using the FORMATTIME command 364
 - using the FREEMAIN command 366
 - using the GETMAIN command 369
 - using the HANDLE ABEND command 370
 - using the HANDLE AID command 372
 - using the HANDLE CONDITION command 373
 - using the LINK command 385
 - using the POST command 389
 - using the READ command 394
 - using the READQ TD command 407
 - using the READQ TS command 409
 - using the RELEASE command 418
 - using the RETRIEVE command 424
 - using the REWRITE command 429
 - using the START command 447
 - using the STORAGE=AUTO operand 44
 - using the WAIT EVENT command 462
 - using the WRITE command 467
 - using the WRITE JOURNALNUM command 469
 - using the WRITEQ TD command 471
 - using the WRITEQ TS command 474
 - using the XCTL command 476
- exception conditions
 - ALLOCATE command 326
 - basic mapping support 162
 - CANCEL command 334
 - CONVERSE (5250 or 3270 logical) command 341
 - CONVERSE (APPC) command 338
 - DELAY command 343
 - DELETE command 346
 - DELETEQ TD command 349
 - DELETEQ TS command 350
 - description 87
 - DISCARD AUTINSTMODEL command 477
 - DISCARD FILE command 478
 - DISCARD PROGRAM command 479
 - DISCARD TRANSACTION command 479
 - ENDBR command 354
 - ENTER TRACENUM command 358
 - FREEMAIN command 365
 - GETMAIN command 368
 - HANDLE ABEND command 370
 - HANDLE CONDITION command 92, 93
 - IGNORE CONDITION command 92
 - INQUIRE AUTINSTMODEL command 480

- exception conditions (*continued*)
 - INQUIRE CONNECTION command 482
 - INQUIRE FILE command 487
 - INQUIRE JOURNALNUM command 488
 - INQUIRE PROGRAM command 492
 - INQUIRE TASK command 495
 - INQUIRE TDQUEUE command 498
 - INQUIRE TERMINAL command 503
 - INQUIRE TRANSACTION command 507
 - LINK command 383
 - LOAD command 386
 - PERFORM SHUTDOWN command 509
 - POP HANDLE command 387
 - POST command 388
 - READ command 392
 - READNEXT command 398
 - READPREV command 402
 - READQ TD command 405
 - READQ TS command 408
 - RECEIVE (5250 or 3270 logical) command 414
 - RECEIVE (APPC) command 411
 - RECEIVE MAP command 416
 - RESETBR command 420
 - RETURN command 426
 - REWRITE command 428
 - SEND (5250 or 3270 logical) command 434
 - SEND (APPC) command 431
 - SEND (SCS) command 433
 - SEND CONTROL command 436
 - SEND MAP command 439
 - SEND TEXT command 440
 - SET CONNECTION command 510
 - SET FILE command 513
 - SET JOURNALNUM command 514
 - SET PROGRAM command 516
 - SET SYSTEM command 517
 - SET TASK command 518
 - SET TDQUEUE command 519
 - SET TERMINAL command 521
 - SET TRACEDEST command 523
 - SET TRANSACTION command 525
 - SPOOLCLOSE command 441
 - SPOOLOPEN OUTPUT command 443
 - SPOOLWRITE command 444
 - STARTBR command 455
 - UNLOCK command 459
 - WAIT CONVID command 461
 - WRITE command 465
 - WRITE JOURNALNUM command 468
 - WRITEQ TD command 470
 - WRITEQ TS command 473
 - XCTL command 475
- exception support commands 320
- exclusive control release, UNLOCK command 458
- exclusive resources 64
- EXEC CICS command format 305

- EXEC CICS commands
 - reference information 323
- EXEC interface block (EIB)
 - description 211
 - fields 529
- EXEC interface program 11, 36
- execution diagnostic facility (CEDF) 229
- EXECUTIONSET option
 - INQUIRE PROGRAM command 490
 - SET PROGRAM command 515
- exit, abnormal termination recovery 369
- expiration time
 - notification when reached 387
 - specifying 194
- EXPIRED condition
 - DELAY command 343
 - POST command 388
- EXTATT operand
 - DFHMDI macro 563
 - DFHMSD macro 557
- EXTDS option
 - ASSIGN command 328
- extended attributes 165
- extended color 145
- EXTRACT ATTRIBUTES (APPC) command 359
- EXTRACT PROCESS command 360
- extrapartition transient data 76, 215

F

- FACILITY option
 - ASSIGN command 328
 - INQUIRE TASK command 494
- FACILITYTYPE option
 - INQUIRE TASK command 494
- FCI option
 - ASSIGN command 328
- field concepts, 3270 143
- field data format, BMS 153
- field definition macro, BMS 148, 553, 567
- FIELD option
 - BIF DEEDIT command 333
- field outlining 146
- FIELDS operand
 - DFHMDI macro 564
- file control
 - an overview 115
 - browse operation, ending 353
 - browse operation, starting 452
 - commands 320
 - deleting records 344
 - emulated VSAM files 127
 - end browse operation 353
 - exception conditions 398
 - read next record 395
 - read previous record 400
 - reading records 390
 - release exclusive control 458
 - reset start for browse 418
 - specify start for browse 452
 - specify starting point 452
 - start browse operation 452
 - table (FCT) 117
 - update a record 427
 - writing new record 463

FILE option
 DELETE command 345
 DISCARD command 478
 ENDBR command 353
 INQUIRE FILE command 485
 INQUIRE FILE NEXT command 487
 READ command 390
 READNEXT command 396
 READPREV command 401
 RESETBR command 419
 REWRITE command 427
 SET FILE 511
 STARTBR command 454
 UNLOCK command 459
 WRITE command 463

FILE, INQUIRE command 484
 FILE, SET command 511

FILENOTFOUND condition
 DELETE command 346
 ENDBR command 354
 READ command 392
 READNEXT command 398
 READPREV command 403
 RESETBR command 420
 REWRITE command 428
 STARTBR command 455
 UNLOCK command 459
 WRITE command 465

files
 input to translator 14, 39

FIND command, CEBR transaction 246

fixed-length records, defining 117

FLAG parameter
 CRTICSCBL CL command 282

FLAGSTD parameter
 CRTICSC CL command 299
 CRTICSCBL CL command 282

FLENGTH option
 GETMAIN command 367
 RECEIVE (5250 or 3270 logical) command 413
 RECEIVE (APPC) command 410
 SEND (5250 or 3270 logical) command 434
 SEND (APPC) command 430
 SEND (SCS) command 432
 SPOOLWRITE command 444

FOLD operand
 DFHMSD macro 557

FOR option
 DELAY command 342

form feed control, BMS 168

format of EXEC CICS commands 305

FORMATTIME command 193, 361

FORMFEED option
 SEND CONTROL command 436
 SEND MAP command 437
 SEND TEXT command 440

FREE (APPC) command 364

free main storage 365

FREEKB option
 SEND CONTROL command 436
 SEND MAP command 437
 SEND TEXT command 440

FREEKB value
 CTRL operand, DFHMDI macro 563
 CTRL operand, DFHMSD macro 557

FREEMAIN command 365

FROM option 125
 CONVERSE (5250 or 3270 logical) command 340
 CONVERSE (APPC) command 337
 ENTER TRACENUM command 358
 RECEIVE MAP command 415
 REWRITE command 427
 SEND (5250 or 3270 logical) command 434
 SEND (APPC) command 430
 SEND (SCS) command 432
 SEND MAP command 437
 SEND TEXT command 440
 SPOOLWRITE command 444
 START command 449
 WRITE command 464
 WRITE JOURNALNUM command 468
 WRITEQ TD command 469
 WRITEQ TS command 472

FROMFLENGTH option
 CONVERSE (5250 or 3270 logical) command 340
 CONVERSE (APPC) command 337

FROMLENGTH option
 CONVERSE (5250 or 3270 logical) command 340
 CONVERSE (APPC) command 337
 ENTER TRACENUM command 358

FRSET option
 SEND CONTROL command 436
 SEND MAP command 438

FRSET value
 CTRL operand, DFHMDI macro 563
 CTRL operand, DFHMSD macro 557

FSET value
 ATTRB operand, DFHMDF macro 569

fullword length options
 FLENGTH 549
 FROMFLENGTH 549
 MAXFLENGTH 549
 TOFLENGTH 549

function (PF) keys, CEBR transaction 245

function shipping 4, 176

G

GCHARS option
 INQUIRE TERMINAL command 501

GCODES option
 INQUIRE TERMINAL command 501

generic key 118

GENERIC option
 DELETE command 345
 READ command 390
 RESETBR command 419
 STARTBR command 454

GET command, CEBR transaction 246

GETMAIN command 367

GINIT operand
 DFHMDI macro 569

glossary of terms and abbreviations 589

graphic data fields 166

GRPNAME operand
 DFHMDF macro 570

GTEQ option
 READ command 391
 RESETBR command 419
 STARTBR command 454

H

HANDLE ABEND command 369

HANDLE AID command 163, 371

HANDLE CONDITION command 92, 372

hardware print key 167

hmmss arguments 307

highlighting 145

HIGHLIGHT operand
 DFHMDF macro 571
 DFHMDI macro 564
 DFHMSD macro 558

HILIGHT option
 ASSIGN command 329

HOLD option
 LOAD command 385

HOLDSTATUS option
 INQUIRE PROGRAM command 491

HOME key 142

HOURS option
 DELAY command 342
 START command 449

I

IC attribute 159

IC value
 ATTRB operand, DFHMDF macro 569

IGNORE CONDITION command 92, 373

ILE C compiler 38

ILE C language variations
 argument values 309
 translated code 40

ILLOGIC condition
 DELETE command 346
 ENDBR command 354
 INQUIRE AUTINSTMODEL (browse) command 481
 INQUIRE CONNECTION (browse) command 483
 INQUIRE FILE (browse) command 488
 INQUIRE JOURNALNUM (browse) commands 489
 INQUIRE PROGRAM (browse) command 493
 INQUIRE TDQUEUE (browse) command 499
 INQUIRE TERMINAL (browse) command 504
 INQUIRE TRANSACTION (browse) command 508
 READ command 393
 READNEXT command 398
 READPREV command 403
 RESETBR command 420

ILLOGIC condition (*continued*)
 REWRITE command 428
 STARTBR command 455
 UNLOCK command 459
 WRITE command 465
 IMMEDIATE option
 PERFORM SHUTDOWN
 command 509
 RETURN command 425
 IMS (DL/I) data base access 4
 inbound data streams 78
 INCFILE parameter
 CRTCICSC CL command 296
 CRTCICSCBL CL command 278
 incorrect output 102
 index, alternate 116
 indirect destinations 216
 INDIRECTNAME option
 INQUIRE TDQUEUE command 497
 INITIAL operand
 DFHMDF macro 571
 initialize main storage 367
 initialize working storage 7
 initiate a task 448
 INITIMG option
 GETMAIN command 368
 input data
 chaining of 171
 input files to translator 14, 39
 input operation without data 551
 input operations 141
 INPUTMSG option 203, 205
 LINK command 382
 RETURN command 425
 XCTL command 475
 INPUTMSGLEN option
 LINK command 382
 RETURN command 426
 XCTL command 475
 INQUIRE commands 211, 315
 AUTINSTMODEL 480
 CONNECTION 481
 FILE 484
 JOURNALNUM 488
 NETNAME 500
 PROGRAM 490
 SYSTEM 493
 TASK 494
 TDQUEUE 496
 TERMINAL 500
 TRACEDEST 505
 TRANSACTION 506
 insert-cursor indicator 146
 interactive debugging
 CECI (command-level
 interpreter) 249
 CECS transaction 249
 CEDF transaction 229
 EDF (execution diagnostic
 facility) 229
 intercommunication 175
 interval control 193
 CANCEL command 193
 cancel interval control command 333
 commands 321
 DELAY command 194
 delay processing of a task 193, 341
 interval control (*continued*)
 expiration time 194
 FORMATTIME options 362
 notification when specified time
 expires 387
 POST command 194
 retrieve data stored for task 421
 specifying request identifier 196
 start a task 193, 445
 START command 194
 task states 194
 timer-related tasks 194
 wait for event to occur 461
 INTERVAL option
 DELAY command 342
 POST command 388
 START command 449
 INTO option 125
 CONVERSE (5250 or 3270 logical)
 command 340
 CONVERSE (APPC) command 337
 READ command 391
 READNEXT command 396
 READPREV command 401
 READQ TD command 405
 READQ TS command 407
 RECEIVE (5250 or 3270 logical)
 command 413
 RECEIVE (APPC) command 410
 RECEIVE MAP command 415
 RETRIEVE command 422
 intrapartition transient data 74, 215
 INTSTATUS option
 INQUIRE TRACEDEST
 command 505
 SET TRACEDEST command 523
 INVITE option
 SEND (5250 or 3270 logical)
 command 434
 SEND (APPC) command 430
 SEND (SCS) command 432
 SEND/RECEIVE protocol 171
 INVMPZ condition
 RECEIVE MAP command 416
 SEND MAP command 439
 invoking CICS services 12, 37
 invoking EDF 229
 INVREQ condition
 ADDRESS command 325
 ALLOCATE command 326
 ASSIGN command 332
 CANCEL command 334
 CONNECT PROCESS command 336
 CONVERSE (5250 or 3270 logical)
 command 341
 CONVERSE (APPC) command 338
 DELAY command 343
 DELETE command 346
 DELETEQ TD command 349
 DELETEQ TS command 350
 DEQ command 351
 DUMP TRANSACTION
 command 352
 ENDBR command 354
 ENQ command 357
 ENTER TRACENUM command 358
 INVREQ condition (*continued*)
 EXTRACT ATTRIBUTES (APPC)
 command 359
 EXTRACT PROCESS command 360
 FORMATTIME command 363
 FREE (APPC) command 365
 FREEMAIN command 365
 HANDLE AID command 372
 ISSUE ABEND command 374
 ISSUE CONFIRMATION
 command 375
 ISSUE ERASEAUP command 376
 ISSUE ERROR command 377
 ISSUE PREPARE command 379
 ISSUE SIGNAL (APPC)
 command 380
 LINK command 383
 POP HANDLE command 387
 POST command 388
 READ command 393
 READNEXT command 398
 READPREV command 403
 READQ TD command 405
 READQ TS command 408
 RECEIVE (5250 or 3270 logical)
 command 414
 RECEIVE (APPC) command 411
 RECEIVE MAP command 416
 RELEASE command 417
 RESETBR command 420
 RETRIEVE command 423
 RETURN command 426
 REWRITE command 428
 SEND (5250 or 3270 logical)
 command 434
 SEND (APPC) command 431
 SEND (SCS) command 433
 SEND CONTROL command 436
 SEND MAP command 439
 SEND TEXT command 440
 SPOOLCLOSE command 441
 START command 451
 STARTBR command 455
 SYNCPOINT command 457
 SYNCPOINT ROLLBACK 458
 WAIT CONVID command 461
 WAIT EVENT command 462
 WRITE command 465
 WRITEQ TD command 470
 WRITEQ TS command 473
 XCTL command 475
 IOERR condition
 DELETE command 347
 READ command 393
 READNEXT command 399
 READPREV command 403
 READQ TD command 406
 READQ TS command 408
 RESETBR command 421
 RETRIEVE command 423
 REWRITE command 428
 START command 452
 STARTBR command 456
 UNLOCK command 460
 WRITE command 465
 WRITE JOURNALNUM
 command 468

IOERR condition (*continued*)
 WRITEQ TD command 470
 WRITEQ TS command 473
 IOTYPE option
 INQUIRE TDQUEUE command 497
 ISCVREQ condition
 CANCEL command 334
 DELETE command 347
 DELETEQ TD command 349
 DELETEQ TS command 350
 ENDBR command 354
 READ command 393
 READNEXT command 399
 READPREV command 403
 READQ TD command 406
 READQ TS command 408
 RESETBR command 421
 REWRITE command 428
 START command 452
 STARTBR command 456
 UNLOCK command 460
 WRITE command 466
 WRITEQ TD command 470
 WRITEQ TS command 473
 ISSUE ABEND command 374
 ISSUE CONFIRMATION command 375
 ISSUE ERASEAUP command 376
 ISSUE ERROR command 377
 ISSUE PREPARE command 378
 ISSUE SIGNAL (APPC) command 379
 ITEM option
 READQ TS command 407
 WRITEQ TS command 472
 ITEMERR condition
 READQ TS command 409
 WRITEQ TS command 473

J

JIDERR condition
 WRITE JOURNALNUM
 command 468
 journal
 creating record 467
 receiver 105
 record layout 107
 records 76
 records (user) 106
 journal control
 output synchronization 107
 journal files
 defining 106
 nonswitchable 107
 physical files 107
 switchable 107
 journaling
 commands 321
 JOURNALNUM option
 INQUIRE FILE command 485
 INQUIRE JOURNALNUM NEXT
 command 489
 WAIT JOURNALNUM
 command 463
 WRITE JOURNALNUM
 command 468
 JOURNALNUM, INQUIRE
 command 488

JOURNALNUM, SET command 514
 JTYPE option
 INQUIRE JOURNALNUM
 command 488
 JTYPEID option
 WRITE JOURNALNUM
 command 468
 JUSTIFY operand
 DFHMDF macro 572

K

KATAKANA option
 ASSIGN command 329
 KEEP option
 SPOOLCLOSE command 441
 KEXTATT operand
 DFHMDI macro 564
 DFHMSD macro 558
 key
 alternate (secondary) 116
 generic 118
 primary 116
 key-sequenced file (KSDS) 116
 KEYLENGTH option
 DELETE command 345
 INQUIRE FILE command 485
 READ command 391
 READNEXT command 396
 READPREV command 401
 remote file 127
 RESETBR command 419
 STARTBR command 454
 WRITE command 464
 KSDS (key-sequenced file) 116

L

label arguments 307
 LABEL option
 HANDLE ABEND command 370
 LANG operand
 DFHMSD macro 558
 LANGID parameter
 CRTICSCBL CL command 283
 LANGUAGE option
 INQUIRE PROGRAM command 491
 LAST option
 bracket protocol 172
 SEND (5250 or 3270 logical)
 command 434
 SEND (APPC) command 430
 SEND (SCS) command 432
 LEFT value
 JUSTIFY operand, DFHMDF
 macro 572
 OUTLINE operand, DFHMDF
 macro 573
 OUTLINE operand, DFHMDI
 macro 565
 OUTLINE operand, DFHMSD
 macro 559
 LENGERR condition
 CONNECT PROCESS command 336
 CONVERSE (5250 or 3270 logical)
 command 341

LENGERR condition (*continued*)
 CONVERSE (APPC) command 338
 DEQ command 352
 ENQ command 357
 ENTER TRACENUM command 358
 EXTRACT PROCESS command 361
 GETMAIN command 368
 LINK command 384
 READ command 393
 READNEXT command 399
 READPREV command 403
 READQ TD command 406
 READQ TS command 409
 RECEIVE (5250 or 3270 logical)
 command 414
 RECEIVE (APPC) command 411
 RETRIEVE command 424
 RETURN command 426
 REWRITE command 429
 SEND (5250 or 3270 logical)
 command 435
 SEND (APPC) command 431
 SEND (SCS) command 433
 SPOOLWRITE command 444
 START command 452
 WRITE command 466
 WRITE JOURNALNUM
 command 468
 WRITEQ TD command 470
 WRITEQ TS command 473
 XCTL command 475
 LENGTH operand
 DFHMDF macro 572
 LENGTH option
 BIF DEEDIT command 333
 built-in function 333
 DEQ command 351
 ENQ command 355
 GETMAIN command 368
 INQUIRE PROGRAM command 491
 LINK command 383
 READ command 391
 READNEXT command 396
 READPREV command 401
 READQ TD command 405
 READQ TS command 408
 RECEIVE (5250 or 3270 logical)
 command 413
 RECEIVE (APPC) command 410
 RECEIVE MAP command 416
 RETRIEVE command 422
 RETURN command 426
 REWRITE command 427
 SEND (5250 or 3270 logical)
 command 434
 SEND (APPC) command 430
 SEND (SCS) command 432
 SEND MAP command 438
 SEND TEXT command 440
 size of 312
 START command 449
 WRITE command 464
 WRITE JOURNALNUM
 command 468
 WRITEQ TD command 469
 WRITEQ TS command 472
 XCTL command 475

- levels, application program logical 200
- LINE command, CEBR transaction 247
- LINE operand
 - DFHMDI macro 565
- line transmission capacity 66
- line width for printer 168
- LINK command 200, 380
- link to program, expecting return 200, 380
- listing, output from translator 15, 39
- LMAPMBR parameter
 - CRTCICSMAP CL command 303
- LMAPSRC parameter
 - CRTCICSMAP CL command 303
- load a map set 151, 385
- load a table 385
- LOAD command 385
- locks, record 128
- logging files 76
- logical levels, application program 27, 53, 200
- logical unit
 - 3270 SCS Printer 432
 - 3270-display 339
 - conversing with (CONVERSE) 550
 - reading data from terminal control 549
- logical unit of work (LUW)
 - description 62
 - recoverable resources 62
 - syncpoints used 106
- logical units (LUs)
 - facilities for 170
- loops 102
- LUs (logical units)
 - facilities for 170

M

- macro
 - field definition, DFHMDF 553, 567
 - map definition, DFHMDI 553, 562
 - map set definition, DFHMSD 553, 555
- macros, BMS
 - field definition, DFHMDF 148
 - map definition, DFHMDI 147
 - map set definition, DFHMSD 147
- MAIN option
 - WRITEQ TS command 472
- main storage
 - initialize 367
 - obtain 367
 - temporary data 219
- main temporary storage 73
- main temporary storage data 219
- map definition macro, BMS 147, 553, 562
- MAP option
 - RECEIVE MAP command 416
 - SEND MAP command 438
- map set
 - definition macro, BMS 147, 553, 555
 - definition, terminating 148
 - loading 151
 - name 148
 - suffixing 149

- MAPATTS operand
 - DFHMDI macro 565
 - DFHMSD macro 558
- MAPCOLUMN option
 - ASSIGN command 329
- MAPFAIL condition 144
 - RECEIVE MAP command 416
- MAPHEIGHT option
 - ASSIGN command 329
- MAPLINE option
 - ASSIGN command 329
- MAPONLY option
 - modifying displays 156
 - SEND MAP command 438
 - sending constant data 78
- MAPONLY value
 - EXTATT operand, DFHMDI macro 564
 - EXTATT operand, DFHMSD macro 557
 - KEXTATT operand, DFHMDI macro 565
- mapping input data 160
- maps
 - BMS 77, 78
 - copying symbolic description 152
 - creating 148
 - physical 136
 - sets 64
 - symbolic 136
- maps, loading 385
- MAPSET option
 - RECEIVE MAP command 416
 - SEND MAP command 438
- MAPWIDTH option
 - ASSIGN command 329
- MARGINS parameter
 - CRTCICSC CL command 294
- MASSINSERT option
 - WRITE command 464
- MAXLENGTH option
 - CONVERSE (5250 or 3270 logical) command 340
 - CONVERSE (APPC) command 337
 - RECEIVE (5250 or 3270 logical) command 413
 - RECEIVE (APPC) command 410
- MAXLENGTH option
 - CONVERSE (5250 or 3270 logical) command 340
 - CONVERSE (APPC) command 337
 - RECEIVE (5250 or 3270 logical) command 413
 - RECEIVE (APPC) command 410
- MAXLIFETIME option
 - DEQ command 351
 - ENQ command 356
- MAXPROCLEN option
 - EXTRACT PROCESS command 360
- migration considerations 5
- MINUTES option
 - DELAY command 342
 - START command 449
- MMDDYY option
 - FORMATTIME command 363
- MMDDYYYY option
 - FORMATTIME command 363

- MODE operand
 - DFHMSD macro 559
- MODENAME option
 - INQUIRE TERMINAL command 502
- modified data tag (MDT) 77, 142, 145
- modifying execution, EDF 240
- modular program 63
- MONTHOFYEAR option
 - FORMATTIME command 363
- MSGLMT parameter
 - CRTCICSC CL command 291
- multithread testing 103
- MUSTENTER value
 - VALIDN operand, DFHMDF macro 575
 - VALIDN operand, DFHMDI macro 566
 - VALIDN operand, DFHMSD macro 561
- MUSTFILL value
 - VALIDN operand, DFHMDF macro 575
 - VALIDN operand, DFHMDI macro 566
 - VALIDN operand, DFHMSD macro 561

N

- name arguments 307
- naming restriction 13, 38
- NETNAME option
 - ASSIGN command 329
 - INQUIRE CONNECTION command 482
 - INQUIRE TERMINAL command 502
- NETNAME, INQUIRE command 500
- new tasks, passing data to 448
- NEXT option
 - INQUIRE AUTINSTMODEL (browse) command 481
 - INQUIRE CONNECTION (browse) command 483
 - INQUIRE FILE (browse) command 487
 - INQUIRE JOURNALNUM (browse) command 489
 - INQUIRE PROGRAM (browse) command 492
 - INQUIRE TDQUEUE (browse) command 499
 - INQUIRE TERMINAL (browse) command 504
 - INQUIRE TRANSACTION (browse) command 508
 - READQ TS command 408
- NEXTTRANSID option
 - INQUIRE TERMINAL command 502
 - SET TERMINAL command 521
- NLEOM option
 - SEND TEXT command 440
- NO value
 - EXTATT operand, DFHMDI macro 564
 - EXTATT operand, DFHMSD macro 557

NO value (*continued*)
 KEXTATT operand, DFHMDI macro 565

NOCC option
 SPOOLOPEN OUTPUT command 443

NOCHECK option
 START command 449

NODUMP option
 ABEND command 323

NOHANDLE option 90, 312

NOJBUFSP condition
 WRITE JOURNALNUM command 468

nonconversational programming 61

nondisplay fields 144

NOQUEUE option
 ALLOCATE command 325

NORM value
 ATTRB operand, DFHMDF macro 569

normal intensity field 144

NOSPACE condition
 REWRITE command 429
 WRITE command 466
 WRITEQ TD command 470
 WRITEQ TS command 473

NOSTG condition
 GETMAIN command 368

NOSUSPEND option
 ALLOCATE command 325
 ENQ command 356
 GETMAIN command 368
 READQ TD command 405
 WRITE JOURNALNUM command 468
 WRITEQ TS command 472

NOTALLOC condition
 CONNECT PROCESS command 336
 CONVERSE (APPC) command 339
 EXTRACT ATTRIBUTES (APPC) command 359
 EXTRACT PROCESS command 361
 FREE (APPC) command 365
 ISSUE ABEND command 375
 ISSUE CONFIRMATION command 376
 ISSUE ERROR command 377
 ISSUE PREPARE command 379
 ISSUE SIGNAL (APPC) command 380
 RECEIVE (APPC) command 412
 SEND (APPC) command 431
 WAIT CONVID command 461

NOTAUTH condition 87
 CANCEL command 334
 DELETE command 347
 DELETEQ TD command 349
 DELETEQ TS command 350
 ENDBR command 354
 HANDLE ABEND command 370
 LINK command 384
 LOAD command 386
 READ command 394
 READNEXT command 399
 READPREV command 404
 READQ TD command 406

NOTAUTH condition (*continued*)
 READQ TS command 409
 RECEIVE MAP command 417
 RELEASE command 418
 RESETBR command 421
 REWRITE command 429
 SEND MAP command 439
 START command 452
 STARTBR command 456
 UNLOCK command 460
 WRITE command 466
 WRITE JOURNALNUM command 469
 WRITEQ TD command 470
 WRITEQ TS command 473
 XCTL command 476

NOTFND condition
 CANCEL command 334
 DELETE command 347
 READ command 394
 READNEXT command 399
 READPREV command 404
 RESETBR command 421
 RETRIEVE command 424
 SPOOLCLOSE command 441
 SPOOLOPEN OUTPUT command 443
 STARTBR command 456

NOTOPEN condition
 DELETE command 347
 READ command 394
 READQ TD command 406
 SPOOLCLOSE command 441
 SPOOLOPEN OUTPUT command 443
 SPOOLWRITE command 445
 STARTBR command 456
 UNLOCK command 460
 WRITE command 466
 WRITE JOURNALNUM command 469
 WRITEQ TD command 470

NOTRUNCATE option
 CONVERSE (5250 or 3270 logical) command 340
 CONVERSE (APPC) command 338
 RECEIVE (5250 or 3270 logical) command 414
 RECEIVE (APPC) command 411

null lines and printers 168

null values 317
 use of 80

NUM value
 ATTRB operand, DFHMDF macro 568

numeric-only field (3270 attribute character) 144

NUMITEMS option
 INQUIRE TDQUEUE command 497
 READQ TS command 408
 WRITEQ TS command 472

NUMREC option
 DELETE command 345

O

OBJ parameter
 CRTICISC CL command 290

object types 289

OBJTYPE parameter
 CRTICISC CL command 290
 CRTICISCBL CL command 272

OCCURS operand
 DFHMDF macro 573

OFF value
 HIGHLIGHT operand, DFHMDF macro 571
 HIGHLIGHT operand, DFHMDI macro 564
 HIGHLIGHT operand, DFHMDS macro 558

OPCLASS option
 ASSIGN command 329

OPENERR condition
 SPOOLOPEN OUTPUT command 443

OPENSTATUS option
 INQUIRE FILE command 486
 INQUIRE JOURNALNUM command 488
 INQUIRE TDQUEUE command 497
 SET FILE command 512
 SET JOURNALNUM command 514
 SET TDQUEUE command 519

OPID option
 ASSIGN command 329

OPREL option
 INQUIRE SYSTEM command 493

OPSYS option
 INQUIRE SYSTEM command 494

options
 fullword length 549
 on function keys, EDF 235
 translator 14

options, translator 38

OS/400 journal 105

OUTFILE parameter
 CRTICISC CL command 293

OUTLINE operand
 DFHMDF macro 573
 DFHMDI macro 565
 DFHMDS macro 559

OUTLINE option
 ASSIGN command 330

OUTMBR parameter
 CRTICISC CL command 293

output data, chaining of 171

output operations 143

output, incorrect 102

OVER value
 OUTLINE operand, DFHMDF macro 573
 OUTLINE operand, DFHMDI macro 565
 OUTLINE operand, DFHMDS macro 559

overtyping EDF displays 240

P

- PA (program access) key 142, 163
- PA1 option 163
- PA1-PA3 option
 - HANDLE AID command 371
- PA2 option 163
- PA3 option 163
- page width for printer 168
- passing control
 - anticipating return (LINK) 200
 - expecting return (LINK) 380
 - on receipt of an AID (HANDLE AID command) 371
 - on receipt of an AID (IGNORE AID) 374
 - without return (XCTL) 474
- passing data
 - to other programs 201
- passing data to new tasks 448
- path, alternate index 116
- PERFORM command 211, 318
 - SHUTDOWN 509
- PERFORM guidelines 22
- performance considerations 81
- period, use of 13, 38
- PF (program function) key
 - BMS 163
- PF1-24 option
 - HANDLE AID command 371
- PFn option 163
- PFXLENG option
 - WRITE JOURNALNUM command 468
- PGM parameter
 - CRTCICSCBL CL command 271
- PGMIDERR condition
 - HANDLE ABEND command 370
 - LINK command 384
 - LOAD command 386
 - RELEASE command 418
 - XCTL command 476
- physical maps 136
- PICIN operand
 - DFHMDF macro 573
- PICOUT operand
 - DFHMDF macro 574
- PIPLENGTH option
 - CONNECT PROCESS command 335
- PIPLIST option
 - CONNECT PROCESS command 335
- pointer-ref arguments 307
- pointer-value arguments 307
- POP HANDLE command 95, 386
- portability 5, 117
- POS operand
 - DFHMDF macro 554, 574
- POST command 193, 387
- posting timer event control area 387
- PREFIX option
 - WRITE JOURNALNUM command 468
- preprocessing
 - PP option of CRTCICSC command 293
- preprocessor 38
- prevention of deadlocks 65, 128
- PRINSYSID option
 - ASSIGN command 330
- PRINT option
 - SEND CONTROL command 436
 - SEND MAP command 438
 - SEND TEXT command 440
 - SPOOLOPEN OUTPUT command 443
- PRINT value
 - CTRL operand, DFHMDI macro 563
 - CTRL operand, DFHMSD macro 557
- printer control character list, DFHBMSCA 545
- printer spooling 223
 - commands 321
- printers
 - and blank lines 167
 - page width 167
 - printing displayed data 167
 - starting a printer task 167
- printing contents of screen 167
- problems, application programs
 - abends 102
 - incorrect output 102
 - loops 102
 - waits 102
- processor storage guidelines 67
- processor time guidelines 67
- PROLENGTH option
 - CONNECT PROCESS command 335
 - EXTRACT PROCESS command 360
- PROCNAME option
 - CONNECT PROCESS command 335
 - EXTRACT PROCESS command 360
- PROFILE option
 - ALLOCATE command 325
- program access (PA) key 163
- program activation 25
- program control
 - commands 321
 - deleting loaded program 417
 - linking to another program 200, 380
 - load a table or map set 385
 - passing data to another program 201
 - program logical levels 200
 - returning program control 424
 - transfer program control 474
- program design
 - conversational 80
- program function (PF) key
 - BMS 163
- program labels in EDF 241
- PROGRAM option
 - HANDLE ABEND command 370
 - INQUIRE PROGRAM NEXT command 492
 - INQUIRE TRANSACTION command 506
 - LINK command 383
 - LOAD command 385
 - RELEASE command 417
 - XCTL command 475
- program segments 18
- program storage 70
- program testing 229
- program, calling 22
- PROGRAM, DISCARD command 478
- PROGRAM, INQUIRE command 490
- PROGRAM, SET command 515
- programming for a CICS environment 66
- programming implications for CICS 66
- programming techniques
 - general 61, 63
- programs
 - compiling application 16, 40
- PROGTYPE option
 - INQUIRE PROGRAM command 491
- PROT value
 - ATTRB operand, DFHMDF macro 568
- PROTECT option
 - START command 450
- protected fields 144
- PS option
 - ASSIGN command 330
- pseudoconversational programming 61
- PUNCH option
 - SPOOLOPEN OUTPUT command 443
- PURGE command, CEBR transaction 247
- PURGEABILITY option
 - INQUIRE TRANSACTION command 507
 - SET TRANSACTION command 525
- PURGETYPE option
 - SET CONNECTION command 510
 - SET TASK command 517
 - SET TERMINAL command 521
- PUSH HANDLE command 95, 389
- PUT command, CEBR transaction 247

Q

- QBUSY condition
 - READQ TD command 406
- QIDERR condition
 - DELETEQ TD command 349
 - DELETEQ TS command 350
 - READQ TD command 406
 - READQ TS command 409
 - WRITEQ TD command 471
 - WRITEQ TS command 473
- QNAME option
 - ASSIGN command 330
- QUEUE command, CEBR transaction 247
- QUEUE option
 - DELETEQ TD command 348
 - DELETEQ TS command 350
 - READQ TD command 405
 - READQ TS command 408
 - RETRIEVE command 422
 - START command 450
 - WRITEQ TD command 470
 - WRITEQ TS command 472
- queues
 - temporary storage 219
 - transient data 215
- QZERO condition
 - READQ TD command 406

R

- RBA (relative byte address) 116, 128
 - RBA option
 - READ command 391
 - READNEXT command 397
 - READPREV command 402
 - RESETBR command 419
 - STARTBR command 454
 - WRITE command 464
 - reactivate an ABEND exit 369
 - READ command 390
 - READ option
 - INQUIRE FILE command 486
 - SET FILE command 512
 - reading data from a display 160
 - reading records 117
 - browsing, next 395
 - from temporary storage queue 407
 - from terminal or LU 549
 - from transient data queue 404
 - READNEXT command 395
 - READPREV command 400
 - READQ TD command 404
 - READQ TS command 407
 - RECEIVE command
 - 5250 or 3270 logical 412
 - APPC 410
 - input operation without data 551
 - read from terminal or logical unit 549
 - RECEIVE MAP command 415
 - RECEIVE MAP, uses of 160, 162
 - receiving CVDA values 310
 - record backspace 142, 163
 - record locks 128
 - RECORDFORMAT option
 - INQUIRE FILE command 486
 - INQUIRE TDQUEUE command 497
 - RECORDLENGTH option
 - INQUIRE TDQUEUE command 497
 - records
 - adding 123
 - browsing 117
 - data definition services (DDS) 117
 - defining length 117
 - deleting 123
 - reading 117
 - release exclusive control 458
 - updating 122
 - user journal 106
 - writing 122, 123
 - writing new 463
 - RECORDSIZE option
 - INQUIRE FILE command 486
 - recoverable files 105
 - defining 105
 - recoverable resources 62
 - recovery
 - exclusive use of resources 62
 - of resources 64
 - problem avoidance 101
 - syncpoint 106
 - RECOVSTATUS option
 - INQUIRE FILE command 486
 - INQUIRE TDQUEUE command 498
 - regression testing 103
 - relative byte address (RBA) 116, 128
 - relative record file (RRDS) 116
 - relative record number (RRN) 116, 128
 - RELEASE command 417
 - RELEASE option
 - INQUIRE SYSTEM command 494
 - release storage, FREEMAIN command 365
 - release-to-release compatibility 10
 - remote file, KEYLENGTH option 127
 - REMTENAME option
 - INQUIRE FILE command 486
 - INQUIRE PROGRAM command 491
 - INQUIRE TDQUEUE command 498
 - INQUIRE TERMINAL command 502
 - INQUIRE TRANSACTION command 507
 - REMOTESYSTEM option
 - INQUIRE FILE command 486
 - INQUIRE PROGRAM command 491
 - INQUIRE TDQUEUE command 498
 - INQUIRE TERMINAL command 502
 - INQUIRE TRANSACTION command 507
 - REPLACE parameter
 - CRTICISC CL command 300
 - CRTICISCBL CL command 286
 - CRTICISMAP CL command 303
 - REQID option
 - CANCEL command 334
 - DELAY command 342
 - ENDBR command 353
 - POST command 388
 - READNEXT command 397
 - READPREV command 402
 - RESETBR command 420
 - START command 450
 - STARTBR command 454
 - WAIT JOURNALNUM command 463
 - WRITE JOURNALNUM command 468
 - request/response unit (RU) 171
 - RESCOUNT option
 - INQUIRE PROGRAM command 491
 - RESET option
 - HANDLE ABEND command 370
 - reset start for browse 418
 - RESETBR command 418
 - resource definitions, browsing 315
 - RESOURCE option
 - DEQ command 351
 - ENQ command 356
 - ENTER TRACENUM command 358
 - resource scheduling 350
 - resources
 - PERFORM guidelines 22
 - RESP option 87, 313
 - deactivating NOHANDLE 93, 373
 - using DFHRESP 87
 - values 540
 - RESP2 option 87, 313
 - values 542
 - RESTART option
 - ASSIGN command 330
 - RETRIEVE command 193, 421
 - retrieve data stored for task 421
 - RETURN command 203, 424
 - return program control 424
 - REVERSE value
 - HIGHLIGHT operand, DFHMDF macro 571
 - HIGHLIGHT operand, DFHMMDI macro 564
 - HIGHLIGHT operand, DFHMMSD macro 558
 - REWRITE command 427
 - REWRITE option
 - WRITEQ TS command 472
 - RIDFLD option 124
 - DELETE command 345
 - READ command 391
 - READNEXT command 397
 - READPREV command 402
 - RESETBR command 420
 - STARTBR command 454
 - WRITE command 464
 - RIGHT value
 - JUSTIFY operand, DFHMDF macro 572
 - OUTLINE operand, DFHMDF macro 573
 - OUTLINE operand, DFHMMDI macro 565
 - OUTLINE operand, DFHMMSD macro 559
 - RMTPGMID parameter 178
 - ROLLEDBACK condition
 - LINK command 384
 - SYNCPOINT command 457
 - RRDS (relative record file) 116
 - RRN (relative record number) 116, 128
 - RRN option
 - DELETE command 346
 - READ command 392
 - READNEXT command 397
 - READPREV command 402
 - RESETBR command 420
 - STARTBR command 454
 - WRITE command 464
 - RTERMID option
 - RETRIEVE command 423
 - START command 450
 - RTIMEOUT option
 - INQUIRE TRANSACTION command 507
 - RTRANSID option
 - RETRIEVE command 423
 - START command 451
 - RU (request/response unit) 171
 - run unit in ILE C 27
- ## S
- SA (set attribute) order 165
 - SAAFLAG parameter
 - CRTICISC CL command 298
 - CRTICISCBL CL command 282
 - sample application programs
 - COBOL 27
 - ILE C 53
 - SBA characters 144
 - schedule use of resource by task 350, 355

screen layout design
 input operations 141
 output operations 143
 requirements 146
 screen size 147
 screen, printing contents 167
 SCREENHEIGHT option
 INQUIRE TERMINAL command 502
 screens, testing 101
 SCREENWIDTH option
 INQUIRE TERMINAL command 502
 SCRNHHT option
 ASSIGN command 330
 SCRNSIZE option
 INQUIRE TRANSACTION
 command 507
 SCRNRWD option
 ASSIGN command 330
 SCS, SNA character string
 SEND command 432
 SECONDS option
 DELAY command 343
 START command 451
 security 223
 segments, program 18
 SEND (5250 or 3270 logical)
 command 433
 SEND (APPC) command 430
 SEND (SCS) command 432
 SEND command
 write to terminal 550
 SEND CONTROL command 435
 SEND MAP command 436
 use of 155
 SEND TEXT command 164, 439
 SEND/RECEIVE mode 170
 sequential files 76
 sequential retrieval, browsing 390
 server program 178
 server system 176, 179
 services, invoking CICS 12, 37
 SERVSTATUS option
 INQUIRE CONNECTION
 command 482
 INQUIRE TERMINAL command 502
 SET CONNECTION command 510
 SET TERMINAL command 521
 SESSIONTYPE option
 INQUIRE TERMINAL command 502
 set attribute (SA) order 165
 set buffer address (SBA) 144
 SET commands 211, 315
 CONNECTION 509
 FILE 511
 JOURNALNUM 514
 PROGRAM 515
 SYSTEM 517
 TASK 517
 TDQUEUE 518
 TERMINAL 520
 TRACEDEST 522
 TRANSACTION 524
 SET option 125
 CONVERSE (5250 or 3270 logical)
 command 340
 CONVERSE (APPC) command 338
 GETMAIN command 368
 LOAD command 386
 POST command 388
 READ command 392
 READNEXT command 397
 READPREV command 402
 READQ TD command 405
 READQ TS command 408
 RECEIVE (5250 or 3270 logical)
 command 414
 RECEIVE (APPC) command 411
 RECEIVE MAP command 416
 RETRIEVE command 423
 SHARED option
 GETMAIN command 213, 368
 sharing data across transactions 70
 SHUTDOWN, PERFORM command 509
 SIGDATA option
 ASSIGN command 330
 SIGNAL condition
 CONVERSE (APPC) command 339
 ISSUE ERROR command 377
 RECEIVE (APPC) command 412
 SEND (APPC) command 431
 SIGNONSTATUS option
 INQUIRE TERMINAL command 503
 single-screen mode, EDF 238
 single-thread testing 103
 SIZE operand
 DFHMDI macro 565
 skip-sequential processing 121
 SOSI option
 ASSIGN command 330
 creation 146
 SPOLBUSY condition
 SPOOLOPEN OUTPUT
 command 443
 SPOLERR condition
 SPOOLWRITE command 445
 SPOOL commands
 SPOOLCLOSE command 441
 SPOOLOPEN OUTPUT
 command 442
 SPOOLWRITE command 444
 spool files, closing 223
 SPOOLCLOSE command 441
 SPOOLOPEN OUTPUT command 442
 SPOOLWRITE command 444
 SQL
 statements, compiling 13, 38, 270
 SQLGENLVL parameter
 CRTICISC CL command 298
 CRTICISCBL CL command 281
 SQLOPT parameter
 CRTICISC CL command 294
 CRTICISCBL CL command 274
 SRCFILE parameter
 CRTICISC CL command 290
 CRTICISCBL CL command 271
 CRTICISMAP CL command 302
 SRCMBR parameter
 CRTICISC CL command 290
 CRTICISCBL CL command 271
 CRTICISMAP CL command 302
 SRTSEQ parameter
 CRTICISCBL CL command 282
 standard attributes (DFHBMSCA) 545
 START command 193, 445
 START option
 INQUIRE AUTINSTMODEL (browse)
 command 481
 INQUIRE CONNECTION (browse)
 command 483
 INQUIRE FILE (browse)
 command 487
 INQUIRE JOURNALNUM (browse)
 command 489
 INQUIRE PROGRAM (browse)
 command 492
 INQUIRE TDQUEUE (browse)
 command 499
 INQUIRE TERMINAL (browse)
 command 504
 INQUIRE TRANSACTION (browse)
 command 508
 STARTBR command 452
 STARTCODE option
 ASSIGN command 330
 INQUIRE TASK command 495
 STARTIO option
 WAIT JOURNALNUM
 command 463
 WRITE JOURNALNUM
 command 468
 STARTUP option
 INQUIRE SYSTEM command 494
 STATE option
 ALLOCATE command 326
 CONNECT PROCESS command 336
 CONVERSE (APPC) command 338
 EXTRACT ATTRIBUTES (APPC)
 command 359
 FREE (APPC) command 364
 ISSUE ABEND command 374
 ISSUE CONFIRMATION
 command 375
 ISSUE ERROR command 377
 ISSUE PREPARE command 378
 ISSUE SIGNAL (APPC)
 command 379
 RECEIVE (APPC) command 411
 SEND (APPC) command 430
 WAIT CONVID command 461
 STATUS option
 INQUIRE PROGRAM command 491
 INQUIRE TRANSACTION
 command 507
 SET PROGRAM command 516
 SET TRANSACTION command 525
 stopper field 144
 storage area length 327
 storage control 213
 commands 321
 storage environment 63
 STORAGE operand
 DFHMSD macro 559
 storage, allocation of 152
 STRCICSUSR CL command 191
 STRFIELD option
 CONVERSE (5250 or 3270 logical)
 command 340
 SEND (5250 or 3270 logical)
 command 434
 SEND (SCS) command 432

- subprogram, calling 20
- SUFFIX operand 149
 - DFHMSD macro 559
- suffixing map sets 149
- SUSPEND command 457
- SWITCHACTION option
 - SET TRACEDEST command 523
- SWITCHSTATUS option
 - INQUIRE TRACEDEST command 505
 - SET TRACEDEST command 523
- symbolic cursor positioning 146, 159
- symbolic description maps
 - copying 152
 - definition 136
 - field data format 153
- synchronize action
 - journal output 107
- SYNCLEVEL option
 - CONNECT PROCESS command 336
 - EXTRACT PROCESS command 360
- SYNCONRETURN option
 - LINK command 383
- syncpoint 106
 - backing out 458
 - command 321
 - establishing 457
- SYNCPOINT command 457
- SYNCPOINT ROLLBACK command 458
- syncpointing, DPL 180
- syntax notation 306
- SYSBUSY condition
 - ALLOCATE command 326
- SYSID option 179
 - ALLOCATE command 326
 - ASSIGN command 331
 - CANCEL command 334
 - DELETE command 346
 - DELETEQ TD command 349
 - DELETEQ TS command 350
 - ENDBR command 354
 - LINK command 383
 - READ command 392
 - READNEXT command 397
 - READPREV command 402
 - READQ TD command 405
 - READQ TS command 408
 - RESETBR command 420
 - REWRITE command 428
 - START command 451
 - STARTBR command 455
 - UNLOCK command 459
 - WRITE command 464
 - WRITEQ TD command 470
 - WRITEQ TS command 473
- SYSIDERR condition
 - ALLOCATE command 326
 - CANCEL command 335
 - DELETE command 348
 - DELETEQ TD command 349
 - DELETEQ TS command 350
 - ENDBR command 354
 - LINK command 384
 - READ command 394
 - READNEXT command 399
 - READPREV command 404

- SYSIDERR condition (*continued*)
 - READQ TD command 406
 - READQ TS command 409
 - RESETBR command 421
 - REWRITE command 429
 - START command 452
 - STARTBR command 456
 - UNLOCK command 460
 - WRITE command 466
 - WRITEQ TD command 471
 - WRITEQ TS command 474
- system connections 481
- system information, access to 211
- system programming commands
 - AUTINSTMODEL, DISCARD command 477
 - AUTINSTMODEL, INQUIRE 480
 - CONNECTION, INQUIRE 481
 - CONNECTION, SET 509
 - FILE, DISCARD 478
 - FILE, INQUIRE 484
 - FILE, SET 511
 - JOURNALNUM, INQUIRE 488
 - JOURNALNUM, SET 514
 - NETNAME, INQUIRE 500
 - PROGRAM, DISCARD 478
 - PROGRAM, INQUIRE 490
 - PROGRAM, SET 515
 - SHUTDOWN, PERFORM 509
 - SYSTEM, INQUIRE 493
 - SYSTEM, SET 517
 - TASK, INQUIRE 494
 - TASK, SET 517
 - TDQUEUE, INQUIRE 496
 - TDQUEUE, SET 518
 - TERMINAL, INQUIRE 500
 - TERMINAL, SET 520
 - TRACEDEST, INQUIRE 505
 - TRACEDEST, SET 522
 - TRANSACTION, DISCARD 479
 - TRANSACTION, INQUIRE 506
 - TRANSACTION, SET 524
- system resources 66
- system trace entry point 112
- SYSTEM, INQUIRE command 493
- SYSTEM, SET command 517

T

- tables
 - loading 385
- TABLESIZE option
 - INQUIRE TRACEDEST command 506
 - SET TRACEDEST command 523
- task
 - abnormal termination 369
 - delay processing of 341
 - initiation 448
- task control 197
 - commands 321
 - DEQ command 197
 - ENQ command 197
- task identification 170
- TASK, INQUIRE command 494
- TASK, SET command 517

- TASKID option
 - INQUIRE TERMINAL command 503
- TCTUA (terminal control table user area) 71
- TCTUA option
 - ADDRESS command 324
- TCTUALENG option
 - ASSIGN command 331
- TDQUEUE option
 - INQUIRE TDQUEUE NEXT command 499
- TDQUEUE, INQUIRE command 496
- TDQUEUE, SET command 518
- techniques, programming 61, 63
- temporary source file members
 - output from translator 15, 39
- temporary storage 4
 - auxiliary 73, 219
 - browse transaction, CEBR 243
 - commands 321
 - data 219
 - main 73, 219
 - queue 219
 - write data to TS queue 471
- TERM operand
 - DFHMSD 560
- TERMCODE option
 - ASSIGN command 331
- TERMERR condition
 - CONVERSE (5250 or 3270 logical) command 341
 - CONVERSE (APPC) command 339
 - ISSUE ABEND command 375
 - ISSUE CONFIRMATION command 376
 - ISSUE ERASEAUP command 376
 - ISSUE ERROR command 378
 - ISSUE PREPARE command 379
 - ISSUE SIGNAL (APPC) command 380
 - LINK command 385
 - RECEIVE (5250 or 3270 logical) command 414
 - RECEIVE (APPC) command 412
 - SEND (5250 or 3270 logical) command 435
 - SEND (APPC) command 431
 - SEND (SCS) command 433
- TERMINAL option
 - START command 451
- TERMINAL condition
 - START command 452
- TERMINAL command, CEBR transaction 247
- terminal control
 - 3270 field concept 143
 - an overview 169
 - bracket protocol, LAST option 172
 - chaining of input data 171
 - chaining of output data 171
 - commands 322
 - definite response 172
 - facilities for logical units 170
 - handle attention identifier 163
 - map input data 160
 - operations 77
 - print (ISSUE PRINT) 167

TERMINAL option
 INQUIRE TERMINAL NEXT
 command 504
 RECEIVE MAP command 416
 terminal sharing 259
 terminal-oriented task identification 170
 TERMINAL, INQUIRE command 500
 TERMINAL, SET command 520
 TERMMODEL option
 INQUIRE TERMINAL command 503
 testing applications 101
 testing CVDA values 310
 testing RESP 87
 testing, levels of
 multithread 103
 regression 103
 single-thread 103
 text data format 136
 TEXT parameter
 CRTICISC CL command 291
 CRTICISCBL CL command 272
 CRTICISMAP CL command 304
 TGTRLS parameter
 CRTICISC CL command 294
 CRTICISMAP CL command 303
 time of day, request 326
 TIME option
 DELAY command 343
 FORMATTIME command 363
 POST command 388
 START command 451
 timer event control area 387
 TIMESEP option
 FORMATTIME command 363
 TIMFMT parameter
 CRTICISC CL command 299
 CRTICISCBL CL command 284
 TIMSEP parameter
 CRTICISC CL command 300
 CRTICISCBL CL command 285
 TIOAPFX operand
 DFHMSD macro 560
 TIOAPFX, effect of
 in RECEIVE MAP 416
 in SEND MAP 438
 TOFLENGTH option
 CONVERSE (5250 or 3270 logical)
 command 341
 CONVERSE (APPC) command 338
 TOKEN option
 SPOOLCLOSE command 441
 SPOOLOPEN OUTPUT
 command 443
 SPOOLWRITE command 444
 TOLENGTH option
 CONVERSE (5250 or 3270 logical)
 command 341
 CONVERSE (APPC) command 338
 TOP command, CEBR transaction 247
 trace
 description 111
 trace entry point 112
 TRACEDEST, INQUIRE command 505
 TRACEDEST, SET command 522
 TRACENUM option
 ENTER TRACENUM command 358
 transaction identifier
 CEBR 243
 CECI 249
 CEDF 229
 TRANSACTION option
 INQUIRE TASK command 495
 INQUIRE TRANSACTION NEXT
 command 508
 transaction routing 4, 175
 transaction work area (TWA) 69
 TRANSACTION, DISCARD
 command 479
 TRANSACTION, INQUIRE
 command 506
 TRANSACTION, SET command 524
 transactions
 conversational 61
 nonconversational 61
 pseudoconversational 61
 transfer program control 474
 TRANSID option 179
 CANCEL command 334
 INQUIRE PROGRAM command 492
 LINK command 383
 RETURN command 426
 START command 451
 TRANSIDERR condition
 START command 452
 transient data 4
 automatic transaction initiation
 (ATI) 216
 delete intrapartition queue 348
 extrapartition 76, 215
 indirect 216
 intrapartition 74, 215
 queues 74
 read data from TD queue 404
 write data to TD queue 469
 transient data control 215
 commands 322
 translated code
 COBOL 14
 ILE C 40
 translator
 basic processes 12, 38
 input to 14, 38
 options 14
 output from
 listing 15, 39
 temporary source file
 members 15, 39
 source command, example 14, 39
 translator files
 input 14, 39
 translator options 38
 transmission 67
 TRANSP operand
 DFHMDF macro 575
 DFHMDI macro 566
 DFHMSD macro 560
 TRIGGER value
 VALIDN operand, DFHMDF
 macro 575
 VALIDN operand, DFHMDI
 macro 566
 VALIDN operand, DFHMSD
 macro 561
 TRIGGERLEVEL option
 INQUIRE TDQUEUE command 498
 SET TDQUEUE command 519
 TRIGRAF operand
 DFHMSD macro 560
 TTISTATUS option
 INQUIRE TERMINAL command 503
 SET TERMINAL command 521
 TWA (transaction work area) 69
 TWA option
 ADDRESS command 324
 TWALENG option
 ASSIGN command 331
 TWASIZE option
 INQUIRE TRANSACTION
 command 507
 TYPE operand
 DFHMSD macro 561
 TYPE option
 INQUIRE FILE command 486
 INQUIRE TDQUEUE command 498

U

UNATTEND option
 ASSIGN command 331
 UNDER value
 OUTLINE operand, DFHMDF
 macro 573
 OUTLINE operand, DFHMDI
 macro 565
 OUTLINE operand, DFHMSD
 macro 559
 UNDERLINE value
 HILIGHT operand, DFHMDF
 macro 571
 HILIGHT operand, DFHMDI
 macro 564
 HILIGHT operand, DFHMSD
 macro 558
 unformatted data stream 144
 UNLOCK command 458
 UNPROT value
 ATTRB operand, DFHMDF
 macro 568
 unprotected field, 3270 attribute
 character 144
 UNTIL option
 DELAY command 343
 UPDATE option
 INQUIRE FILE command 487
 READ command 392
 SET FILE command 513
 updating records 122, 427
 use of #include 42
 USECOUNT option
 INQUIRE PROGRAM command 492
 user files 74
 user journaling 105, 106
 user storage 69
 user trace entry point 112
 USERAREA option
 INQUIRE TERMINAL command 503
 USERAREALEN option
 INQUIRE TERMINAL command 503
 USERID option
 ASSIGN command 332

USERID option (*continued*)
 INQUIRE TASK command 495
 INQUIRE TERMINAL command 503
 SPOOL OPEN OUTPUT
 command 443
USERNAME option
 INQUIRE TERMINAL command 503
USERSTATUS option
 SET TRACEDEST command 506, 523

V

VALIDATION option
 ASSIGN command 332
VALIDN operand
 DFHMDF macro 575
 DFHMDI macro 566
 DFHMSD macro 561
values of arguments 307
variable-length records, defining 117
variables, CECL/CECS 255
vertical forms control 168
virtual storage environment 61, 63
VSAM, access to mainframe VSAM
 data 4

W

WAIT CONVID command 461
WAIT EVENT command 193, 461
WAIT JOURNALNUM command 462
WAIT option
 RETRIEVE command 423
 SEND (5250 or 3270 logical)
 command 434
 SEND (APPC) command 431
 SEND (SCS) command 432
 SEND command 550
 SEND MAP command 438
 SEND TEXT command 440
 WRITE JOURNALNUM
 command 468
waits
 for event to occur 461
 problems, application programs 102
WRITE command 463
WRITE JOURNALNUM command 467
 create a journal record 106
WRITEQ TD command 469
WRITEQ TS command 471
writing CICS programs
 COBOL 17
 ILE C 41
writing data
 to terminal or logical unit 550
writing records 122, 123
 file control 463

X

XCTL command 201, 474
XINIT operand
 DFHMDF macro 575

Y

YEAR option
 FORMATTIME command 363
YES value
 EXTATT operand, DFHMDI
 macro 564
 EXTATT operand, DFHMSD
 macro 557
 KEXTATT operand, DFHMDI
 macro 565
YYDDD option
 FORMATTIME command 363
YYDDMM option
 FORMATTIME command 363
YYMMDD option
 FORMATTIME command 363
YYYYDDD option
 FORMATTIME command 363
YYYYDDMM option
 FORMATTIME command 363
YYYYMMDD option
 FORMATTIME command 363

Z

ZERO value
 JUSTIFY operand, DFHMDF
 macro 572

Readers' Comments — We'd Like to Hear from You

iSeries
CICS for iSeries Application
Programming Guide
Version 5

Publication No. SC41-5454-02

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	<input type="checkbox"/>				

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	<input type="checkbox"/>				
Complete	<input type="checkbox"/>				
Easy to find	<input type="checkbox"/>				
Easy to understand	<input type="checkbox"/>				
Well organized	<input type="checkbox"/>				
Applicable to your tasks	<input type="checkbox"/>				

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? Yes No

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

Name

Address

Company or Organization

Phone No.



Fold and Tape

Please do not staple

Fold and Tape



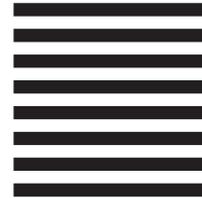
NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM CORPORATION
ATTN DEPT 542 IDCLERK
3605 HWY 52 N
ROCHESTER MN 55901-7829



Fold and Tape

Please do not staple

Fold and Tape



Printed in USA

SC41-5454-02

