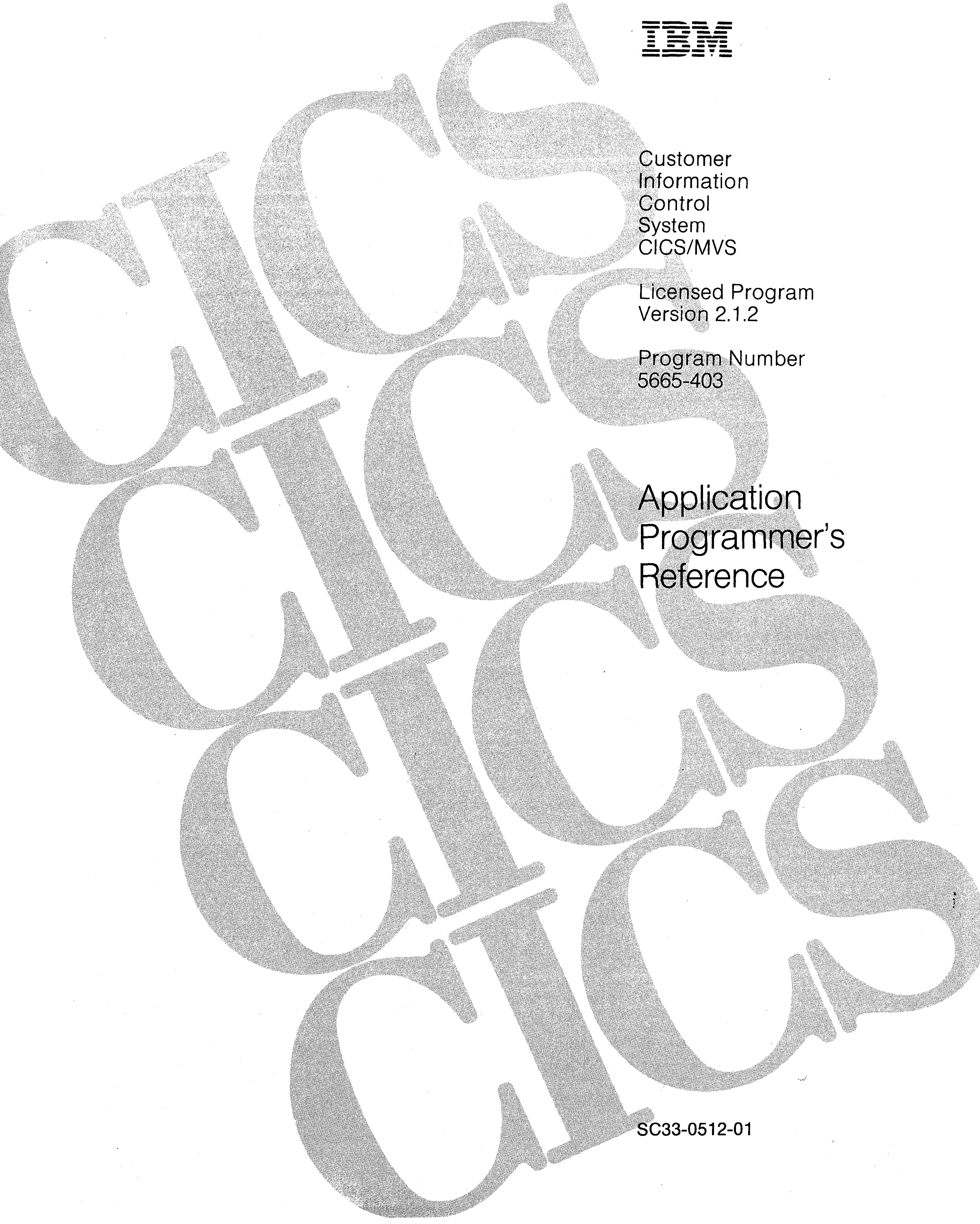


IBM

Customer
Information
Control
System
CICS/MVS

Licensed Program
Version 2.1.2

Program Number
5665-403

Application Programmer's Reference

SC33-0512-01

Second Edition (March 1991)

This edition applies to Version 2 Release 1 Modification 2 of the IBM licensed program Customer Information Control System/Multiple Virtual Storage (CICS/MVS), program number 5665-403, and to all subsequent versions, releases, and modifications until otherwise indicated in new editions. Consult the latest edition of the applicable IBM system bibliography for current information on this product.

This book is based on the *Application Programmer's Reference* for CICS/MVS 2.1, SC33-0512. Changes from that edition are marked by vertical lines to the left of the changes.

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

A form for reader's comments appears at the back of this publication. If the form has been removed, address your comments to:

International Business Machines Corporation, Department 6R1H,
180 Kost Road, Mechanicsburg, PA 17055, U.S.A.

or to:

IBM United Kingdom Laboratories Limited, Information Development,
Mail Point 095, Hursley Park, Winchester, Hampshire, England,
SO21 2JN.

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

Special notices

The following paragraph does not apply to the United Kingdom or any country 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 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.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates.

Any reference to an IBM licensed program or other IBM product in this publication is not intended to state or imply that only IBM's program or other product may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Commercial Relations, IBM Corporation, Purchase, NY 10577.

This book is intended to help you write and prepare programs that use the CICS command-level programming interface. It contains reference information and guidance about using EXEC CICS commands in application programs. This book primarily documents General-Use Programming Interface and Associated Guidance Information provided by CICS.

General-Use programming interfaces allow the customer to write programs that obtain the services of CICS.

However, this book also documents Product-Sensitive Programming Interface and Associated Guidance Information.

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

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

Product-Sensitive Programming Interface

Product-Sensitive Programming Interface and Associated Guidance Information

End of Product-Sensitive Programming Interface

The following terms, denoted by an asterisk (★), used in this publication, are trademarks or service marks of IBM Corporation in the United States or other countries:

CICS/MVS, CICS/ESA, IBM, MVS/XA, VTAM, ACF/VTAM

Preface

What this book is about: This book describes the IBM* Customer Information Control System/Multiple Virtual Storage (CICS/MVS*) command level application programming interface; it contains introductory and reference information necessary to prepare assembler language, COBOL, and PL/I application programs, using CICS commands, to execute under the IBM licensed program CICS/MVS (5665-403).

Note: The INQUIRE and SET commands of the command level application programming interface, together with the spool commands of the CICS interface to JES, are primarily for the use of the system programmer. The commands are fully described in the *CICS/MVS Customization Guide*.

Who should read this book: The book is intended primarily for use by application programmers, but will be useful also for system programmers and systems analysts.

What you need to know to understand this book: Experience in writing programs in assembler language, COBOL, or PL/I is assumed. No previous experience of CICS is assumed.

However, a knowledge of the concepts and terminology introduced in the *CICS/MVS Facilities and Planning Guide* is required. This guide also contains details of system requirements and a glossary applicable to CICS.

How to use this book: This book is mainly for reference. Each of the chapters (other than the introductory chapter) of the parts of the book has a standard format. The first section of a chapter describes, in general terms, functions of the commands included in the chapter. For each command, the following information is presented:

- The syntax of the command and its associated options
- Exceptional conditions that can occur
- A detailed description of what the command does
- And possibly one or more examples showing typical coding of the command.

Finally, two alphabetic lists are given:

- A list of the options, with their functions, that can be used in any of the commands in the chapter
- A list of the exceptional conditions, and their causes, that can occur during execution of the commands.

Notes on terminology

- **VTAM*** refers to ACF/VTAM* and to the record interface of ACF/TCAM
- **ASM** is used sometimes as the abbreviation for assembler language.

* IBM Trademark. For a list of trademarks see page iii.

Book structure

"Part 1. Command level programming" on page 1
Introduces CICS commands and describes the basic facilities that are available to the user. A chapter is included about the command language translator and the options that can be selected to modify the way in which the translator operates.

"Part 2. Files and databases" on page 75
Deals with access to data sets in the user's CICS system either through CICS file control or through DL/I.

"Part 3. Data communication operations" on page 131
Deals with communication with terminals, logical units, and subsystems in the telecommunications network to which the CICS system is connected.

"Part 4. Control operations" on page 271
Describes the facilities for controlling the operation of application programs in the CICS system.

"Part 5. Recovery and debugging" on page 309
Deals with facilities available for recovery from abnormal termination, monitoring, tracing program operation, and dumping areas of main storage.

"Part 6. The CICS built-in function command" on page 333
Describes the one built-in function (BIF DEEDIT) available with the command-level interface.

"Appendixes" on page 337
Contain information on the following topics:

A. EXEC interface block
B. Translation tables for the 2980
C. CICS macros and equivalent commands
D. Sample programs (ASM)
E. Sample programs (COBOL)
F. Sample programs (PL/I).

"Index" on page 473

CICS/MVS 2.1.2 library

General

CICS Library Guide	GC33-0356-04
Master Index	SC33-0513-01
User's Handbook	SX33-6061-01
Messages and Codes	SC33-0514-02

Evaluation and planning

Bróchure	GC33-0503-00
CICS General Information	GC33-0155-01
Facilities and Planning Guide	SC33-0504-01
Release Guide	GC33-0505-03
Data Tables General Information	SC33-0684

Administration

Installation Guide	SC33-0506-01
Customization Guide	SC33-0507-02
Resource Definition (Online)	SC33-0508-01
Resource Definition (Macro)	SC33-0509-02
Operations Guide	SC33-0510-01
CICS-Supplied Transactions	SC33-0511-01

Special topics

Intercommunication Guide	SC33-0519-02
Recovery and Restart Guide	SC33-0520-01
Performance Guide	SC33-0521-01
XRF Guide	SC33-0522-02
CICS Communicating with CICS OS/2	SC33-0736-1
Data Tables Guide	SC33-0632-01

Service

Problem Determination Guide	SC33-0516-01
Diagnosis Handbook	LX33-6062-01
Diagnosis Reference	LY33-6077-00
Data Areas	LY33-6078-00

Programming

CICS Application Programming Primer	SC33-0674-00
Application Programmer's Reference	SC33-0512-01

Version 1 books

CICS/VS Application Programmer's Reference Manual (Macro Level) (SC33-0079)

CICS/OS/VS IBM 3270 Data Stream Device Guide (SC33-0232)

CICS/OS/VS IBM 4700/3600/3630 Guide (SC33-0233)

CICS/OS/VS IBM 3650/3680 Guide (SC33-0234)

CICS/OS/VS IBM 3767/3770/6670 Guide (SC33-0235)

CICS/OS/VS IBM 3790/3730/8100 Guide (SC33-0236)

Books from related libraries

The reader of this book may also want to refer to the following IBM publications:

Operator's Library: OS/VS2 MVS System Commands, GC38-0229

OS/VS Display Consoles, GC38-0255

OS/VS Data Management Macro Instructions, GC26-3793

MVS/ESA Operations: System Commands, GC28-1826-1

Resource Access Control Facility (RACF): Security Administrator's Guide, SC28-1340-5

IMS/VS Messages and Codes Reference Manual, SH20-9030

IMS/VS Utilities Reference Manual, SH20-9029

IMS/VS Data Base Recovery Control Guide and Reference, SH35-0027

IMS/ESA Utilities Reference, SC26-4284

IMS/ESA Messages and Codes, SC26-4290

IMS/ESA System Administration Guide, SC26-4278

IMS/ESA Operations Guide, SC26-4287

CICS/CMS User's Guide, SC33-0285-0

An Introduction to the IBM 3270 Information Display System, GA27-2739.

Contents

Special notices iii

Preface v
Book structure vi
CICS/MVS 2.1.2 library vii

Books from related libraries viii

Summary of changes xiii

Questionnaire xv

Part 1. Command level programming 1

Chapter 1.1. Introduction to command level programming 3
Commands instead of macros 3
| CICS syntax notation used in this book 3

Chapter 1.2. Command format and argument values . 5
Command format 5
Argument values 5

Chapter 1.3. Command language translator 9
Translator data sets 9
Translated code 10
Translator options 13

Chapter 1.4. Programming techniques and restrictions 17
General programming techniques 17
Program size 19
BMS map size 19
The EXEC interface stubs 19
Assembler-language considerations 19
COBOL considerations 20
VS COBOL II considerations 24
VS COBOL II with the ANSI85 COBOL standards . . . 36
PL/I considerations 41

Chapter 1.5. Exceptional conditions 43
Handle exceptional conditions (HANDLE CONDITION) . 45
Ignore exceptional conditions (IGNORE CONDITION) . 45
Suspend condition handling (PUSH and POP) 46
List of exceptional conditions 46

Chapter 1.6. Access to system information 51
Access to CICS storage areas (ADDRESS) 51
Values outside the application program (ASSIGN) . . . 52

Chapter 1.7. Execution (command level) diagnostic facility 57
Functions of EDF 57
Security rules 58
Installing EDF 58
Invoking EDF 58
EDF displays 59
Checking pseudoconversational programs 64
Program labels 64
EDF and EXEC DLI commands 64

Chapter 1.8. Command level interpreter 67
Invoking the command level interpreter 67
Screen layout 67
Program control 72
Security rules 72
Installing the command level interpreter 72

Chapter 1.9. Temporary storage browse 73
Using the transaction 73
CEBR commands 73
Resource definition 74

Part 2. Files and databases 75

Introduction to files and databases 76

Chapter 2.1. General description of file control facilities 77
VSAM data sets 77
BDAM data sets 78
Data set identification 79
Accessing data sets from CICS application programs . 79
Review of file control command options 83
Preventing transaction deadlocks 84
KEYLENGTH option for remote data sets 85

Chapter 2.2. File control – VSAM considerations . . . 87
Record identification 87
CICS locking of VSAM records in recoverable files . . 87

Chapter 2.3. File control – BDAM considerations . . . 89
Record identification 89
Browsing records from BDAM data sets 89
Adding records to BDAM data sets 90
BDAM exclusive control 91

Chapter 2.4. File control – commands, options, and conditions 93
Read a record (READ) 93
Write a record (WRITE) 93
Update a record (REWRITE) 94
Delete a record (DELETE) – VSAM only 94
Release exclusive control (UNLOCK) 95
Start browse (STARTBR) 95
Read next record during a browse (READNEXT) 95

Read previous record during a browse (READPREV) – VSAM only 96
Reset start of browse (RESETBR) 96
End browse (ENDBR) 96
File control options 97
File control exceptional conditions 98

Chapter 2.5. DL/I services (EXEC DLI command) . . 101
EXEC DLI command 101
General format of EXEC DLI command 101
General rules and conventions 104
Example of DL/I requests using EXEC DLI 105

Chapter 2.6. DL/I services (DL/I CALL statement) . . 117
User interface block (UIB) 117
Schedule the PSB and obtain PCB addresses 118
Segment search arguments 118
I/O work area for DL/I segments 119
Issue a DL/I database call 119
Terminate a PSB in the CICS application program . . 119
Check the response to a DL/I CALL 120
Example of DL/I request using call 122

Chapter 2.7. DL/I batch programs (shared DB) . . . 129

Part 3. Data communication operations **131**

Chapter 3.1. Introduction to data communication operations 133

Chapter 3.2-1. Introduction to basic mapping support 135
How BMS affects programming 135
BMS maps 135
BMS map definition 136
Cataloging BMS map sets 136
BMS commands 137
Facilities provided by BMS 137
Sample programs 138

Chapter 3.2-2. Minimum function BMS 139
IBM 3270 information display system 139
Screen layout design 142
Defining BMS maps 143
Writing programs to use BMS services 148
Exceptional conditions 156
Printed output 156

Chapter 3.2-3. Standard function BMS 159
Text processing 159
Printer support 160
Partition support 162
Applications of partitions under CICS 162
How CICS manages partitions 163
Summary of implementation and use of partitions . 165
Application programming 165
Logical device components 169
10/63 magnetic slot reader control 170
Trigger fields 171
Outboard formatting 172
Block data format 172

Chapter 3.2-4. Full function BMS 173
Logical message handling 173
Message routing 184
Message switching transaction (CMSG) 189
Returning mapped data to a program before output 189

Chapter 3.2-5. BMS macro and command reference summary 193
Map set, map, and field definition 193
Partition set definition 204
BMS related constants 205
Input commands 209
Output commands 209
BMS related ASSIGN options 212
BMS options 213
BMS exceptional conditions 217

Chapter 3.3. Terminal control 221
Commands and options for terminals and logical units 222
Commands and options for logical units 225
TCAM-supported terminals and logical units 229
BTAM programmable terminals 229
Teletypewriter programming 230
Display device operations 230
Standard CICS terminal support (BTAM or TCAM) . 234
LUTYPE4 logical unit 235
LUTYPE6.1 logical unit 235
LUTYPE6.2 logical unit (VTAM only) 237
System/3 239
System/370 239
System/7 240
2260 display station 241
2265 display station 241
2741 communication terminal 241
2770 data communication system 242
2780 data transmission terminal 242
2980 general banking terminal system 243
3270 information display system (BTAM or TCAM) . 244
3270 logical unit 245
3270 SCS printer logical unit 245
3270-display logical unit (LUTYPE2) and 3270-printer logical unit (LUTYPE3) 246
3600 finance communication system (BTAM) 246
3600 pipeline logical unit 247
3600 (3601) logical unit 248
3600 (3614) logical unit 248
3630 plant communication system 249
3650/3680 host command processor logical unit . . . 249
3650 host conversational (3270) logical unit 249
3650 host conversational (3653) logical unit 250
3650 interpreter logical unit 250
3650 pipeline logical unit 251

3650/3680 full function logical unit 251
3660 supermarket scanning system 251
3735 programmable buffered terminal 251
3740 data entry system 252
3767 interactive logical unit 252
3770 batch logical unit 253
3770 interactive logical unit 253
3770 full function logical unit 253
3780 communications terminal 253
3790 full function logical unit 254
3790 inquiry logical unit 254
3790 SCS printer logical unit 255
3790 (3270-display) logical unit 255
3790 (3270-printer) logical unit 256
4700 finance communication system 256
7770 audio response unit 256
Terminal control options 257
Terminal control exceptional conditions 262

Chapter 3.4. Batch data interchange 265
Interrogate a data set (ISSUE QUERY) 266
Read a record from a data set (ISSUE RECEIVE) . . 266
Add a record to a data set (ISSUE ADD) 266
Update a record in a data set (ISSUE REPLACE) . . 266
Delete a record from a data set (ISSUE ERASE) . . . 267
End processing of a data set (ISSUE END) 267
End processing of a data set abnormally (ISSUE ABORT) 267
Send data to an output device (ISSUE SEND) 268
Request next record number (ISSUE NOTE) 268
Wait for an operation to be completed (ISSUE WAIT) 268
Batch data interchange options 268
Batch data interchange exceptional conditions . . . 269

Part 4. Control operations 271

Chapter 4.1. Introduction to control operations . . . 273

Chapter 4.2. Interval control 275
Request current date and time of day (ASKTIME) . . 275
Select the format of date and time (FORMATTIME) . 276
Delay processing of a task (DELAY) 276
Request notification when specified time has expired (POST) 276
Wait for an event to occur (WAIT EVENT) 277
Start a task (START) 277
Retrieve data stored for a task (RETRIEVE) 279
Cancel interval control requests (CANCEL) 280
Interval control options 280
Interval control exceptional conditions 283

Chapter 4.3. Task control 285
Suspend a task (SUSPEND) 285
Schedule use of a resource by a task (ENQ and DEQ) 285
Task control options 286
Task control exceptional condition 286
Controlling access sequence 286

Chapter 4.4. Program control 289
Application program logical levels 289
Link to another program anticipating return (LINK) . 289
Transfer program control (XCTL) 290
Return program control (RETURN) 291
Load a program (LOAD) 291
Delete a loaded program (RELEASE) 291
Passing data to other programs 292
Program control options 296
Program control exceptional conditions 297

Chapter 4.5. Storage control 299
Obtain and initialize main storage (GETMAIN) 299
Release main storage (FREEMAIN) 299
Storage control options 300
Storage control exceptional conditions 300

Chapter 4.6. Transient data control 301
Write data to transient data queue (WRITEQ TD) . . 302
Read data from transient data queue (READQ TD) . 302
Delete an intrapartition transient data queue (DELETEQ TD) 303
Transient data control options 303
Transient data control exceptional conditions 303

Chapter 4.7. Temporary storage control 305
Write data to a temporary storage queue (WRITEQ TS) 306
Read data from temporary storage queue (READQ TS) 306
Delete temporary storage queue (DELETEQ TS) . . . 307
Temporary storage control options 307
Temporary storage control exceptional conditions . 308

Part 5. Recovery and debugging 309

Chapter 5.1. Introduction to recovery and debugging 311
Sequential terminal support 311

Chapter 5.2. Abnormal termination recovery 313
Handle an abnormal termination exit (HANDLE ABEND) 313
Terminate task abnormally (ABEND) 314
Abnormal termination recovery options 315
Abnormal termination recovery exceptional condition 315

Chapter 5.3. Trace control 317
Trace entry points 317
Event monitoring points 317
Trace facility control 317

Trace table format 318
CICS auxiliary trace facility 319
User trace entry point and event monitoring point (ENTER) 320
Control the CICS trace facility (TRACE ON, TRACE OFF) 320
Trace control options 321
Trace control exceptional conditions 321

Chapter 5.4. Dump control 323
Dump main storage (DUMP) 323
Dump control options 323
Dump control exceptional conditions 325

Chapter 5.5. Journal control 327
Create a journal record (JOURNAL) 328
Synchronize with journal output (WAIT JOURNAL) . 329
Journal control options 329
Journal control exceptional conditions 329

Chapter 5.6. Recovery (sync points) 331
Establish a sync point (SYNCPOINT) 331
Sync point option 331
Sync point exceptional condition 332

Part 6. The CICS built-in function command 333

Chapter 6.1. The field edit built-in function (BIF DEEDIT) command 335

Appendixes 337

Appendix A. EXEC interface block 339
EIB fields 339

Appendix B. Translation tables for the 2980 347

Appendix C. CICS macros and equivalent commands 351

Appendix D. Sample programs (assembler language) 353
Operator instruction program (ASM) 354
Inquiry/update sample program (ASM) 354
Browse sample program (ASM) 361
Order entry sample program (ASM) 368
Order entry queue print sample program (ASM) . . . 372
Low balance report sample program (ASM) 375
Maps and screen layouts for ASM sample programs 378
Record descriptions for ASM sample programs . . . 392

Appendix E. Sample programs (COBOL) 393
Operator instruction program (COBOL) 394
Inquiry/update sample program (COBOL) 395
Browse sample program (COBOL) 401
Order entry sample program (COBOL) 407
Order entry queue print sample program (COBOL) . 411
Low balance report sample program (COBOL) 414
Maps and screen layouts for COBOL sample programs 417
Record descriptions for COBOL sample programs . . 431

Appendix F. Sample programs (PL/I) 433
Operator instruction program (PL/I) 434
Inquiry/update sample program (PL/I) 435
Browse sample program (PL/I) 441
Order entry sample program (PL/I) 447
Order entry queue print sample program (PL/I) . . . 451
Low balance report sample program (PL/I) 454
Maps and screen layouts for PL/I sample programs . 457
Record descriptions for PL/I sample programs . . . 471

Index 473

Summary of changes

This edition is based on the *CICS/MVS Application Programmer's Reference* manual (SC33-0512-0), and incorporates updates and revisions as well as enhancements introduced by CICS/MVS 2.1.1 and CICS/MVS 2.1.2. These enhancements are described in the *CICS/MVS Release Guide*.

The opportunity has also been taken to correct errors and incorporate readers' comments.

All changes that are new in this edition, other than editorial changes, are marked by revision bars in the left margin, like this paragraph.

Questionnaire

CICS/MVS Version 2 Release 1 Modification 2 Application Programmer's Reference

Publication No. SC33-0512-01

To help us produce books that meet your needs, please fill in this questionnaire. A reader's comment form is also included at the back of this book should you want to make more detailed comments. Whichever form you use, your comments will be sent to the author's department for review and appropriate action.

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

1. Please rate the book on the points shown below

The book is:

accurate	1	2	3	4	5	inaccurate
readable	1	2	3	4	5	unreadable
well laid out	1	2	3	4	5	badly laid out
well organized	1	2	3	4	5	badly organized
easy to understand	1	2	3	4	5	incomprehensible
adequately illustrated	1	2	3	4	5	inadequately illustrated
has enough examples	1	2	3	4	5	has too few examples

And the book as a whole?

excellent	1	2	3	4	5	poor

2. Which topics does the book handle well?

3. And which does it handle badly?

4. How could the book be improved?

5. How often do you use this book? Less than once a month? □ Monthly? □ Weekly? □ Daily? □

6. What sort of work do you use CICS for?

7. How long have you been using CICS?________years/months

8. Have you any other comments to make?

Thank you for your time and effort. No postage stamp necessary if mailed in USA. (If you are outside the USA, please mail this form to your local IBM office or representative who will be happy to forward your comments or you may mail directly to either address in the Edition Notice on the back of the title page.) Be sure to print your name and address below if you would like a reply.

Name..Job Title ....

Company..Address...

....Zip.......................................

Reader's Comments
SC33-0512-01

IBM®

Cut or F
Along Li

Fold and Tape

Please do not staple

Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department 6R1H
180 KOST ROAD
MECHANICSBURG PA 17055-0786

Fold and Tape

Please do not staple

Fold and Tape

Cut or F
Along Li

Part 1. Command level programming

Chapter 1.1. Introduction to command level programming 3

Chapter 1.2. Command format and argument values 5

Chapter 1.3. Command language translator 9

Chapter 1.4. Programming techniques and restrictions 17

Chapter 1.5. Exceptional conditions 43

Chapter 1.6. Access to system information 51

Chapter 1.7. Execution (command level) diagnostic facility 57

Chapter 1.8. Command level interpreter 67

Chapter 1.9. Temporary storage browse 73

Chapter 1.1. Introduction to command level programming

The IBM Customer Information Control System/Multiple Virtual Storage (CICS/MVS*) command level application programming interface allows you to request CICS services by CICS **commands**. These commands are statements that you can include at appropriate points in your application program. They have a format similar to the statements of the programming language in use.

You can include CICS commands in application programs written in assembler language, COBOL, or PL/I. The commands are essentially the same in each language, differing only in the delimiter used.

Application programs that include CICS commands are processed by the **command language translator**, which translates the commands into statements in the language being used. You can then assemble (or compile) and link edit your programs in the usual way. When your application programs are executed, the statements inserted by the translator invoke the **EXEC interface program**, which provides the service requested by each command by invoking one or more CICS control programs.

Besides invoking CICS control programs, the EXEC interface program obtains and provides addressability to required areas of storage, which are released automatically when no longer required.

Generally, you need only select the required function and code the appropriate command. You do not normally need to know about CICS storage areas and control blocks; in those cases when you do need access to such areas, the command level interface provides commands for this purpose, as described in "Chapter 1.6. Access to system information" on page 51.

For a basic description of CICS, see the *CICS Application Programming Primer*, which has been designed to help you learn, step by step, how to write a realistic CICS application program, using the CICS command level interface and the COBOL programming language.

The primer answers questions like "What is CICS?," "Why have CICS?," "What does CICS do?," "How does a CICS-based system differ from a batch system?," "How does CICS help you set up an online system?," and "How do you use CICS?"

For information about the performance of a CICS application program, see the *CICS/MVS Performance Guide*.

Commands instead of macros

You are advised to use the command level interface instead of the macro level interface for all new application programs.

Usually, macro level programs will work as before unless you need to use the new or enhanced functions introduced in the more recent releases, because these functions are only carried out at command level.

If you have an existing macro level program that you enhance using CICS commands, you will sometimes get unpredictable results because of addressability and storage problems. (See "CICS macros used with CICS commands" on page 18 for more details.)

It is therefore better to convert your macro level program to command level and then add the enhancements using CICS commands. There is a program that will do most of this conversion for you. It is called the *IBM CICS Conversion Utility Program Offering (CICS/CVT)*, program number 5789-DPL.

Similar considerations apply if you have a command level program that invokes a macro level program, and vice-versa. You must ensure that addressability is maintained across such invocations.

CICS syntax notation used in this book

In the CICS books, we show you CICS commands in a standard way.

We do not include the EXEC CICS that always precedes each command's keyword; nor do we include the END EXEC statement used in COBOL programs, or the semicolon (;) used in PL/I programs, which you must code at the end of each CICS command.

* IBM Trademark. For a list of trademarks see page iii.

We use a number of symbols to show you how to code the commands. ***These symbols are not part of the command, so never include them in your code.*** They are as follows:

Symbol	**Meaning**
ƀ or b	A blank that you must code.
[ ]	You can, but need not, code the enclosed options.
{ }	A set of alternatives—one of which you ***must*** code.
|	A logical "OR" symbol that separates alternatives. If the alternatives are between { and }, you ***must*** code one of them. If they are between [and], you ***may*** code one of them.
——	Underlining indicates that the option is a default that is assumed unless you specify otherwise.
...	An ellipsis indicates that you can code the immediately preceding identifier(s) more than once.
Punctuation and uppercase characters	Code exactly as shown.
Lowercase characters	Code your own text, as appropriate (for example, `name`).

For example, with `READ FILE(name)` you must code `READ FILE` and () unchanged, but are free to code any valid text string to mean the name of the file.

Thus, to show that either GTEQ, or EQUAL, or neither, can be coded (and that EQUAL is the default), the syntax notation is:

```
[GTEQ|EQUAL]
```

Chapter 1.2. Command format and argument values

This chapter explains the general rules governing the use of the CICS commands that are described in the chapters that follow.

Command format

The general format of a CICS command is EXECUTE CICS (or EXEC CICS) followed by the required **function** name, and possibly by one or more **options**, as follows:

```
EXEC CICS function [option[(arg)]]...
```

where:

- "function" describes the operation required (for example READ).
- "option" describes any of the many optional facilities available with each function. Some options are followed by an argument in parentheses, others are not. You can write options (including those that require arguments) in any order.
- "arg" (short for argument) is a value such as "data-value" or "name", as defined below.

An example of a CICS command (from "Chapter 2.4. File control – commands, options, and conditions" on page 93) is as follows:

```
EXEC CICS READ FILE('FILEA') INTO(FILEA)
   RIDFLD(KEYNUM) UPDATE
```

You must add the appropriate end-of-command delimiter, described in the next section.

Coding conventions

You can include CICS commands in an assembler language, COBOL, or PL/I program anywhere that an executable statement can be included.

In assembler language:

- You must code the keyword EXEC in an operator position. You can also label the command.
- You must use either a blank or a comma, but not both, as the delimiter between options. The appearance of 'comma-blank' or 'period-blank' immediately following an option shows that the rest of the line is a comment.
- The usual continuation conventions for macros apply (use a nonblank character in column 72, and start the continuation line in column 16).

In COBOL, you must delimit a command with 'END-EXEC'. For example:

```
EXEC CICS ISSUE RESET END-EXEC
```

This delimiter allows you to write a command within a THEN clause.

In PL/I, you must delimit a command with a semicolon in PL/I. For example:

```
EXEC CICS ISSUE RESET;
```

In the following chapters, for simplicity, the syntax of each of the commands that you can specify in an application program is presented without the phrase EXEC CICS, without the continuation conventions, and without the end-of-command delimiter (END-EXEC or semicolon).

In the programming examples in the text, the phrase EXEC CICS is added but not the continuation conventions or end-of-command delimiter. When coding commands, you must add these as appropriate for the programming language you are using.

Argument values

The parenthesized argument values that follow options in a CICS command are specified as follows:

- data-value
- data-area
- pointer-value (or ptr-value)
- pointer-ref (or ptr-ref)
- name
- label
- hhmmss.

When a CICS command offers the LENGTH option, the LENGTH is generally expressed as a signed half-word binary value. This puts a theoretical upper limit of 32767 bytes on the LENGTH. In practice (depending on issues of recoverability, function shipping, and so on) the achievable upper limit varies from command to command, but will be somewhat less than the theoretical maximum.

Whatever the CICS command, to be on the safe side, do not let the LENGTH you code exceed 24K bytes.

For ***journaled items***, the length may be further restricted by the buffer size of the journal.

For ***journal*** commands, the restrictions apply to the sum of the LENGTH and PFXLENG values (see "Chapter 5.5. Journal control" on page 327).

Finally, for ***temporary storage, transient data, and file control***, the dataset definitions may themselves impose further restrictions. You will find any such restrictions documented in the books describing installation and resource definition.

Most programmers are unlikely to find a 24K-byte limit a hindrance; for the sake of efficiency and response time, online programs will not often handle such large amounts of data.

The argument values are defined in the sections that follow.

Argument values in assembler language

Usually, an argument may be either the address of the data or the data itself (in assembler language terms, either a relocatable expression or an absolute expression).

A relocatable expression must not contain unmatched brackets (outside quotes) or unmatched quotes (apart from length attribute references). Provided this rule is obeyed, any expression may be used, including literal constants such as =AL2(100), forms such as 20(0,R11), and forms that use the macro replacement facilities.

An absolute expression must be a single term that may be either a length attribute reference or a self-defining constant.

Care must be taken with equated symbols, which should be used only when referring to registers (pointer references). If an equated symbol is used for a length, say, it will be treated as the address of the length and an unpredictable error will occur.

- 'data-value' can be replaced by a relocatable expression that is an assembler-language reference to data of the correct type for the argument, or by a constant of the correct type for the argument.
- 'data-area' can be replaced by a relocatable expression that is an assembler-language reference to data of the correct type for the argument.
- 'pointer-value' can be replaced by an absolute expression that is an assembler-language reference to a register.
- 'pointer-ref' can be replaced by an absolute expression that is an assembler-language reference to a register.
- 'name' can be replaced **either** by a character string in quotes, **or** by an assembler-language relocatable expression reference to a character string with a length equal to the maximum length allowed for the name. The value of the character string is the name to be used by the argument.
- 'label' is intended to refer to a destination address to which control is transferred. It can be replaced by the label of the destination instruction or by the label of an address constant for the destination. This constant must not specify a length.

  The expression used is =A(dest) where "dest" is a relocatable expression denoting the destination.

  For example, the following commands are equivalent:

  ```
  HANDLE CONDITION ERROR(DEST)
  HANDLE CONDITION ERROR(ADCON)
  HANDLE CONDITION ERROR(=A(DEST))
  ⋮
  DEST BR 14
  ADCON DC A(DEST)
  ```
- 'hhmmss' can be replaced by a self-defining decimal constant or an assembler language reference to a field defined as PL4. The value must be of the form 0HHMMSS+ where HH represents hours from 00 through 99, MM represents minutes from 00 through 59, and SS represents seconds from 00 through 59.

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. However, if a data area is specified as the source or target, it is not necessary to provide the length explicitly because the command language translator will generate a default length.

Although the DESTIDLENG, FROMLENGTH, KEYLENGTH, LENGTH, PFXLENG, TOLENGTH, and VOLUMELENG options are shown as required options in the syntax for a command, these options are always optional in an assembler language program that specifies a data area (except for the ENQ and DEQ commands). In most cases, the LENGTH option must be specified if SET is used; the syntax of each command and its associated options show whether this rule applies.

Argument values in COBOL

- '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-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.

  In cases where the data type is unspecified, the data area can refer to an elementary or group item.
- 'pointer-value' can be replaced by any BLL (base locator for linkage) cell name, or by any COBOL data

name that contains a copy of such a pointer in a BLL cell.

- 'pointer-ref' can be replaced by any BLL cell name.
- 'name' can be replaced by either of the following:
  - A character string in quotes (that is, a nonnumeric literal). If this is shorter than the required length, it is padded 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 a section name.
- 'hhmmss' can be replaced by a decimal constant or by a data name of the form 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, and SS represents seconds from 00 through 59.

Argument values in PL/I

- 'data-value' can be replaced by any PL/I 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 – FIXED BIN(15)

  Fullword binary – FIXED BIN(31)

  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 PL/I data reference that has the correct data type for the argument. The data type can be specified as one of the following:

  Halfword binary – FIXED BIN(15)

  Fullword binary – FIXED BIN(31)

  Character string – CHAR(n) where "n" is the number of bytes.

  If the data type is unspecified, the data area can refer to an element, array, or structure; for example, FROM(P– >STRUCTURE) LENGTH(LNG). The reference must be to connected storage.

  The data area must also have the correct PL/I alignment attribute. This is ALIGNED for binary items, and UNALIGNED for strings.

  If data that is not in varying length string format is read into a varying length string, the length bytes at the beginning of the varying length string will be corrupted.
- 'pointer-value' (which includes "pointer-ref" as a subset) can be replaced by any PL/I expression that can be converted to POINTER.
- 'pointer-ref' can be replaced by any PL/I reference of type POINTER ALIGNED.
- 'name' can be replaced by either of the following:
  - A character string in quotes (that is, a literal constant)
  - A PL/I expression or reference whose value can be converted to a character string with a length equal to the maximum length allowed for the name. The value of the character string is the name to be used by the argument.
- 'label' can be replaced by any PL/I expression whose value is a label.
- 'hhmmss' can be replaced by a decimal constant or an expression that can be converted to a FIXED DECIMAL(7,0) value. The value must be of the form 0HHMMSS+ where HH represents hours from 00 through 99, MM represents minutes from 00 through 59, and SS represents seconds from 00 through 59.

If the UNALIGNED attribute is added to the ENTRY declarations generated by the CICS translator by a DEFAULT DESCRIPTORS statement, data area or pointer reference arguments to CICS commands must also be UNALIGNED. Similarly for the ALIGNED attribute; data area or pointer reference arguments must be ALIGNED.

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. However, if a data area is specified as the source or target, it is not necessary to provide the length explicitly because the command language translator will generate a default length value of either STG(data-area) or CSTG(data-area) as appropriate.

Although the DESTIDLENG, FROMLENGTH, KEYLENGTH, LENGTH, PFXLENG, TOLENGTH, and VOLUMELENG options may be shown as required options in the syntax for a command, these options are always optional in a PL/I program that specifies a data area (except for the ENQ and DEQ commands). In most cases, the LENGTH option must be specified if SET is used; the syntax of each command and its associated options show whether this rule applies.

Chapter 1.3. Command language translator

The command language translator accepts as input a source program written in assembler language, COBOL, or PL/I in which CICS commands have been coded, and produces as output an equivalent source program in which each command has been translated into a call macro or statement in the language of the source program.

At execution time, the call macro or statement invokes the EXEC interface program, which accepts the arguments passed by the call macro or statement, sets up the parameters in the CICS control blocks, and passes control to the appropriate CICS facility.

The translator is executed in a separate job step. The job step sequence for preparing an application program is translate — assemble (or compile) — link-edit. Cataloged procedures are supplied to assist the user; see the *CICS/MVS Installation Guide* for details. The translator requires a region of 256K bytes.

There are three separate translators: one for assembler language, one for COBOL, and one for PL/I. Each translator reads its input from SYSIN.

For COBOL and PL/I, the translator writes its output (the translated source program) on SYSPUNCH, and writes the source listing, error messages, and so on, on SYSPRINT.

For assembler language, the translator writes its output to SYSPUNCH, which also contains error messages (if any) as assembler comments. SYSPRINT contains only a list of translator options, the number of messages produced together with the highest message severity, and the return code from the translator step.

All the translators also accept the commands that can be used to access DL/I databases. These commands are identified by EXEC DLI and are translated in a similar way to the EXEC CICS commands; they are described in "Chapter 2.5. DL/I services (EXEC DLI command)" on page 101.

Translator data sets

Input data set

The input data set must be a sequential data set. It may be on punched cards, on a direct-access device, or on magnetic tape.

The input data set for assembler language can contain either fixed-length or variable-length records.

The input data set for COBOL must contain fixed-length records (blocked or unblocked).

The input data set for PL/I can contain either fixed-length or variable-length records.

The record length (LRECL) must not exceed 104 bytes.

Output data set

The output data set must be a sequential data set. It may be on punched cards, on a direct-access device, or on magnetic tape.

The output data set must contain 80-byte fixed-length records (blocked or unblocked).

Later copying or manipulating of statements originally inserted by the CICS translator in an application program may produce unpredictable results.

Listing data set

The listing data set must be a sequential data set. Although the listing is usually printed, it can be stored on any direct-access device or on magnetic tape.

The listing data set for COBOL must contain 121-byte fixed-length blocked records (RECFM=FBA).

The listing data set for assembler language and PL/I must contain variable-length blocked records with a maximum length of 121 bytes (RECFM=VBA).

Translated code

Assembler language

The invocation of a CICS assembler-language application program obeys system standards. This means that, on entry to the application program, registers 1, 15, 14, and 13 contain the following:

- Register 1 contains the address of the parameter list; there are two entries in this list, as follows:
  - Address of the EIB (EXEC interface block)
  - Address of the COMMAREA; if there is no COMMAREA, EIBCALEN will be zero.
- Register 15 contains the address of the entry point
- Register 14 contains the address of the return point
- Register 13 contains the address of the save area.

All other registers are undefined.

DFHECALL macro: For an assembler-language application program, each command is replaced by an invocation of the DFHECALL macro.

This macro expands to a system-standard call sequence using registers 15, 14, 0, and 1:

- Register 15 contains the address of the entry point in the EXEC interface program
- Register 14 contains the address of the return point in your application program
- Register 0 is undefined
- Register 1 contains the address of the parameter list.

The entry point held in register 15 is resolved in the EXEC interface processor (DFHEAI), which must be link-edited with your application program.

You can specify the exit from the application program by a CICS RETURN command in your source program. Alternatively, you can let the translator-inserted macro DFHEIRET, which has been inserted before the END statement, do it. This macro restores the registers and returns control to the address in register 14.

During assembly, the DFHECALL macro builds an argument list in dynamic storage, so that the application program is reentrant, and then invokes the EXEC interface program (DFHEIP). DFHEIP also obeys system standards, as described above.

The translator inserts the following macros into your source program and invokes the DFHECALL macro:

DFHEIENT. This macro is inserted after the first CSECT or START instruction. It does prolog code; that is, it:

- Saves registers
- Obtains an initial allocation of the storage defined by DFHEISTG (see below)
- Sets up a base register (default register 3)
- Sets up a dynamic storage register (default register 13)
- Sets up a register to address the EIB (default register 11).

DFHEIRET. This macro performs epilog code; that is, it:

- Restores registers
- Returns control to the address in register 14.

DFHEISTG and DFHEIEND. These macros define dynamic storage; that is, they:

- Define the storage required for the parameter list
- Define a save area.

A copy book, DFHEIBLK, containing a DSECT that describes the EIB is also included automatically.

The example in Figure 1 on page 11 shows a simple assembler-language application program that uses the BMS command SEND MAP to send a map to a terminal. The lower part of the figure shows the output after program INSTRUCT has been translated.

```
INSTRUCT CSECT
         EXEC CICS SEND MAP('DFH$AGA') MAPONLY ERASE
         END


The above source program is translated to:

         DFHEIGBL ,                INSERTED BY TRANSLATOR
INSTRUCT CSECT
         DFHEIENT                  INSERTED BY TRANSLATOR
*        EXEC CICS SEND MAP('DFH$AGA') MAPONLY ERASE
         DFHECALL =X'1804C000080000000004620400002 0',(CHA7,=CL7'DFH$AGA*
               '),(______RF,DFHEIV00)
         DFHEIRET                  INSERTED BY TRANSLATOR
         DFHEISTG                  INSERTED BY TRANSLATOR
         DFHEIEND                  INSERTED BY TRANSLATOR
         END
```

Figure 1. Translated code for a CICS command

Extensions to dynamic storage: You can extend dynamic storage to provide extra storage for user variables. You do this by defining these variables in your source program in a DSECT called DFHEISTG.

The maximum amount of dynamic storage obtainable using the DFHEISTG DSECT is 65264 bytes. (DFHEISTG is a reserved name.) At translation, the translator inserts the DFHEISTG macro immediately following your DFHEISTG DSECT instruction. In this way the DSECT describes dynamic storage needed for the parameter list, for the command-level interface, and for any user variables.

The example in Figure 2 on page 12 shows a simple assembler-language application program that uses such variables in dynamic storage.

```
DFHEISTG DSECT
         COPY DFH$AGA         INPUT MAP DSECT
         COPY DFH$AGB         OUTPUT MAP DSECT
MESSAGE  DS   CL39
INQUIRY  CSECT
         EXEC CICS RECEIVE MAP('DFH$AGA')
         MVC  NUMBO,KEYI
         MVC  MESSAGE,=CL(L'MESSAGE)'THIS IS A MESSAGE'
         EXEC CICS SEND MAP('DFH$AGB') ERASE
         END


The above source program is translated to:

         DFHEIGBL ,           INSERTED BY TRANSLATOR
DFHEISTG DSECT
         DFHEISTG             INSERTED BY TRANSLATOR
         COPY DFH$AGA         INPUT MAP DSECT
         COPY DFH$AGB         OUTPUT MAP DSECT
MESSAGE  DS   CL39
INQUIRY  CSECT
         DFHEIENT             INSERTED BY TRANSLATOR
*        EXEC CICS RECEIVE MAP('DFH$AGA')
         DFHECALL =X'1802C0000800000000040900000020',(CHA7,=CL7'DFH$AGA*
               '),(______RF,DFH$AGAI)
         MVC  NUMBO,KEYI
         MVC  MESSAGE,=CL(L'MESSAGE)'THIS IS A MESSAGE'
*        EXEC CICS SEND MAP('DFH$AGB') ERASE
         DFHECALL =X'1804C000080000000004E204000020',(CHA7,=CL7'DFH$AGB*
               '),(______RF,DFH$AGBO)
         DFHEIRET             INSERTED BY TRANSLATOR
         DFHEISTG             INSERTED BY TRANSLATOR
         DFHEIEND             INSERTED BY TRANSLATOR
         END
```

Figure 2. Translated code for user variables

Multiple base registers: The values provided by the automatic insertion of DFHEIENT may be inadequate for application programs that produce a translated output greater than 4095 bytes.

For example, by default, the translator sets up only one base register (register 3) and, in some circumstances (for example, when the DLI translator option has been specified), the literals produced by the translator initializing the DIB could fall outside the range of that single base register.

To overcome this problem, you can prevent the translator from automatically inserting its version of the DFHEIENT macro by specifying the NOPROLOG translator option. This will enable you to specify your own DFHEIENT macro with the CODEREG operand so that you can specify more than one base register.

You must code your own version of the DFHEIENT macro, which can have up to three operands, in place of the first CSECT or START instruction in your source program. The three operands are:

- CODEREG – base registers
- DATAREG – dynamic storage registers
- EIBREG – register to address the EIB.

For example, the source code shown in Figure 1 on page 11 would become:

```
INSTRUCT  DFHEIENT,
          CODEREG=(2,3,4),
          DATAREG=(13,5),
          EIBREG=6
          EXEC CICS SEND
          MAP('DFH$AGA')
          MAPONLY ERASE
          END
```

The symbolic register DFHEIPLR is equated to the first DATAREG that is either explicitly specified or obtained by default. Because register 13 points to the save area defined in dynamic storage by DFHEISTG, you should use register 13 as the first dynamic storage register.

DFHEIPLR will be assumed by the growth of a CICS command to contain the value set up by DFHEIENT. Either dedicate this register or ensure that it is restored before each CICS command.

Assembler language programs that are translated with the DLI option will have a DLI initialization call inserted after each CSECT statement. Assembler language programs larger than 4095 bytes which do not use the CODEREG parameter of the DFHEIENT macro to establish multiple base registers, must include an LTORG statement to ensure that the literals, generated by either DFHEIENT or a DLI initialization call, fall within the range of the base register.

Usually, an LTORG statement is needed for every CSECT that exceeds 4095 bytes in length.

Commands mixed with macros: An assembler-language application program that uses both the command level interface and the macro level interface (that is, a mixture of commands and macros) must define the macro global bit &DFHEIMX and set it to 1. This will ensure that register 13 points to the CSA, and register 12 to the TCA. Here, DFHEIPLR will not be assumed by the growth of a CICS command.

COBOL

For a COBOL application program, each command is replaced by one or more COBOL MOVE statements followed by a COBOL CALL statement.

MOVE statements assign constants to COBOL data variables; this allows constants and names to be specified as arguments to options in the commands. For example, a command such as:

```
EXEC CICS RECEIVE MAP('A') END-EXEC
```

may be translated to:

```
MOVE ' ' TO DFHEIV0
MOVE 'A' TO DFHC0070
CALL 'DFHEI1' USING DFHEIV0 DFHC0070 AI
```

Declarations for the generated variables DFHEIV0 and DFHC0070 are included automatically in working storage; their names are reserved. The string within the quotes moved to DFHEIV0 consists of characters, some of which may be unprintable. Avoid using EXEC, CICS, DLI, 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 declarations of the temporary variables that it requires into the working-storage section.

PL/I

For a PL/I application program, each command is always replaced by a DO statement, a declaration of a generated entry name, a CALL statement, and an END statement. The ENTRY declaration ensures that the appropriate conversions for argument values take place.

If a PL/I on-unit consists of a single EXEC CICS command, the command should be inside a BEGIN block, for example:

```
ON ERROR BEGIN;
         EXEC CICS RETURN;
         END;
```

In a similar way, if an EXEC CICS command is associated with a PL/I condition prefix, the command should be inside a BEGIN block, for example:

```
(NOZERODIVIDE):  BEGIN;
                 EXEC CICS GETMAIN
                 SET(ptr-ref)
                 LENGTH(data-value);
                 END;
```

If OPTIONS(MAIN) is specified, the translator modifies the parameter list by inserting the EIB structure pointer as the first parameter. If OPTIONS(MAIN) is not specified (that is, if the program is to be link-edited to the main module), the parameter list is not modified, and it is the application programmer's responsibility to address the EIB structure in the link-edited program if access to it is required. In any case, where a program begins with a valid PL/I PROCEDURE statement, the translator will insert the declaration of the EIB structure.

Translator options

The translator provides several optional facilities: for example, to allow for different record formats and to specify what information is required on the listing. The translator options and their defaults (underlined) are listed below.

Translator options are specified in the *ASM statement for assembler language, the CBL statement for COBOL, or in the *PROCESS statement for PL/I. Except for VS COBOL II programs translated with the ANSI85 option, these statements must precede the source program; **there is no batching facility** for other languages or for VS COBOL II without the ANSI85 option. The *ASM statement must obey the same syntax and continuation rules as the assembler-language comment statement.

Translator options may also be specified in the PARM operand of the EXEC job control statement that invokes the translator.

If both methods are used, the options specified in the *ASM, CBL, or *PROCESS statements override those in the

EXEC job control statement, and the last setting for each option takes precedence.

Translator options are written as a list within the XOPTS keyword option, for example:

```
*ASM XOPTS(NOPROLOG NOEPILOG)
```

or

```
CBL XOPTS(QUOTE SPACE(2))
```

or

```
*PROCESS XOPTS(FLAG(W) SOURCE);
```

No characters, other than blanks, can appear before the CBL statement on the COBOL options card.

The options may appear in any order. They may be separated by one or more blanks or by a comma[1] . If coded in the PARM operand of the EXEC job control statement, the XOPT keyword (and its associated parentheses) is unnecessary; only options for the translator are permitted.

For compatibility with previous releases, the CICS keyword can be used as an alternative to XOPTS.

If the application program contains EXEC DLI commands, the options DLI and CICS must be specified in an *ASM, CBL, or *PROCESS statement, as follows:

```
*ASM XOPTS(DLI,CICS)
```

or

```
CBL XOPTS(DLI,CICS)
```

or

```
*PROCESS XOPTS(DLI,CICS);
```

The CBL or *PROCESS statement can also contain options that apply to the following compiler. These options will be ignored by the translator (that is, they will not be checked for validity) but they will be copied through onto the output data set. For example, a PL/I application program preceded by:

```
*PROCESS XOPTS(SOURCE),ATTRIBUTES;
```

will be passed to the PL/I compiler preceded by:

```
*PROCESS ATTRIBUTES;
```

The following translator options apply to all three languages (ASM, COBOL, and PL/I) except where stated otherwise.

ANSI85(VS COBOL II only)
specifies that the translator is to translate VS COBOL II programs that implement the ANSI85 standards. If this option is specified, the **COBOL2** option is assumed.

CICS
specifies that the translator is to process EXEC CICS commands. This option may be specified either as an alternative to, or as a suboption of, the XOPTS option. If neither XOPTS nor CICS is specified, CICS is assumed by default. This option must not be specified for batch DL/I application programs containing EXEC DLI commands; XOPTS(DLI) must be specified instead.

COBOL2 (VS COBOL II only)
specifies that the translator is to translate VS COBOL II programs.

DEBUG|NODEBUG (COBOL and PL/I only)
specifies whether the translator is to produce code that passes the translator line number through to CICS to be displayed by the execution (command level) diagnostic facility (EDF).

DLI
specifies that the translator is to process EXEC DLI commands.

EDF|NOEDF
specifies whether the EDF is to apply to the program. There is no performance advantage in specifying NOEDF, but the option can be useful to prevent commands in well debugged subprograms appearing on EDF displays.

EPILOG|NOEPILOG (ASM only)
specifies that the translator is to insert the DFHEIRET macro. NOEPILOG prevents the translator inserting the DFHEIRET macro. DFHEIRET is described on page 10.

NOFE|FE
produces translator informative messages which print (in hexadecimal notation) the bit pattern corresponding to the first argument of the translated call. This bit pattern has the encoded information that the EXEC interface program uses to determine which function is required and which options are specified. If FE is specified, all diagnostic messages are listed, whatever the FLAG option specifies.

FLAG[(I|W|E|S)] (COBOL and PL/I only) – abbreviation: F
specifies the minimum severity of error that requires a message to be listed.

FLAG(I)
All messages

FLAG|FLAG(W)
All except informative messages

FLAG(E)
All except warning and informative messages

[1] Any keyword not enclosed in the brackets following the XOPTS|CICS operand is passed to the compiler as a compiler option.

FLAG(S)
Only severe and unrecoverable error messages.

GDS (ASM only)
specifies whether the translator is to process EXEC CICS GDS commands. For more information, see the *CICS/MVS Intercommunication Guide*.

LANGLVL(1)|LANGLVL(2) (OS/VS COBOL only)
specifies whether the translator is to analyze the source program and generate code according to the American National Standard X3.23-1968 (LANGLVL(1)) or X3.23-1974 (LANGLVL(2)) interpretations. The same value for this option must be specified for the translator and following compiler.

LINECOUNT(n) (COBOL and PL/I only) – abbreviation: LC
specifies the number of lines to be included in each page of translator listing, including heading and blank lines. The value of "n" must be an integer in the range 1 through 255; if "n" is less than 5, only the heading and one line of listing will be included on each page. The default is 60.

MARGINS(m,n[,c]) (PL/I only) – abbreviation: MAR
specifies the extent of the part of each input line or record that contains PL/I statements. The translator does not process data that is outside these limits (but it does include it in the source listings).

The option can also specify the position of an American National Standard printer control character to format the listing produced if the SOURCE option applies; otherwise the input records will be listed without any intervening blank lines.

"m" – column number of left-hand margin.

"n" – column number of right-hand margin. It must be greater than "m".

"c" – column number of the American National Standard printer control character. It must be outside the values specified for "m" and "n". A zero value for "c" means no printer control character. Only the following printer control characters can be used (b represents a blank):

b	Skip 1 line before printing.
0	Skip 2 lines before printing.
–	Skip 3 lines before printing.
+	No skip before printing.
1	Start new page.

The default is MARGINS(2,72,0) for fixed-length records and MARGINS(10,100,0) for variable-length records.

PROLOG|NOPROLOG (ASM only)
specifies that the translator is to insert the DFHEISTG, DFHEIEND, and DFHEIENT macros. NOPROLOG prevents the translator inserting the DFHEISTG, DFHEIEND, and DFHEIENT macros. These macros are described on page 10.

NUM|NONUM (COBOL only)
specifies whether the translator is to use the line numbers appearing in columns 1 through 6 of the card as the line number in its diagnostic messages and cross-reference listing. If NUM is not specified, the translator generates its own line numbers.

OPMARGINS(m,n[,c]) (PL/I only) – abbreviation: OM
specifies the translator output margins; that is, the margins of the input to the following compiler. Normally these will be the same as the input margins. For the meaning of "m", "n", and "c" see MARGINS. The default is OPMARGINS (2,72,0).

OPSEQUENCE(m,n)|NOOPSEQUENCE (PL/I only) – abbreviations: OS and NOS
specifies the position of the sequence field in the output records. For the meaning of "m" and "n" see SEQUENCE. The default is OPSEQUENCE(73,80).

OPT|NOOPT (COBOL only)
specifies whether the translator is to generate SERVICE RELOAD statements to address the EIB and DFHCOMMAREA. You must specify this option if the translated program is to be compiled using the optimization feature of COBOL. If the program is not optimized, you need not specify OPT.

OPTIONS|NOOPTIONS – abbreviations: OP and NOP
specifies whether the translator is to include in the listing a list of **all** the translator options used during this translation.

QUOTE|APOST (COBOL only)
suggests to the translator that the double quotation marks (") should be accepted as the character to delineate literals; APOST suggests that the apostrophe (') should be accepted instead. The same value must be specified for the translator and following compiler.

The CICS-supplied COBOL copy books use APOST.

SEQ|NOSEQ (COBOL only)
shows whether the translator is required to check the sequence of source statements. If SEQ is specified and a statement is not in sequence it is flagged.

If the **ANSI85** option is specified, the translator does no sequence checking and the **SEQ|NOSEQ** option is ignored.

SEQUENCE(m,n)|NOSEQUENCE (PL/I only) – abbreviations: SEQ and NSEQ
specifies the extent of the part of each input line or record that contains a sequence number. This number is included in the source listing and is used in the error message and cross-reference listings. No attempt is made to sort the input lines or records into sequence. If no sequence field is specified, the translator creates and prints its own sequence numbers in the source listing; this is so that the error

messages and cross-reference listings can refer to a particular line in the source listing.

"m" — column number of left-hand margin.

"n" — column number of right-hand margin.

The extent must not exceed 8 characters and must not overlap the source program (as specified in the MARGINS option).

The default for fixed-length records is SEQUENCE(73,80); for varying-length records, the default is SEQUENCE(1,8).

SOURCE|NOSOURCE (COBOL and PL/I only)
specifies whether the translator is to produce a listing of the source program.

SPACE(1|2|3) (COBOL only)
shows the type of spacing to be used in the output listing: SPACE(1) specifies single spacing; SPACE(2) double spacing; and SPACE(3) triple spacing.

SPIE|NOSPIE
specifies that the translator is to trap unrecoverable errors. NOSPIE prevents the translator from trapping unrecoverable errors; instead, a dump is produced.

VBREF|NOVBREF (COBOL and PL/I only)
specifies whether the translator is required to provide a cross-reference list of all the commands used in its input. For compatibility, XREF and NOXREF are still accepted.

Chapter 1.4. Programming techniques and restrictions

This chapter contains information that will help improve the performance and efficiency of an application program in the CICS system.

The first section deals with general programming techniques; this section gives advice about the virtual-storage environment in which CICS application programs operate. The rest of the chapter contains information that is applicable only to programs written in assembler language, COBOL, and PL/I, and includes the restrictions that apply to each language when CICS commands are used.

This manual does not contain any guidance on the use of programming language statements or programming techniques that are unrelated to CICS; such information is given in the appropriate language publications.

Files and queues are not defined within application programs; these definitions are established with the help of the system programmer. See the *CICS/MVS Resource Definition (Macro)* manual for further information on these definitions.

General programming techniques

To see how programming techniques can affect the performance and efficiency of the CICS system, it is necessary to understand something about the virtual storage environment in which CICS operates. Two concepts are important: multithreading and virtual-storage paging.

Multithreading is a technique used by CICS which allows a single copy of an application program to process several transactions concurrently. For example, the first section of an application program may be processing one transaction. When that section is completed (in general, signaled by the execution of a CICS command that causes a wait), processing of another transaction using a different section of the application program may take place. (Compare this with single threading, which is the execution of a program to completion. Processing of one transaction is completed before another transaction is started.)

Multithreading requires all CICS application programs to be quasi-reentrant; that is, they must be serially reusable between entry and exit points, and any instructions or data altered in them must be restored. CICS application programs using the command level interface obey this rule automatically (if, in PL/I programs, static storage is used for read-only data). For these programs to stay reentrant, variable data should not appear as static storage in PL/I, nor as a DC in the program CSECT in assembler language. For COBOL programs, quasi-reenterability is ensured by a fresh copy of working storage being obtained each time the program is invoked.

Care must be taken if a program involves lengthy calculations; because an application program retains control from one CICS command to the next, processing of other transactions is completely excluded. However, the task control SUSPEND command can be used to allow other transaction processing to proceed; see "Chapter 4.3. Task control" on page 285 for details.

Virtual-storage paging is a technique used by CICS in a virtual-storage environment. The key objective of programming in this environment is the reduction of page faults. A page fault occurs when a program refers to instructions or data that do not reside in real storage, in which case the page in virtual storage that contains the referenced instructions or data must be paged into real storage. The more paging required, the lower the overall system performance.

Although an application program is not precluded from direct communication with the operating system, the results of such action are unpredictable and can degrade performance.

An understanding of the following terms is necessary for writing application programs to be run in a virtual-storage environment:

- **Locality of reference** — the consistent reference, during the execution of the application program, to instructions and data within a relatively small number of pages (compared with the total number of pages in a program) for relatively long periods.
- **Working set** — the number and combination of pages of a program required during a given period.
- **Validity of reference** — direct reference to the required pages, without intermediate storage references that retrieve useless data.

In general, the following techniques should be used:

1. To improve locality of reference, processing should be sequential for both code and data, where possible.
   a. The ideal application program executes sequentially with no branch logic reference beyond a small range of address space. However, error-handling or unusual-situation routines should be separated from the main section of a program; they should be subprograms.
   b. Subroutines should be placed near to the caller.
   c. Subprograms that are short and used only once or twice (other than error-handling or unusual-situation routines) should be coded inline in the calling program.

   d. Try to keep the execution path in a straight line by using XCTL commands to transfer control to other programs when necessary, rather than LINK commands.
   e. Initialize data as close as possible to its first use.
   f. Define arrays or other data structures in the order in which they will be referred to. Refer to elements within arrays in the order in which they are stored; for example, in PL/I programs, to rows rather than columns.
   g. Issue as few GETMAIN commands as possible.
   h. In COBOL programs, avoid using EXAMINE or VARIABLE MOVE operations, because these expand into subroutine executions.
2. To minimize the size of the working set, the amount of storage that a program refers to in a given period should be as small as possible.
   a. Write modular programs and structure 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.
   b. Use separate subprograms whenever the flow of the program suggests that execution will not be sequential.
   c. Do not tie up main storage while awaiting a reply from a terminal user.
   d. Use command-level file control locate-mode input/output rather than move-mode.
   e. In COBOL programs, specify constants directly, rather than as data variables in the working-storage section.
   f. In PL/I programs, use static storage for constant data.
   g. Avoid using LINK commands where possible, because they generate requests for main storage.
3. To improve validity of reference, the correct page should be determined directly.
   a. Avoid long searches for data.
   b. Use data structures that can be addressed directly, such as arrays, rather than structures that must be searched, such as chains.
   c. Avoid indirect addressing and any methods that simulate indirect addressing.

Do not use overlays (paging techniques) in an application program. System paging is provided automatically and has superior performance. The design of an application program for a virtual-storage environment is similar to that for a real environment. The system should have all modules resident so that code on unreferenced pages need not be paged in.

If the program is dynamic, the entire program must be loaded across adjacent pages before execution begins. Dynamic programs can be purged from storage if they are not in use and an unsatisfied storage request exists. Allowing sufficient dynamic area to prevent purging is more expensive than making them resident, because a dynamic program will not share unused space on a page with another program.

If you program in assembler language, the program mask is undefined to CICS on entry to an application program. You must set the program mask for any module that requires a specific value for the mask. CICS does not preserve the mask value across the interface to other called programs, for example, when a LINK or XCTL command is used.

CICS macros used with CICS commands

Take care when writing application programs that contain a mixture of CICS commands and CICS macros, or in a macro-level program that invokes a command-level program and vice-versa.

Avoid mixtures of commands and macros whenever you can. Any program containing a mixture of CICS commands and CICS macros, and which is link-edited with the DFHEAI stub, is a command-level program and should return to the program that linked to it by using an EXEC CICS RETURN command.

Using commands and macros that use the same component of CICS will often give wrong results. For example, using a mixture of BMS commands and DFHTC macros is likely to cause an error in TIOA usage.

When a RECEIVE MAP command is used with the SET option, the EXEC interface program always reuses the terminal input/output area (TIOA) obtained. This TIOA should be released in the command level program only. Do not use a DFHSC TYPE=FREEMAIN, RELEASE=ALL macro in the same or an invoked program, because the TIOA is freed unknown to the EXEC interface program, which will attempt to reuse it, giving unpredictable results.

Lengths of areas passed to CICS commands

When a CICS command includes a LENGTH operand, it usually accepts the length as a signed halfword binary value. This places a theoretical upper limit of 32K bytes on the length. In practice, the limits are less than this and vary for each command. The limits depend on data set definitions, recoverability requirements, buffer sizes, and local networking characteristics.

COMMAREA: The maximum size of a dynamic log record is 32K bytes. If a COMMAREA is passed to a transaction defined with the DFHPCT operand RESTART=YES, CICS journal control adds a 5-byte header before writing the COMMAREA contents to the dynamic log. Therefore, in this case, the practical limit to the COMMAREA size is 32K bytes minus 5 bytes.

For a COMMAREA passed between successive transactions in a pseudoconversational sequence, 32K bytes is the limit imposed by VTAM on the total data length. This limit applies to the entire transmitted package, which includes control data added by VTAM. The amount of control data increases if the transmission uses intermediate links.

Journal records: For journal records, the journal buffer size may impose a limit lower than 24K bytes. For journal records, the limit applies to the sum of the LENGTH and PFXLENG values.

Data set definitions: For temporary storage, transient data, and file control, the data set definitions can impose limits lower than 24K bytes. For details, see the *CICS/MVS Operations Guide* and the *CICS/MVS Resource Definition (Macro)* manual.

Recommendation: *For any command in any system,* **24K** *bytes is a good working limit for* LENGTH *specifications.* Subject to user-specified record and buffer sizes, this limit is unlikely either to cause an error or to place a constraint on applications.

Note: The value in the LENGTH operand should never exceed the length of the data area addressed by the command.

Program size

The load module resulting from any application program must not occupy more than 524,152 bytes of main storage, except that an RMODE=ANY program on MVS/XA* can be up to 16 megabytes in length (although this is not recommended).

BMS map size

The load module of a BMS map that is loaded dynamically using the LOAD command must not exceed 65,520 bytes in size.

The EXEC interface stubs

Each application program you write must contain an interface to CICS. This takes the form of an EXEC interface stub, used by the CICS high level programming interface. The stub must be link-edited with your application program to provide communication between your code and the EXEC interface program (DFHEIP).

Note: COBOL and PL/I application programs cannot be linked together.

There are stubs for each programming language.

Assembler: Each EXEC command is translated into an invocation of the DFHECALL macro by the command translator, and the external entry point invoked by DFHECALL is resolved to an entry in the stub.

COBOL and PL/I: Each EXEC command is translated into a COBOL or PL/I CALL statement (as appropriate) by the command translator. The external entry point invoked by the CALL statement is resolved to an entry in the stub.

The VS COBOL II command level interface has an assembler stub in the VS COBOL II library. Similarly, a PL/I application program must include a PL/I-supplied stub as well as the EXEC interface stub. This stub will be included by automatic library call.

See the *CICS/MVS Operations Guide* for more details.

Assembler-language considerations

Restrictions

The following instructions cannot be used in an assembler-language program that is to be used as a CICS application program:

- COM (identify blank common control section)
- ICTL (input format control)
- OPSYN (equate operation code).

MVS/XA restrictions

The following restrictions apply to an assembler-language application program executing in 31-bit mode (that is, a program written to use the extended addressing capabilities of processors under MVS/XA).

- DL/I DFHFC requests are not supported when running in 31-bit mode.

* IBM Trademark. For a list of trademarks see page iii.

- BMS maps, map sets, and partition sets resident above the 16-megabyte line are not supported.
- The WAIT EVENT interval control command is not supported when the associated event control block (ECB) resides above the 16-megabyte line.
- The COMMAREA option is restricted in a mixed addressing mode transaction environment. For a description of the restriction, see "Chapter 4.4. Program control" on page 289.

Commands contained within macros and COPY code

Macros that generate commands, and COPY code that contains commands, must be translated and stored in the source library in translated form for later inclusion by the assembler.

Invoking assembler-language application programs by a call statement

Assembler-language application programs containing commands can be treated as separate CICS programs that have their own program processing table (PPT) entries and that can be invoked by assembler-language, COBOL, or PL/I application programs using LINK or XCTL commands (see "Chapter 4.4. Program control" on page 289).

However, because assembler-language application programs containing commands are invoked by a system standard call, they can also be invoked by a COBOL or PL/I CALL statement or by an assembler-language CALL macro. A single CICS application program with one PPT entry may consist of a module containing separate CSECTs linked together, although they may have been compiled or assembled separately.

Also, assembler-language application programs containing commands can be linked with other assembler-language programs, or with programs in one of the high-level languages (COBOL or PL/I), but with only one of these. When such an application program is linked with an assembler-language application program, the main program must be the one coded in the high-level language, and the PPT must specify that high-level language.

The main program must be the only one containing the CICS interface stub, and the link edit must be done so that this stub is the first CSECT in the load module.

Because assembler-language application programs containing commands are always passed the EIB and COMMAREA parameters when they are invoked, the CALL statement or macro must pass these two parameters followed, optionally, by other parameters.

An assembler-language application program that is called by another application program must be preceded by the DFHEIENT macro and followed by the DFHEIRET macro.

DL/I CALL interface: Normally, with MVS/XA, you link edit your Assembler programs with the AMODE(31) and RMODE(ANY) options so that they can be loaded and acquire working storage above the 16-megabyte line. However, if a program uses the CALL DL/I interface, the program can reside above the 16-megabyte line, although its call parameter list and the call parameters must reside below the 16-megabyte line.

COBOL considerations

The following considerations apply to OS/VS COBOL. For information about VS COBOL II, which simplifies the CICS-COBOL interface, see "VS COBOL II considerations" on page 24.

As well as passing control to other programs by means of LINK and XCTL commands, a CICS COBOL program can invoke another COBOL or assembler program with a standard COBOL CALL statement. However, the following considerations may count against the use of a standard COBOL CALL:

1. A called program remains in its last-used state after it returns control, so a second CALL finds the program in this state. LINK and XCTL commands, on the other hand, always find the 'new' program in its initial state.
2. When you use the COBOL static CALL, you must link-edit the calling and called programs together and present them to CICS as a single unit, with one name and one entry in the PPT.

Restrictions

The following restrictions apply to a COBOL program that is to be used as a CICS application program. (See the appropriate COBOL programmer's guide for more information about these functions.)

1. You cannot use the entries in the environment division and data division that are normally associated with data management. However, you still need to code the headers for both of these divisions.
2. You cannot use the file section of the data division.
3. You cannot use these special options:

   REPORT WRITER
   SEGMENTATION
   SORT
   TRACE

4. You cannot use compiler options that require the use of operating system services:

   COUNT
   DYNAM
   ENDJOB
   FLOW
   STATE
   SYMDUMP
   SYST
   TEST

5. You cannot use the following COBOL statements that require the use of operating system services:

   ACCEPT
   CURRENT-DATE
   DATE
   DAY
   DISPLAY
   EXHIBIT
   INSPECT
   SIGN IS SEPARATE
   STOP RUN[1]
   STRING
   TIME
   UNSTRING

   [1] A COBOL GOBACK is required to satisfy the compiler's need for a logical "end of program." (The translator expands all CICS commands to COBOL CALLs, so the compiler expects a return to the calling program. Control *actually* returns to CICS after the RETURN command.)

6. Do not use the following COBOL statements:

   CLOSE
   OPEN
   READ
   WRITE

   because you are provided with CICS commands for the storage and retrieval of data, and for communication with terminals.

7. When you link edit separate COBOL routines together, only the first can invoke CICS or DL/I.

8. The length of working storage plus the length of the TGT (task global table), plus the length of the RSA (60 bytes), plus 24 bytes for SCP accounting, must not exceed 64K bytes.

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

   ```
   DFHEIVAR
   DFHEIBLK
   DFHCOMMAREA
   PROCEDURE DIVISION USING
      DFHEIBLK DFHCOMMAREA
      .
      .
   CICS commands
      .
      .
   ```

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

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

10. Statements that produce variable length areas, such as OCCURS DEPENDING ON, cannot be used within the working-storage section.

Compilers supported

You can use only the following COBOL compilers to process your COBOL application programs:

- OS Full COBOL Version 4 Compiler (5734-CB2).
- OS/VS COBOL Compiler (5740-CB1).
- VS COBOL II Compiler (5668-958).

Base locator for linkage (BLL)

The BLL mechanism is used to address storage outside the working-storage section of an application program. It operates by addressing the storage as if it is a parameter to the program. The storage must be defined by means of an 01-level data definition in the linkage section of the program. The COBOL compiler generates code to address the storage via the parameter list. When the program is invoked, CICS sets up the parameter list in such a way that the parameter list is itself addressable by the application program.

The parameter list must be defined as the first parameter to the program unless a communication area is being passed to the program, in which case the DFHCOMMAREA definition must precede it. (See "Passing data to other programs" on page 292.)

In the following example, the first 02-level data name (that is, FILLER) is set up by CICS to provide addressability to the other fields in the parameter list. The other data names are known as BLL cells, and address the remaining parameters of the program. There is a one-to-one correspondence between the 02-level data names of the parameter list definition and the 01-level data definitions in the linkage section.

```
LINKAGE SECTION.
01 PARMLIST.
   02 FILLER PIC S9(8) COMP.
   02 A-POINTER PIC S9(8) COMP.
   02 B-POINTER PIC S9(8) COMP.
   02 C-POINTER PIC S9(8) COMP.
01 A-DATA.
   02 PARTNO PIC 9(4).
   02 QUANTITY PIC 9(4) .
   02 DESCRIPTION PIC X(100).
01 B-DATA PIC X.
01 C-DATA PIC X.
```

In this example, A-POINTER addresses A-DATA, B-POINTER addresses B-DATA, and C-POINTER addresses C-DATA. The data names chosen for the BLL cells and for the storage areas that they address are not significant, but the names must be defined in the correct order so that the necessary correspondence is established.

If a BLL cell is named in the SET option of a CICS command, subsequent reference to the corresponding data definition name will address the storage supplied by CICS as a result of executing the command. For example, suppose that a program is required to read a variable-length record from a file, examine part of it, and update it; all of this is to be done without providing storage for the record within the program. Using the data definitions shown in the example above, the program could be written as follows:

```
EXEC CICS READ UPDATE DATASET('FILEA')
     RIDFLD(PART-REQD) SET(A-POINTER)
     LENGTH(A-LRECL) END-EXEC
IF A-LRECL LESS THAN 8 GO TO ERRORS.
IF QUANTITY GREATER ZERO
     SUBTRACT 1 FROM QUANTITY
     EXEC CICS REWRITE DATASET('FILEA')
          FROM(A-DATA) LENGTH(A-LRECL)
          END-EXEC.
```

CICS reads the record into an internal buffer and supplies the address of the record in the buffer to the application program. The application program updates the record in the buffer and rewrites the record to the data set.

If a storage area is defined in the linkage section, addressability to that area must be established before use. The area itself must be a valid and active CICS area.

BLL and chained storage areas: If access is required to a series of chained storage areas (that is, areas each of which contain a pointer to the next area in the chain), a paragraph name must be inserted immediately following any statement that establishes addressability to one of the storage areas. For example:

```
LINKAGE SECTION.
01 PARMLIST.
   .
   .
   02 USERPTR PIC S9(8) COMP.
   .
   .
01 USERAREA.
   02 FIELD PIC X(4).
   02 NEXTAREA PIC S9(8) COMP.
   .
   .
PROCEDURE DIVISION.
   .
   .
   MOVE NEXTAREA TO USERPTR.
ANYNAME.
   MOVE FIELD TO TESTVAL.
   .
   .
```

In this example, storage areas mapped or defined by USERAREA are chained. The first MOVE statement establishes addressability to the next area in the chain. The second MOVE statement moves data from the newly addressed area, but only because a paragraph name follows the first MOVE statement. If no paragraph name is inserted, the reference to FIELD is taken as being to the storage area that is addressed when the first MOVE statement refers to NEXTAREA. Insertion of a paragraph name causes the compiler to generate code to reestablish addressability through USERPTR, so that the reference to FIELD (and the next reference to NEXTAREA) is to the newly addressed storage area.

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

```
LINKAGE SECTION.
   .
   .
01 FILE-REC.
   .
   .
   02 FIELD-COUNTER PIC 9(4) COMP.
   02 FIELDS PIC X(5) OCCURS 1 TO 5
      TIMES DEPENDING ON FIELD-COUNTER.
   02 DATA PIC X(20).
   .
   .
```

```
PROCEDURE DIVISION.
     .
     .
    EXEC CICS READ DATASET('FILEA')
                   RIDFLD(KEYVAL)
                   SET(RECPTR)
                   END-EXEC.
    MOVE FIELD-COUNTER TO FIELD-COUNTER.
    MOVE DATA TO DATA-VAL.
     .
     .
```

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

BLL and large storage areas: If an area greater than 4096 bytes is defined (but not as part of an OCCURS DEPENDING ON clause) in the linkage section, additional statements may be required to establish addressability to the extra area. The ADD statement is placed after the statement that establishes addressability to the data area. No additional corresponding 01-level data name definition is added, so the usual one-to-one correspondence of BLL cells to the data areas is not maintained.

The extra statements are shown in the following example:

```
LINKAGE SECTION.
01 PARMLIST.
     .
     .
   02 FRPTR PIC S9(8) COMP.
   02 FRPTR1 PIC S9(8) COMP.
     .
     .
01 FILE-REC.
   02 FIELD1 PIC X(4000).
   02 FIELD2 PIC X(1000).
   02 FIELD3 PIC X(400).
PROCEDURE DIVISION.
     .
     .
    EXEC CICS READ DATASET('FILEA')
                   RIDFLD(KEYVAL)
                   SET(FRPTR) END-EXEC.
    ADD 4096 FRPTR GIVING FRPTR1.
```

No additional BLL cell is required if DFHCOMMAREA itself is larger than 4096 bytes.

SERVICE RELOAD statement: If an application program is to be compiled using the OS Full COBOL Version 4 compiler or the OS/VS COBOL compiler, a special compiler control statement must be inserted at appropriate places within the program to ensure addressability to a particular area defined in the linkage section. This control statement has the form:

```
SERVICE RELOAD fieldname
```

where 'fieldname' is the symbolic name of a specific storage area that is also defined in an 01-level statement in the linkage section. The SERVICE RELOAD statement must be used following each statement that modifies addressability to an area defined in the linkage section, that is, whenever the contents of a BLL cell is changed in any way.

When using HANDLE CONDITION or HANDLE AID commands, SERVICE RELOAD statements should be specified at the start of the paragraph whose name is specified in the HANDLE command. This applies to all those BLL cells that may have been altered from the time when the first HANDLE command activated the exit routine, up to and including any CICS command that can cause the HANDLE exit to be invoked.

If the BLL mechanism (described earlier in this chapter) is used, addressability to the parameter list must be established at the start of the procedure division. This is done by adding a SERVICE RELOAD PARMLIST statement at the start of the procedure division in the earlier examples.

For example, after a locate-mode input operation, the SERVICE RELOAD statement must be issued to establish addressability to the data. If areas larger than 4096 bytes are being addressed, the secondary BLL cells must first be reset before the SERVICE RELOAD statement is executed (resetting a BLL cell is described in the previous section). If an address is moved into a BLL cell, addressability must be established in the same way.

An example for the SERVICE RELOAD statement is as follows:

```
LINKAGE SECTION.
01 PARMLIST.
    .
    .
    02 FRPTR    PIC S9(8)    COMP.
    02 FRPTR1   PIC S9(8)    COMP.
    02 TSPTR    PIC S9(8)    COMP.
01 FILE-REC.
    02 FIELD1 PIC X(4000).
    02 FIELD2 PIC X(1000).
    02 FIELD3 PIX X(400).
01 TS-REC.
    02 FIELD1 PIX X(4000).

PROCEDURE DIVISION.
    .
    .
    EXEC CICS HANDLE CONDITION
              ERROR(GIVEUP)
              LENGERR(BADLENGTH)
              END-EXEC.
    EXEC CICS READ DATASET('FILEA')
              RIDFLD(PART-REQD)
              SET(FRPTR)
              LENGTH(A-LRECL)
              END-EXEC.

    ADD 4096 FRPTR GIVING FRPTR1.
    SERVICE RELOAD FILE-REC.
    MOVE FRPTR TO TSPTR.
    SERVICE RELOAD TS-REC.
    .
    .
BADLENG.
    ADD 4096 FRPTR GIVING FRPTR1.
    SERVICE RELOAD FILE-REC.
    .
    .
```

If an address is moved into a BLL cell, addressability must be established in the same way, for example:

```
MOVE B-POINTER TO A-POINTER SERVICE RELOAD A-DATA.
```

NOTRUNC compiler option

If an argument to a command is greater than 9999 in value, the NOTRUNC compiler option must be specified to ensure successful execution.

Program segments

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.

Subsequent copying or manipulating of statements originally inserted by the CICS translator in an application program may produce unpredictable results.

VS COBOL II considerations

VS COBOL II has a simpler CICS command-level interface than OS/VS COBOL. You can compile OS/VS COBOL programs with the VS COBOL II compiler, but there are some restrictions (see the *VS COBOL II General Information* manual, GC26-4042 and the *VS COBOL II Application Programming Guide*, SC26-4045). For suggested changes to OS/VS COBOL programs (only one of which is essential) see "Converting to VS COBOL II" on page 29.

VS COBOL II is a licensed program (number 5668-958) that conforms to the standard set by American National Standard COBOL, X3.23-1974.

All CICS VS COBOL II application programs must be written using the CICS command level interface. A program that issues a macro call will produce compiler diagnostics if it is compiled using the VS COBOL II compiler. (You may be able to compile such a program using the preprocessor and assembler before the compile step, but it will then fail at execution time, causing a transaction abend.)

When you translate a VS COBOL II program, you must use the COBOL2 translator option. For information about translating, compiling, and link-editing VS COBOL II programs, see the *CICS/MVS Installation Guide*.

Coding

With VS COBOL II, you must use only CICS commands to invoke operating system services. You must not use CICS macros or some COBOL statements. These statements are listed under "Restrictions" on page 29.

The principal advantages of VS COBOL II over OS/VS COBOL for the CICS programmer are:

- Simplified based addressing, with the replacement of BLL cell manipulation by pointer variables and the ADDRESS special register.
- The ability to use COBOL CALL statements to call assembler-language and other VS COBOL II programs.
- The provision of a LENGTH special register, which CICS uses to deduce the length of data items. You no

longer need to specify a length in a CICS command that specifies a data name in the FROM option, or, if the data is fixed length, in the INTO option.

- The ability to use the RETURN-CODE special register in a CICS application program. This register allows you to set and access return codes in VS COBOL II programs.
- The elimination of the SERVICE RELOAD statement. If it is included, it is ignored. The VS COBOL II compiler recognizes references to based addresses and generates code to manipulate them (even when the OPTIMIZE compiler option is used).

Based addressing

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

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

As well as defining the data area in its linkage section, an OS/VS COBOL program that needs to access dynamically addressed data must define and set internal COBOL areas called BLL (base locator for linkage) cells. The order of the data area definitions must be that of the corresponding BLL cell definitions. The program may also need extra statements (for example, SERVICE RELOAD) to ensure correct addressing (see the description starting at "Base locator for linkage (BLL)" on page 21).

VS COBOL II provides a much simpler method, using pointer variables and the ADDRESS special register. Figure 3 on page 26 gives a comparison between the OS/VS COBOL and the VS COBOL II methods.

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 CICS GETMAIN, LOAD, READ, and READQ commands. A pointer variable is defined with the USAGE IS POINTER clause, to handle chained lists.

In VS COBOL II, you can use the SET statement to assign the address of a linkage section area to a pointer variable, or, conversely, the value of a pointer variable to the ADDRESS special register of a linkage section area. The latter has the effect of positioning the linkage section area at the address held in the pointer variable.

For example, the statement

```
SET PTRX TO ADDRESS OF STRUCA
```

assigns the current value of the ADDRESS special register of STRUCA to the pointer variable PTRX, while the statement

```
SET ADDRESS OF STRUCA TO PTRX
```

assigns the current value of the PTRX pointer variable to the ADDRESS special register of STRUCA; that is, it positions STRUCA at the address in PTRX. (As with any data field, you must initialize a pointer or ADDRESS special register before you assign its value to another field.)

Wherever in this book a CICS command description includes 'ptr-ref', you can use as the 'ptr-ref' a VS COBOL II pointer variable or ADDRESS special register. For example, you can specify the SET option of the CICS GETMAIN, LOAD, READ, READQ, RECEIVE, or SEND commands as follows:

```
SET(ADDRESS OF STRUCA) ....    SET(PTRX) ....
```

If you use the second form of the SET option, you can use the SET statement to move the pointer value, PTRX, to the ADDRESS special register of any linkage section area. Then you can use that area in the same way as STRUCA in the first form of the option.

This technique has other advantages over the OS/VS COBOL method. You do not need to set additional address variables when a data area exceeds 4096 bytes and, with the elimination of BLL cell definitions, you can arrange your linkage-section area definitions in any order you choose.

For examples of the use of the ADDRESS command, the ADDRESS special register, and the SET statement, see Figure 4 on page 27.
This figure shows how an application can use a common table of tax codes stored in the CICS program library. Program 1 sets up addressability to the CWA, loads the table and places its address in the CWA for use by subsequent programs or other transactions. Because addressability is established, you can process the table in this program.

The HOLD option is coded in the CICS LOAD command so that the table TAXTAB remains loaded at the same storage address after the current transaction terminates.

Program 2 shows how you can use the address in the CWA to address the tax table from another program.

Figure 3 on page 26 shows the use of a CICS READ command with the SET option as it is used in OS/VS COBOL and VS COBOL II. The record read is variable-length. The OS/VS COBOL version has extra statements to:

- Define the BLL cells,
- Compute the address of data beyond the first 4096 bytes of REC-1, and

```
OS/VS COBOL                         | VS COBOL II
                                    |
WORKING-STORAGE SECTION.            | WORKING-STORAGE SECTION.
77 LRECL-REC1    PIC S9(4) COMP.    | 77 LRECL-REC1    PIC S9(4) COMP.
LINKAGE SECTION.                    | LINKAGE SECTION.
01 BLLCELLS.                        |
   02 FILLER     PIC S9(8) COMP.    |
   02 BLL-REC1A  PIC S9(8) COMP.    |
   02 BLL-REC1B  PIC S9(8) COMP.    |
   02 BLL-REC2   PIC S9(8) COMP.    |
                                    |
01 REC-1.                           | 01 REC-1.
   02 FLAG1      PIC  X(1).         |    02 FLAG1      PIC  X(1).
   02 MAIN-DATA  PIC  X(5000).      |    02 MAIN-DATA  PIC  X(5000).
   02 OPTL-DATA  PIC  X(1000).      |    02 OPTL-DATA  PIC  X(1000).
                                    |
01 REC-2.                           | 01 REC-2.
   02 ...                           |    02 ...
                                    |
PROCEDURE DIVISION.                 | PROCEDURE DIVISION.
                                    |
   EXEC CICS READ UPDATE...         |    EXEC CICS READ UPDATE...
       SET(BLL-REC1A)               |        SET(ADDRESS OF REC-1)
       LENGTH(LRECL-REC1)           |        LENGTH(LRECL-REC1)
       END-EXEC.                    |        END-EXEC.
                                    |
                                    |
   ADD 4096, BLL-REC1A              |
   GIVING BLL-REC1B.                |
   SERVICE RELOAD REC-1.            |
                                    |
                                    |
   IF FLAG1 EQUAL 'Y'               |    IF FLAG1 EQUAL 'Y'
   MOVE OPTL-DATA TO ...            |    MOVE OPTL-DATA TO ...
         .                          |          .
                                    |
                                    |
   EXEC CICS REWRITE...             |    EXEC CICS REWRITE...
       FROM(REC-1)                  |        FROM(REC-1)
       LENGTH(LRECL-REC1)           |        END-EXEC.
       END-EXEC.                    |
```

Figure 3. Addressing CICS data areas in locate mode. This figure shows the replacement of BLL cells and SERVICE RELOAD in OS/VS COBOL by the use of ADDRESS special registers in VS COBOL II. If the records in the READ or REWRITE commands are fixed length, VS COBOL II does not require a LENGTH option. This example assumes variable-length records. After the read, you can get the length of the record from the field named in the LENGTH option, (here, `LRECL-REC1`*). In the REWRITE command, you must code a LENGTH option if you want to replace the updated record with a record of a different length.*

- Ensure addressability following a statement that modifies a BLL cell.

LENGTH option in CICS commands: CICS uses the LENGTH special register to determine the length of a referenced data item. Therefore, you no longer need to code the following options on CICS commands that name a data item in a FROM or INTO option.

`LENGTH, FLENGTH, FROMLENGTH, MAXLENGTH, MAXFLENGTH, DESTIDLENG, VOLUMELENG, FIELDLENGTH (EXEC DLI only), SEGLENGTH (EXEC DLI only)`

If you want the program to read or write data of a length different from that of the referenced variable, you must still code the LENGTH option. Otherwise, you need not.

Figure 3 shows a CICS REWRITE command without a LENGTH option. The length of the REC-1 data field is implied. If you use the execution diagnostic facility (EDF) to look at the command, you will see the length that has been adopted by default.

```
                    Placing Address of External Area in CWA - Program 1
LINKAGE SECTION.
01  COMMON-WA.
    02 TAX-TABLE-ADDRESS USAGE IS POINTER.
    02...
01  TAXTAB.
    02 TAX-FLDA PIC 99.
    02 ................
PROCEDURE DIVISION.
* SET UP ADDRESSABILITY TO THE CWA
    EXEC CICS ADDRESS CWA(ADDRESS OF COMMON-WA) END-EXEC.
* LOAD EXTERNAL TABLE 'TAXTAB' PLACING ITS ADDRESS IN CWA
    EXEC CICS LOAD PROGRAM('TAXTAB') SET(TAX-TABLE-ADDRESS) HOLD END-EXEC.
* THE ABOVE IS SUFFICIENT TO ENABLE OTHER PROGRAMS/TRANSACTIONS TO
* ADDRESS THE EXTERNAL TABLE
*
* TO PROCESS THE EXTERNAL TABLE IN THIS PROGRAM, PLACE ITS ADDRESS IN
* ADDRESS REGISTER OF THE 01 DEFINITION IN LINKAGE SECTION
    SET ADDRESS OF TAXTAB TO TAX-TABLE-ADDRESS.
* PROCESS TAXTAB, IF DESIRED, USING 01 DEFINITION IN LINKAGE SECTION
    ..........
    MOVE TAX-FLDA TO ..............
    ..........

            Use of the Global Table in Another Transaction  - Program 2
LINKAGE SECTION.
01  COMMON-WA.
    02 TAX-TABLE-ADDRESS USAGE IS POINTER.
    02...
01  TAXTAB.
    02 TAX-FLDA PIC 99.
    02 ................
PROCEDURE DIVISION.
* SET UP ADDRESSABILITY TO THE CWA, AND THEN TO TAXTAB
    EXEC CICS ADDRESS CWA(ADDRESS OF COMMON-WA) END-EXEC.
    SET ADDRESS OF TAXTAB TO TAX-TABLE-ADDRESS.
* NOW PROCESS TAXTAB USING 01 DEFINITION IN LINKAGE SECTION
    ..........
    MOVE TAX-FLDA TO ..............
    ..........
```

Figure 4. Addressing external storage

RETURN-CODE special register: Because CICS supports calls to VS COBOL II programs, you can use the RETURN-CODE special register in a called program to set a return code before issuing an EXIT PROGRAM or GOBACK statement. If control is returned to a calling VS COBOL II program, that program can access the return code in the RETURN-CODE special register.

In a called assembler program, you can set a return code in register 15. In the calling VS COBOL II program, you can access the return code in the RETURN-CODE special register.

CICS commands change the RETURN-CODE special register to an undefined value. Therefore:

- In the calling program, access the RETURN-CODE special register on return from a called program **before** you issue any CICS commands.
- In the called program, do not issue any CICS commands **after** you have set the RETURN-CODE special register.

Restrictions lifted: For VS COBOL II, CICS supports the use of some statements that are not supported for OS/VS COBOL. These are:

INSPECT
STOP RUN
STRING
UNSTRING
USE FOR DEBUGGING
DATE
TIME

Returning from VS COBOL II program: Some "return" operations are prohibited or cause errors with OS/VS COBOL under CICS. With VS COBOL II, the following are permitted under CICS:

GOBACK
Returns control to another COBOL program or to CICS.

EXIT PROGRAM
Issued within a program invoked by COBOL CALL. Returns control from a COBOL subprogram to a COBOL main program.

EXIT PROGRAM
Issued within program invoked directly by CICS, or by CICS LINK or XCTL. Ignore. Control is not returned to CICS.

STOP RUN
Terminate run unit normally and return control to CICS.

EXEC CICS XCTL
Terminate run unit normally.

EXEC CICS RETURN
Terminate run unit normally.

EXEC CICS ABEND
Abend the transaction.

In a VS COBOL II program running under CICS, if (due to a program error) control passes beyond the last statement of the program, VS COBOL II detects the condition, writes an exception message (IGZ037I), and issues the command:

```
EXEC CICS ABEND ABCODE(1037) NOHANDLE.
```

(In OS/VS COBOL, if control is not transferred before program-end, the result is an undefined error condition.)

CALL statement from VS COBOL II program: CICS allows your VS COBOL II programs to use the CALL statement to call other programs. With the CALL statement, you can invoke:

- A VS COBOL II or assembler program link-edited into the same load module as the calling program. This is known as a **static call**.
- A separate VS COBOL II load module that does not contain CICS commands or CICS dependencies. This is known as a **dynamic call**.

A dynamically called VS COBOL II program can reside in the link pack area or in a shared library. It can also be shared with non-CICS regions.

If a program, invoked by a static CALL, is processed by the CICS translator, the CALL statement must pass the addresses of DFHEIBLK and DFHCOMMAREA as the first two parameters, whether or not any other parameters are passed. The call has the following form:

```
CALL 'PROG' USING DFHEIBLK DFHCOMMAREA
PARM1 PARM2...
```

In the called program PROG, the CICS translator inserts DFHEIBLK and DFHCOMMAREA into the linkage section and into the USING list of the procedure division statement. You code the procedure division statement normally, as follows:

```
PROCEDURE DIVISION USING PARM1 PARM2...
```

The translator inserts DFHEIBLK and DFHCOMMAREA into this statement before PARM1.

Notes:

1. You must always use the NODYNAM compiler option (the default) when you compile a VS COBOL II program that is to run with CICS, even if the program issues dynamic calls.
2. In a CICS environment, you cannot call a VS COBOL II program from an assembler program.
3. As in a non-CICS environment, you use the 'CALL literal' and 'CALL IDENT' forms of the CALL statement for static and dynamic calls respectively.
4. In a VS COBOL II program, you must use the CICS LINK or XCTL commands to invoke load modules containing programs of any language (including other versions of COBOL).

Debugging facilities: You can use the CICS execution diagnostic facility (EDF) with VS COBOL II. You use EDF by invoking the CICS-supplied CEDF transaction. The transaction under test can then be interrupted at defined points. While the transaction is halted, you can obtain a great deal of information about the state of the transaction, and can also examine any area in the CICS region.

Under EDF, key PF5 gives you a display of working storage, which includes displacements from the start of working storage. If you use the VS COBOL II compiler MAP option, you can locate any item in the display. In the compiler map, find the BLW number (B) and displacement (D) of the item. The displacement of the item from the start of working storage is:

```
(B * X'1000') + D
```

When you examine a dump of the CICS region after a transaction abend, you may see more commands than you expect recorded in the CICS trace table. This is because the VS COBOL II library routines use the CICS command level interface. (When you use EDF, you do not see CICS commands issued by the library routines, because these routines are translated with the NOEDF option.)

If you compile your program with the FDUMP compiler option, abnormal termination at execution time produces a formatted dump of key program information. This dump is written to the temporary storage queue, CEBRxxxx (where xxxx is the TERMID of the terminal associated with the failing transaction). The use of the CICS HANDLE ABEND command suppresses FDUMP.

VS COBOL II execution-time messages are also written to CEBRxxxx.

You can use the CEBR CICS transaction to browse this queue. When you are using terminal xxxx, CEBRxxxx is the default queue-name for CEBR.

Restrictions

This section describes VS COBOL II language elements that you cannot use under CICS, or whose use is restricted or can cause problems under CICS.

In general, neither the CICS translator nor the VS COBOL II compiler detects the use of COBOL words affected by the following restrictions. The use of a restricted word in a CICS environment may cause a failure at execution time.

So avoid using variable names such as 'CICS' or 'EXEC' in your VS COBOL II programs.

Requests for operating system services: Do not use VS COBOL II statements that invoke operating system functions. Instead, use CICS commands. Do not use the following statements:

ACCEPT
CALL 'identifier'
CALL 'literal' with DYNAM
CLOSE
DELETE
DISPLAY
MERGE
OPEN
READ
RERUN
REWRITE
SORT
START
STOP 'literal'
WRITE

Other facilities: Do not use:

- USE declaratives (except USE FOR DEBUGGING)
- ENVIRONMENT DIVISION and FILE SECTION entries associated with data management, because CICS handles data management
- User-specified parameters to the main program.

Compiler options: Do not use the following compiler options:

DYNAM
GRAPHIC
NOLIB (if program is to be translated)
NORENT
NORES
TRUNC
NOTRUNC is not recommended either

The following compiler options have no effect in a CICS environment:

ADV
FASTSRT
OUTDD

Converting to VS COBOL II

Many of the changes in the CICS-COBOL interface occur because VS COBOL II simplifies the procedures. This means that you do not need to use some CICS-specific OS/VS COBOL programming techniques. Of the changes described in this section, the only one that is mandatory is the replacement (removal) of all PROCEDURE DIVISION references to BLL cells.

Based addressing: You no longer need to define and manipulate BLL cells. Indeed, you cannot manipulate BLL cells for base address manipulation, as in the management of chained lists. Review programs that use the CICS SET option and BLL cells, and make the following changes:

1. Remove, from the linkage section, the entire structure defining BLL cells and the FILLER field. See Figure 3 on page 26.
2. Revise code that deals with chained storage areas to take advantage of the ADDRESS special register and POINTER variables.
3. Change every SET(BLL cell) option in CICS commands to SET(ADDRESS OF A-DATA) or SET(A-POINTER) where A-DATA is a structure in the linkage section and A-POINTER is defined with the USAGE IS POINTER clause.
4. Remove all SERVICE RELOAD statements.
5. Remove all program statements required in OS/VS COBOL to address structures in the linkage section longer than 4K bytes. A typical statement is:
   ```
   ADD 4096, D-PTR1 GIVING D-PTR2.
   ```
6. Remove artificial paragraph names where BLL cells are used to address chained storage areas (see "BLL and chained storage areas" on page 22).
7. Review any program that uses BMS map data structures in its linkage section. VS COBOL II makes it easier to handle such maps, but it also eliminates one disadvantage of having maps in working storage. The points to consider are:
   a. In OS/VS COBOL programs, working storage is part of the compiled and saved program. Placing the maps in the linkage section thus reduces the size of the saved program, saving library space. In VS COBOL II, working storage is not part of the compiled program but is acquired dynamically. This eliminates one disadvantage of placing maps in working storage.

b. If your map is in the linkage section, you can acquire and release the map storage dynamically with CICS GETMAIN and FREEMAIN commands. This helps you to optimize storage use, and can be useful in a long conversational transaction. This advantage of linkage section maps still applies in VS COBOL II.

c. If your map is in the linkage section, you must issue a CICS GETMAIN command to acquire storage for the map. With OS/VS COBOL, 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 VS COBOL II, use the LENGTH special register:

```
EXEC CICS GETMAIN
     SET(ADDRESS OF DATAREA)
     LENGTH(LENGTH OF DATAREA)
```

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

Figure 5 shows the old and new methods of processing BMS maps in the linkage section. This example assumes that the OS/VS COBOL program has been compiled with the LANGLVL(1) option, and that the following map set has been installed:

```
MAPSET1 DFHMSD TYPE=DSECT,
               TERM=2780,LANG=COBOL,
               STORAGE=AUTO,
               MODE=IN
```

```
OS/VS COBOL                              | VS COBOL II
         .                               |          .
         .                               |          .
WORKING-STORAGE SECTION.                 |  WORKING-STORAGE SECTION.
77 FLD0 PIC X VALUE IS LOW-VALUE.        |  77 FLD0 PIC X VALUE IS LOW-VALUE.
                                         |
                                         |
LINKAGE SECTION.                         |  LINKAGE SECTION.
                                         |
01 BLLCELLS.                             |
   02 FILLER     PIC S9(8) COMP.         |
   02 BLL-DATAA PIC S9(8) COMP.          |
                                         |
01 DATA1 COPY MAPSET1.                   |  01 DATA1 COPY MAPSET1.
                                         |
PROCEDURE DIVISION.                      |  PROCEDURE DIVISION.
                                         |
   EXEC CICS GETMAIN LENGTH(1000)        |     EXEC CICS GETMAIN
   SET(BLL-DATAA) INITIMG(FLD0)          |     FLENGTH(LENGTH OF DATA1)
   END-EXEC.                             |     SET(ADDRESS OF DATA1)
                                         |     INITIMG(FLD0) END-EXEC.
```

Figure 5. Addressing BMS map sets in the linkage section

The new ADDRESS special register used in the example is described under "Based addressing" on page 25.

Artificial assignments: Remove artificial assignments from an OCCURS .. DEPENDING ON object to itself. These are needed in OS/VS COBOL to ensure addressability.

MVS/XA restrictions: These restrictions apply to a VS COBOL II program running above the 16-megabyte line:

- BMS maps, map sets, and partitions resident above the line are not supported.
- If the receiving program is link-edited with AMODE=31, addresses passed to it must be 31 bits long (or 24 bits long with the leftmost byte set to zeros).
- If the receiving program is link-edited with AMODE=24, addresses passed to it must be 24 bits long.

DL/I CALL interface: Normally, with MVS/XA, you link edit your VS COBOL II programs with the options AMODE(31) and RMODE(ANY), so that they can be loaded, and acquire working storage, above the 16-megabyte line. However, if a program uses the CALL DL/I interface, the program can reside above the 16-megabyte line, although its call parameter list and the call parameters must reside below the 16M byte line. You can ensure this by compiling with the DATA(24) option. DL/I DFHFC requests are not supported when running in 31-bit mode.

If you link edit a CALL DLI program with the AMODE(24) and RMODE(24) options, you can run it below the 16-megabyte line on a CICS system running with MVS/XA.

If you wish to continue using CALL DLI, review your programs and make the following changes:

1. Remove BLL cells for addressing the user interface block (UIB) and program control blocks (PCBs).
2. Retain the DLIUIB declaration and at least one PCB declaration in the linkage section.
3. Change the PCB call to specify the UIB directly, as follows:

   ```
   CALL 'CBLTDLI' USING PCB-CALL
                  PSB-NAME
                  ADDRESS OF DLIUIB.
   ```
4. Obtain the address of the required PCB from the address list in the UIB.

Figure 6 illustrates the whole of the above process. The example in the figure assumes that you want to use the second PCB in the address list. Therefore, when setting up the ADDRESS special register of the linkage section group item PCB, the program uses 2 to index the working storage table, PCB-ADDRESS-LIST. To use the nth PCB, you use the number n to index PCB-ADDRESS-LIST.

```
WORKING-STORAGE SECTION.
    77 PCB-CALL        PIC X(4) VALUE 'PCB '.
    77 GET-HOLD-UNIQUE PIC X(4) VALUE 'GHU '.
    77 PSB-NAME        PIC X(8) VALUE 'CBLPSB'.
    77 SSA1            PIC X(40) VALUE SPACES.
    01 DLI-IO-AREA.
       02 ............
       02 ............
       ................
LINKAGE SECTION.
    DLIUIB COPY DLIUIB.
    01 OVERLAY-DLIUIB REDEFINES DLIUIB.
       02  PCBADDR USAGE IS POINTER.
       02  FILLER       PIC XX.
    01 PCBADDR-LIST.
       02  PCBADDR-AREA PIC X(40).
       02  PCBADDR-ITEM REDEFINES PCBADDR-AREA USAGE IS POINTER
                        OCCURS 10 TIMES.
*DEFINE PCBADDR-LIST TO HAVE CHARACTERS EQUAL TO 4 TIMES THE NUMBER
*OF OCCURRENCES OF PCBADDR-ITEM
       02 PCB-DBD-NAME PIC X(8).
       02 PCB-SEG-LEVEL PIC XX.
       02 PCB-STATUS-CODE PIC XX.
       ..........................
PROCEDURE DIVISION.
*SCHEDULE THE PSB AND ADDRESS THE UIB
    CALL 'CBLTDLI' USING PCB-CALL PSB-NAME ADDRESS OF DLIUIB.
*CHECK SUCCESS OF CALL
    IF UIBFCTR IS NOT EQUAL LOW-VALUES THEN
                                          ...... error diagnostic code

*MOVE VALUE OF UIBPCBAL, ADDRESS OF PCB ADDRESS LIST (HELD IN UIB)
*(REDEFINED AS PCBADDR, A POINTER VARIABLE), TO
*ADDRESS SPECIAL REGISTER OF PCB-ADDRESS-LIST, DEFINED IN LINKAGE SECTION
    SET ADDRESS OF PCB-ADDRESS-LIST TO PCBADDR.
*MOVE VALUE OF SECOND ITEM IN PCB-ADDRESS-LIST TO ADDRESS SPECIAL
*REGISTER OF PCB, DEFINED IN LINKAGE SECTION.
    SET ADDRESS OF PCB TO PCB-ADDRESS-LIST(2).
*PERFORM DATABASE CALLS ......
    ........
    MOVE ........ TO SSA1.
    CALL 'CBLTDLI' USING GET-HOLD-UNIQUE PCB DLI-IO-AREA SSA1.
*CHECK SUCCESS OF CALLS .......
    IF UIBFCTR IS NOT EQUAL LOW-VALUES THEN
                                          ...... error diagnostic code
    ........
    IF PCB-STATUS-CODE IS NOT EQUAL SPACES THEN
                                          ...... error diagnostic code
    ........
```

Figure 6. Using the DL/I CALL interface

Mixing languages

A run unit is a running set of one or more programs that communicate with each other by VS COBOL II static CALL statements. In a CICS environment, a run unit is entered at the start of a CICS task, or invoked by a CICS LINK or XCTL command. A run unit can be defined as the execution of a single entry in the processing program table (PPT).

Because CICS does not support a mixture of language levels in a run unit, a VS COBOL II run unit can contain only:

- VS COBOL II programs compiled with the same compiler
- Assembler routines.

CICS supports only COBOL-to-COBOL and COBOL-to-assembler calls. CICS does **not** support the use of the macro-level interface by an assembler routine in a VS COBOL II run unit.

However, a CICS transaction can consist of many run units, each of which can be at a different language level. This means that a single transaction can consist of programs compiled by different compilers (including non-COBOL compilers), provided that programs compiled by different compilers communicate with each other only by using CICS LINK or XCTL commands.

Note: In general, when you call a VS COBOL II program, you must pass two special parameters if the called program has been processed by the CICS translator (see "CALL statement from VS COBOL II program" on page 28). This rule is varied for calls to nested programs in a unit of compilation translated with the ANSI85 option (see "Nesting — what the application programmer must do" on page 38).

Calling subprograms from VS COBOL II

In a CICS system, when control is transferred from the 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. There are three ways of transferring control to a subprogram:

EXEC CICS LINK

The calling program contains a CICS command in one of these forms:

```
EXEC CICS LINK PROGRAM('subpgname')
EXEC CICS LINK PROGRAM(name)
```

In the first form, the called subprogam is explicitly named as a literal string. In the second form, the literal is replaced by the name of a COBOL data area containing the name of the called subprogram.

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.

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.

Table 1 on page 33 gives the rules governing the use of the three ways to call a subprogram.

Table 1 on page 33 refers to CICS application logical levels, which are defined on page 289. Each LINK command creates a new logical level, the called program being at a level one lower than the level of the calling program (CICS is deemed to be at level 0). Figure 7 on page 35 shows logical levels and the effect of RETURN commands and CALL statements in linked-to and called programs.

Notes:

1. When control is passed by an EXEC CICS XCTL command, the program receiving control **cannot** return control to the calling program by an EXEC CICS RETURN command or a GOBACK statement, and is therefore not a subprogram.
2. In an ANSI85 unit of compilation, a called nested program is **internal** to the calling program, and is therefore not a subprogram.

Table 1 (Page 1 of 2). Calling a subprogram from VS COBOL II

<table>
<tr><th></th><th>EXEC CICS LINK</th><th>Static COBOL CALL</th><th>Dynamic COBOL CALL</th></tr>
<tr><td>Translation</td><td>The linked-to subprogram must be translated if it, or any subprogram invoked from it, contains CICS function.</td><td colspan="2">The called subprogram must be translated if it, or any subprogram invoked from it, contains CICS commands or references to the EXEC interface block (DFHEIBLK) or to the CICS communication area (DFHCOMMAREA).</td></tr>
<tr><td>Link-editing (See Note 2 on page 34.)</td><td>The linked-to subprogram must be compiled and link-edited as a separate program.</td><td>The called subprogram must be link-edited with the calling program to form a single load module (but the programs can be compiled separately). This can produce large program modules, and it also stops two programs that call the same program from sharing a copy of that program.</td><td>The called subprogram must be compiled and link-edited as a separate load module. Can reside in the link pack area or in a shared library. It can also be shared with non-CICS regions.</td></tr>
<tr><td rowspan="2">CSD entries</td><td rowspan="2">Both programs must be defined in the CSD if the linked-to subprogram is unknown or unavailable, the LINK fails with the PGMIDERR condition.</td><td colspan="2">The calling program must be defined in the CSD. An attempt to call the calling program results in a COBOL message and abend (1015).</td></tr>
<tr><td></td><td>The called subprogram must be defined in the CSD. If the called subprogram is unknown or unavailable, COBOL issues a message and abends (1029).</td></tr>
<tr><td>Return from called subprogram</td><td>The linked-to subprogram must return using EXEC CICS RETURN.</td><td colspan="2">The called subprogram must return using the COBOL GOBACK statement. The use of EXEC CICS RETURN in the called subprogram terminates the calling program.</td></tr>
<tr><td>Language of called subprogram</td><td>Any language supported by CICS</td><td colspan="2">VS COBOL II or Assembler</td></tr>
<tr><td>Contents of called or linked-to subprogram</td><td colspan="3"><b><i>Any function supported by CICS for the language (including calls to external databases, for example, DB2, and DL/I)</i></b>, with the exception that an Assembler subprogram cannot call a lower level subprogram.</td></tr>
<tr><td rowspan="3">Passing parameters to the subprogram</td><td colspan="3">Data can be passed by any of the standard CICS methods (COMMAREA, TWA, TCTUA, TS queues) provided that the called or linked-to subprogram is processed by the CICS translator.</td></tr>
<tr><td rowspan="2">If the COMMAREA is used, its address must be passed in the LINK command. If the linked-to subprogram uses 24-bit addressing and the COMMAREA is above the 16-megabyte line, CICS copies it below the 16-megabyte line and recopies it on return.</td><td colspan="2">The CALL statement must pass DFHEIBLK and DFHCOMMAREA as the first two parameters. See Note 1 on page 34.</td></tr>
<tr><td>In an ANSI85 unit of compilation, a nested program is not a subprogram, and the above rule can be varied. See "Nesting — what the application programmer must do" on page 38.</td><td>If the called subprogram uses 24-bit addressing and any parameter is above the 16-megabyte line, COBOL issues a message and abends (1033).</td></tr>
<tr><td>Storage</td><td>On each entry to the linked-to subprogram, a new initialized copy of its working storage is provided, and the run unit is reinitialized (in some circumstances, this can cause a performance degradation).</td><td colspan="2">On the first entry to the called subprogram within a CICS logical level, a new initialized copy of its working storage is provided. On subsequent entries to the called subprogram at the same logical level, the same working storage is provided in its last-used state, that is, no storage is freed, acquired, or initialized. If performance is unsatisfactory with LINK commands, COBOL calls may give improved results.</td></tr>
</table>

Table 1 (Page 2 of 2). Calling a subprogram from VS COBOL II

	EXEC CICS LINK	**Static COBOL CALL**	**Dynamic COBOL CALL**
CICS condition/AID and abend handling	On entry to the linked-to subprogram, 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 specified in the first active HANDLE ABEND found. If none is found, the transaction abends.	On entry to the called subprogram, no abend or condition handling is active. Within the subprogram, the normal CICS rules apply. To reinstate the caller's condition and abend handling, the called subprogram can issue EXEC CICS POP HANDLE. If this is done, the subprogram must issue EXEC CICS PUSH HANDLE before returning to the caller, otherwise COBOL abends (1013). If an abend occurs in the called subprogram while there is no abend handling, the effect is the same as that of an abend in the calling program.	

Notes:

1. The CALL has the following form:

   ```
   CALL 'PROG' USING DFHEIBLK DFHCOMMAREA
            PARM1 PARM2...
   ```

   In the called program PROG, the CICS translator inserts DFHEIBLK and DFHCOMMAREA into the linkage section and into the USING list of the procedure division statement. You code the procedure division statement normally, as follows:

   ```
   PROCEDURE DIVISION USING PARM1 PARM2...
   ```

   and the translator inserts DFHEIBLK and DFHCOMMAREA into this statement before PARM1.

2. You must always use the NODYNAM compiler option (the default) when you compile a VS COBOL II program that is to run with CICS, even if the program issues dynamic calls.

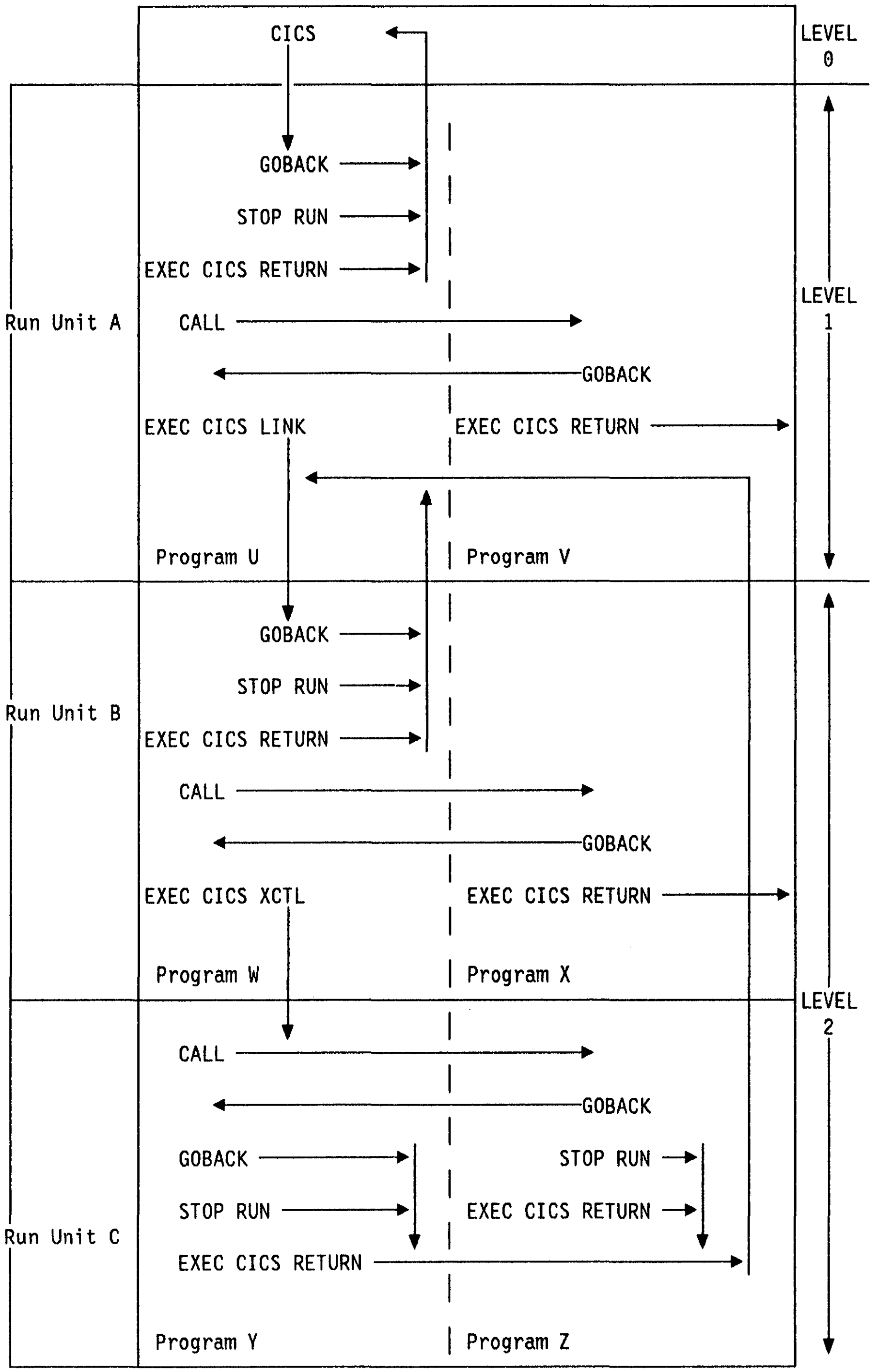


Figure 7. Flow of control between VS COBOL II programs and run units in a CICS environment. 'Run unit' is defined in "Mixing languages" on page 32.

VS COBOL II with the ANSI85 COBOL standards

VS COBOL II Release 1.3 supports the ANSI85 COBOL standards. CICS/MVS Release 2.1 introduces a new translator option, ANSI85, which supports most of these standards. If invoked with the ANSI85 option, the translator assumes the COBOL2 option.

CICS support for these standards takes the form of changes to the translator. Because the translator is not a compiler, it is not affected by all the ANSI85 standards. The standards that do affect the translator are:

- Blank lines intervening in literals
- Sequence numbers containing any character
- Lowercase characters supported in all COBOL words
- REPLACE statement
- Batch compilation
- Nested programs
- Reference modification
- Global variables
- Interchangeability of comma, semicolon, and space
- Symbolic character definition.

If a standard is not fully supported by the translator, a programming restriction is introduced. These restrictions are explained in the following descriptions of the standards and a summary is given after the descriptions. The translator actions given in the descriptions assume that the ANSI85 translator option has been specified.

As used here, a **unit of compilation** is a section of source input from which the compiler produces a single object program. A unit of compilation can consist of a containing program and other programs nested within it.

Blank lines intervening in literals

Blank lines can appear anywhere in a COBOL source program. A blank line is a line that contains only blanks between margin C (the continuation column) and margin R (the last character in the line) inclusive.

Translator action: If blank lines occur within literals in a VS COBOL II source program, the translator eliminates them from the translated output but includes them in the translated listing.

(If the ANSI85 option is not specified, a blank line in a literal causes a translator error.)

Sequence numbers containing any character

New standard: In a COBOL source program, the sequence number field can contain any character in the computer's character set. The sequence number fields need not be in any order and need not be unique.

Translator action: The translator makes no check on the contents or sequence of the sequence number fields.

If the SEQ translator option is specified, the translator issues a message saying that the SEQ option has no effect when the ANSI85 option is specified.

Lowercase characters supported in all COBOL words

New standard: Lowercase characters can occur anywhere in any COBOL word, including user-defined names, system names, and reserved words. A lowercase character can be used wherever an uppercase character is required by a COBOL compiler that does not conform to the ANSI85 standards.

Translator action: The translator listing and output preserve the case of COBOL text as entered.

In addition, the translator accepts:

- Mixed case in translator options.
- Mixed case in EXEC CICS commands, both for keywords and for arguments to keywords.

  (If the ANSI85 option is not specified, the translator expects COBOL words to consist entirely of uppercase characters.)

Notes:

1. The translator does not translate lowercase text into uppercase. Some names in COBOL text – for example, filenames and transaction IDs – must match externally defined names. Such names should always be entered in the same case as the external definition.
2. CBL and PROCESS cards must be in uppercase.

REPLACE statement

New standard: COBOL programs can include the new REPLACE statement, which allows the replacement of identified text by defined substitution text. The text to be replaced and inserted is specified as in the REPLACING option of the COPY statement, and can be pseudo-text, an identifier, a literal, or a COBOL word.

REPLACE statements are processed after COPY statements.

Translator action: The translator accepts REPLACE statements but does not translate text between pseudo-text delimiters, with one exception. CICS built-in functions (DFHRESP and DFHVALUE) are translated wherever they occur. CICS commands should not be placed between pseudotext delimiters.

Batch compilation

New standard: Separate COBOL programs can be compiled together as one input file. An END PROGRAM statement terminates each program. The END PROGRAM statement is optional for the last program in the batch.

Translator action: The translator accepts separate COBOL programs in a single input file, and interprets END PROGRAM statements according to the new standard.

Translator options specified as parameters when invoking the translator are in effect for the whole batch, but can be overridden for a unit of compilation by options specified in the CBL or PROCESS card that initiates the unit.

The options for a unit of compilation are determined according to the following order of priority:

1. Options specified in the CBL or PROCESS card that initiates the unit
2. Options specified when the translator is invoked
3. Default options.

Compiler and linkage editor: If you are using batch compilation, you must take some additional action to ensure that compilation and linkage editing are successful.

- Include the compiler NAME option as a parameter in the JCL statement that invokes the compiler or in a CBL statement for each top-level (non-nested) program. This causes the inclusion of a NAME statement at the end of each program. See Figure 8.

```
                  .................
                  ....program a....
                  .................
NAME PROGA(R)
                  .................
                  .................
                  ....program b....
                  .................
                  .................
NAME PROGB(R)
                  .................
                  ....program c....
                  .................
NAME PROGC(R)
```

Figure 8. Compiler output before editing

- Edit the compiler output to add INCLUDE and ORDER statements for the CICS COBOL stub to each object module. These statements cause the linkage editor to include the stub at the start of each load module. These statements can be anywhere in the module although by convention they are at the start. You may find it convenient to place them at the end of the module, immediately before each NAME statement. Figure 9 shows the output in Figure 8 after editing in this way.

```
                  ....program a....
                  .................
INCLUDE COBLIB(DFHECI)
ORDER DFHECI
NAME PROGA(R)
                  .................
                  .................
                  ....program b....
                  .................
                  .................
INCLUDE COBLIB(DFHECI)
ORDER DFHECI
NAME PROGB(R)
                  .................
                  ....program c....
                  .................
INCLUDE COBLIB(DFHECI)
ORDER DFHECI
NAME PROGC(R)
```

Figure 9. Linkage editor input

For batch compilation, you must vary the procedure described in the *CICS/MVS Operations Guide.* The following is a suggested method.

1. Split the supplied cataloged procedure, DFHEITCL, into two procedures: PROC1 containing the translate and compilation steps (TRN and COB), and PROC2 containing the LKED linkage editor step.
2. In PROC1, add the NAME option to the parameters in the EXEC statement for the compiler, which then looks like this:

   ```
   //COB EXEC PGM=IGYCRCTL,
   // PARM='....,NAME,....',
   // REGION=1024K
   ```

3. In PROC1, change the name and disposition of the &&LOADSET compiler output data set. At least remove the initial && from the data set name and change the disposition to CATLG. The SYSLIN statement should then read:

   ```
   //SYSLIN DD DSN=LOADSET,
   // DISP=(NEW,CATLG),UNIT=&WORK,
   // SPACE=(80,(250,100))
   ```

4. Run PROC1.
5. Edit the compiler output in the LOADSET data set to add the INCLUDE and ORDER statements as shown in Figure 9. If you use large numbers of programs in batches, you should write a simple program or REXX EXEC to insert the ORDER and INCLUDE statements.

6. In PROC2, add a DD statement for the library including the CICS stub. The standard name of this library is CICS21.COBLIB. The INCLUDE statement for the stub refers to this library by the DD name. In Figure 9 on page 37, we assume you have used the DD name COBLIB. The suggested statement is:

   ```
   //COBLIB DD DSN=CICS21.COBLIB,
   // DISP=SHR
   ```

7. In PROC2, replace the SYSLIN concatenation with the single statement:

   ```
   //SYSLIN DD DSN=LOADSET,
   // DISP=(OLD,DELETE)
   ```

   The above statement assumes that you have renamed the compiler output data set LOADSET.

8. Run PROC2.

Nested programs

New standard: Under the ANSI85 standard:

- COBOL programs can contain COBOL programs.
- Contained programs are included immediately before the END PROGRAM statement of the containing program.
- A contained program can also be a containing program; that is, it can itself contain other programs.
- Each contained or containing program is terminated by an END PROGRAM statement.

For an explanation of valid calls to nested programs and of the COMMON attribute of a nested program, see the *VS COBOL II Application Programming Guide*.

Translator action: The translator treats top-level and nested programs differently.

Top-level programs: The translator translates a top-level program (a program that is not contained by any other program) in the normal way, with one addition. The translator uses the GLOBAL storage class for all translator-generated variables in the working-storage section.

Nested programs: The translator translates nested or contained programs in a special way, as follows:

- A data division and linkage section are added if they do not already exist.
- Declarations for DFHEIBLK (EXEC interface block) and DFHCOMMAREA (communication area) are inserted into the linkage section.
- EXEC CICS commands and CICS built-in functions are translated.
- The PROCEDURE DIVISION statement is **not** modified.
- No translator-generated temporary variables, used for pre-call assignments, are inserted in the working-storage section.

Recognition of nested programs: If the **ANSI85** option is specified, the translator assumes that the input source starts with a top-level program if the first non-comment record is any of the following:

```
IDENTIFICATION DIVISION statement
CBL card
PROCESS card
```

If the first record is none of these, the translator treats the input as part of the PROCEDURE DIVISION of a nested program. The first CBL or PROCESS card indicates the start of a top-level program and of a new unit of compilation. Any IDENTIFICATION DIVISION statements that are found before the first top-level program indicate the start of a new nested program.

The practical effect of these rules is that nested programs cannot be held in separate files and translated separately. A top-level program and all its directly and indirectly contained programs constitute a single unit of compilation and should be submitted together to the translator.

Positioning of comments: The translator treats comments that follow an END PROGRAM statement as belonging to the *next* program in the input source. Comments that precede an IDENTIFICATION DIVISION statement appear in the listing *after* the IDENTIFICATION DIVISION statement.

To avoid confusion, always place comments:

- After the IDENTIFICATION DIVISION statement that initiates the program to which they refer, and
- Before the END PROGRAM statement that terminates the program to which they refer.

Nesting — what the application programmer must do

1. Submit a top-level containing program and all its directly and indirectly contained programs as a single unit of compilation.
2. In each nested program that contains EXEC commands, CICS built-in functions, or references to the EIB or COMMAREA, code DFHEIBLK and DFHCOMMAREA as the first two parameters of the PROCEDURE DIVISION statement as follows:

   ```
   PROCEDURE DIVISION USING DFHEIBLK
          DFHCOMMAREA PARM1 PARM2 ...
   ```

3. In every call to a nested program that contains EXEC commands, CICS built-in functions, or references to the EIB or COMMAREA, code DFHEIBLK and DFHCOMMAREA as the first two parameters of the CALL statement as follows:

   ```
   CALL 'PROGA' USING DFHEIBLK DFHCOMMAREA PARM1 PARM2 .
   ```

4. For every call that forms part of the control hierarchy between the top-level program and a nested program that contains EXEC commands, CICS built-in functions, or references to the EIB or COMMAREA, code DFHEIBLK and DFHCOMMAREA as the first two parameters of the CALL and PROCEDURE DIVISION

statements in the calling and called programs respectively. This is necessary to allow addressability to the EIB and COMMAREA to be passed to programs not directly contained by the top-level program.

5. If it is not necessary to insert DFHEIBLK and DFHCOMMAREA in the PROCEDURE DIVISION of a nested program for any of the reasons given above (1, 3, and 4), calls to that program should **not** include DFHEIBLK and DFHCOMMAREA in the parameter list of the CALL statement.

Nesting — example: A unit of compilation (see Figure 10) consists of a top-level program **W** and three nested programs, **X**, **Y**, and **Z**, all directly contained by **W**.

Program W During initialization and termination, calls **Y** and **Z** to do initial CICS processing and non-CICS file access. Calls **X** to do main processing.

Program X Calls **Z** for non-CICS file access and **Y** for CICS processing.

Program Y Issues CICS commands. Calls **Z** for non-CICS file access

Program Z Accesses files in batch mode.

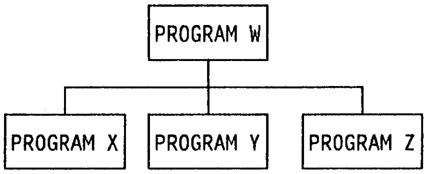


Figure 10. Nested program example—nesting structure

Applying the rules:

- **Y** must be COMMON to enable a call from **X**.
- **Z** must be COMMON to enable calls from **X** and **Y**.
- **Y** issues CICS commands, therefore:
  - All calls to **Y** must have DFHEIBLK and DFHCOMMAREA as the first two parameters.
  - **Y**'s PROCEDURE DIVISION statement must have DFHEIBLK and DFHCOMMAREA as the first two parameters.
- Though **X** does not access the EIB or the communication area, it calls **Y**, which issues CICS commands. Therefore both the call to **X** and **X**'s PROCEDURE DIVISION statement must have DFHEIBLK and DFHCOMMAREA as the first two parameters.

Figure 11 illustrates the above points.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.W
                 .
                 .
PROCEDURE DIVISION.
                 .
                 .
    CALL Z.
                 .
                 .
    CALL Y USING DFHEIBLK DFHCOMMAREA.
                 .
                 .
    CALL X USING DFHEIBLK DFHCOMMAREA.
                 .
                 .

  IDENTIFICATION DIVISION.
  PROGRAM-ID.X
                 .
                 .
  PROCEDURE DIVISION USING DFHEIBLK DFHCOMMAREA.
                 .
                 .
    CALL Z.
                 .
                 .
    CALL Y USING DFHEIBLK DFHCOMMAREA.
                 .
                 .
  END PROGRAM X

  IDENTIFICATION DIVISION.
  PROGRAM-ID.Y IS COMMON
                 .
                 .
  PROCEDURE DIVISION USING DFHEIBLK DFHCOMMAREA.
                 .
                 .
    CALL Z.
                 .
                 .
    EXEC CICS ....
                 .
                 .
  END PROGRAM Y

  IDENTIFICATION DIVISION.
  PROGRAM-ID.Z IS COMMON
                 .
                 .
  PROCEDURE DIVISION.
                 .
                 .
                 .
  END PROGRAM Z

END PROGRAM W
```

Figure 11. Nested program example — coding

Reference modification

New standard: Reference modification is supported. Reference modification is a method of referencing a substring of a character data item by specifying the starting (leftmost) position of the substring in the data item and, optionally the length of the substring. The acceptable formats are:

```
data-name (starting-position:)
data-name (starting-position:  length)
```

Data-name can be subscripted or qualified or both. Both *starting-position* and *length* can be arithmetic expressions.

Translator action: The translator accepts reference modification wherever the name of a character variable is permitted in a COBOL program or in an EXEC CICS command.

Note: If a CICS command uses reference modification in defining a data value, it *must* include a LENGTH option to specify the data length. Otherwise the translator generates a COBOL call with a LENGTH register reference in the form:

```
LENGTH OF (reference modification)
```

This is rejected by the compiler.

Global variables

New standard: The GLOBAL variable storage class is supported. A variable defined with the GLOBAL variable storage class in a top-level program (see "Top-level programs" on page 38) can be referred to in any of its nested programs, whether directly or indirectly contained.

Translator action: The translator accepts the GLOBAL keyword.

Comma and semicolon as delimiters

New standard: A separator comma is a comma followed by a space. A separator semicolon is a semicolon followed by a space. A separator comma or a separator semicolon can be used as a separator wherever a space alone can be used. (VS COBOL II Release 1.2 restricted the use of commas and semicolons to positions specifically defined in individual statement formats.)

Translator action: The translator accepts the use in COBOL statements of a separator comma or a separator semicolon wherever a space can be used. For example, the translator accepts the statement:

```
IDENTIFICATION; DIVISION
```

The translator does not support the use of the separator comma and separator semicolon as delimiters in EXEC CICS commands. The only acceptable word delimiter in an EXEC CICS command continues to be a space.

Symbolic character definition

New standard: Symbolic characters can be defined in the SPECIAL-NAMES paragraph after the ALPHABET clause. A symbolic character is a program-defined word that represents a one-character figurative constant.

Translator action: The translator accepts the use of symbolic characters as specified in the standard.

Note: In general, the compiler does not accept the use of figurative constants and symbolic characters as arguments in CALL statements. For this reason, do not use figurative constants or symbolic constants in CICS commands, which are converted into CALL statements by the translator. There is one exception to this restriction: a figurative constant is acceptable in a CICS command as an argument to *pass* a value provided it is of the correct data type. For example, a numeric figurative constant can be used in the LENGTH option.

Summary of restrictions

The following is a summary of the programming restrictions associated with CICS translator suppport for the ANSI85 COBOL standards.

- With the **ANSI85** option, the translator varies the rule for parameters to be passed by a static VS COBOL II call (see "CALL statement from VS COBOL II program" on page 28). For details, see "Nesting – what the application programmer must do" on page 38.
- A REPLACE statement must not contain an EXEC CICS command in pseudo-text.
- Programs cannot use a comma or semicolon as a word separator in a CICS command.
- Programs cannot use a symbolic character as an argument in a CICS command.
- Comments should not precede the IDENTIFICATION DIVISION statement or follow the END PROGRAM statement.
- CICS commands that use reference modification to define a character data value must include a LENGTH option to define the data length.
- A name that must match an external definition, for example a filename or a transaction ID, must be coded in the same case as the external definition.

PL/I considerations

Restrictions

The following restrictions apply to a PL/I program that is to be used as a CICS application program. See the *PL/I Optimizing Compiler Programmer's Guide* for more information about these facilities.

1. You cannot use the multitasking built-in functions:
   COMPLETION
   PRIORITY
   STATUS

2. You cannot use the multitasking options:
   EVENT
   PRIORITY
   TASK

3. Do not use the PL/I statements:
   READ
   WRITE
   GET
   PUT
   OPEN
   CLOSE
   DISPLAY
   DELAY
   REWRITE
   LOCATE
   DELETE
   UNLOCK
   STOP
   HALT
   EXIT
   FETCH
   RELEASE

   You are provided with CICS commands for the storage and retrieval of data, and for communication with terminals. (However, you can use CLOSE, PUT, and OPEN for SYSPRINT.)

   See the *PL/I Optimizing Compiler Programmer's Guide* for information on when the use of these PL/I statements is necessary and the consequences of using them.

4. You cannot use PL/I Sort/Merge.

5. You cannot use static storage (except for read-only data).

6. If you declare a variable with the attributes STATIC and EXTERNAL, you must also include the INITIAL attribute. If you do not, such a declaration will generate a common CSECT that cannot be handled by CICS.

7. You cannot use the PL/I 48-character set option.

8. You cannot write your PL/I programs with lowercase keywords.

9. If a CICS command uses the SUBSTR built-in function in defining a data value, it *must* include a LENGTH option to specify the data length. Otherwise the translator generates a PL/I call including an invocation of the CSTG builtin function in the form:

   ```
   CSTG(SUBSTR(..,..,..))
   ```

   This is rejected by the compiler.

10. If you are using PL/I and your load module requires more than 64K bytes of dynamic storage to initialize, this results in an APLG abend. If you are using MVS/XA, this limit is increased to 1 Mbyte for areas that you allocate explicitly above the 16-megabyte line using an EXEC CICS GETMAIN command with the FLENGTH option. However, all automatic storage and DSA for PL/I save areas are below the 16-megabyte line even when the program is specified as AMODE(31) and RMODE(ANY). This is because PL/I has to do a single GETMAIN below the line for this storage and CICS has a restriction of 64K bytes in a single GETMAIN below the 16-megabyte line.

    You can avoid this happening in an XA AMODE(31) environment by doing the following. Instead of making your biggest PL/I structures and arrays AUTOMATIC, define them as based on a POINTER variable, which you initialize using EXEC CICS GETMAIN SET(pointer) FLENGTH(length). Use FLENGTH instead of LENGTH, and make the length more than 4K bytes so that it is allocated above the 16-megabyte line. For example, suppose you have a PL/I program with these arrays declared:

    ```
    DCL A(1000,10) FLOAT
    ```

    and

    ```
    DCL B(100,10) CHAR(100)
    ```

    These arrays need 40000 bytes (that is, 1000 x 10 x 4) and 100000 bytes (that is, 100 x 10 x 100), respectively. Code your PL/I program as follows:

    ```
    DCL (APOINTER, BPOINTER) POINTER;
    DCL A(1000,10) FLOAT BASED(APOINTER),
        B(100,10) CHAR(100) BASED(BPOINTER),
        CSTG BUILTIN;
    EXEC CICS GETMAIN SET(APOINTER) FLENGTH(L1);
    EXEC CICS GETMAIN SET(BPOINTER) FLENGTH(L2);
    ```

    This prevents an APLG abend occurring and means that your program can use storage above the line that would otherwise have been required below the line.

DL/I CALL interface: Normally, with MVS/XA, you link edit your PL/I programs with the options AMODE(31) and RMODE(ANY) so that they can be loaded and acquire working storage above the 16-megabyte line. However, if a program uses the CALL DL/I interface, the program can reside above the 16-megabyte line, although its call parameter list and the call parameters must reside below the 16-megabyte line. DL/I DFHFC requests are not supported when running in 31-bit mode.

PL/I STAE execution-time option

If this option is specified, an abend occurring in the transaction will be handled by PL/I error handling routines and the transaction may terminate normally, in which case CICS facilities such as dynamic transaction backout (DTB) will not be invoked.

Further information about PL/I and the STAE option is given in the *CICS/MVS Recovery and Restart Guide.*

Compilers supported

You can use only the following PL/I compilers to process your PL/I application programs:

- OS PL/I Optimizing Compiler, Version 1, Release 4.0.
- OS PL/I Optimizing Compiler, Version 1, Release 5.1.

OPTIONS(MAIN) specification

If OPTIONS(MAIN) is specified in an application program, that program can be the first program of a transaction, or control can be passed to it by means of a LINK or XCTL command.

If OPTIONS(MAIN) is not specified, it cannot be the first program in a transaction, nor can it have control passed to it by a LINK or XCTL command, but it can be link-edited to a main program.

The definition of the EIB is generated in each program. However, in programs other than that declared with OPTIONS(MAIN), addressability to the EIB is the user's responsibility.

Addressability is achieved by using the command:

```
EXEC CICS ADDRESS EIB(DFHEIPTR)
```

or by passing the EIB address or particular fields therein as arguments to the CALL statement that invokes the external procedure.

Program segments

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

Subsequent copying or manipulating of statements originally inserted by the CICS translator in an application program may produce unpredictable results.

The external procedure must always be passed through the CICS translator, even when all its commands are in included segments.

Chapter 1.5. Exceptional conditions

A CICS "exceptional condition", or "condition" for short, is defined as the reason why a CICS command cannot be executed.

Conditions may occur at any time during the execution of a command and, unless you specify otherwise in your application program, a standard system (default) action for each condition will be taken. Usually, this default action is to terminate the task abnormally.

There are about 70 conditions in all, each one identified by name (for example LENGERR) and a corresponding number. All the conditions are listed, in alphabetic order of name, at the end of this chapter, and under EIBRCODE in Appendix A, "EXEC interface block" on page 339. Their numbers are listed in numeric order under EIBRESP, also in Appendix A.

An application program can be in one of three possible states with respect to a condition detected during the attempted execution of a command:

1. **HANDLE CONDITION active**. Control goes to a label in the program defined earlier by a HANDLE CONDITION command.

   This state occurs after the execution of a

   ```
   HANDLE CONDITION condition(label)
   ```

   or

   ```
   HANDLE CONDITION ERROR(label)
   ```

2. **Take no action**. Control returns to the next instruction following the command that has failed to execute. At the same time, a return code is set in EIBRESP and EIBRCODE corresponding to the condition encountered.

   This state occurs after the execution of an

   ```
   IGNORE CONDITION condition
   ```

   or an

   ```
   IGNORE CONDITION ERROR
   ```

   if there is no current active HANDLE CONDITION command that includes a label.

   This state also occurs during execution of a command that includes one of the options NOHANDLE or RESP.

3. **Standard system action (default action)**. For most conditions, this is to terminate the task abnormally. However, for the ENQBUSY, NOJBUFSP, NOSPACE, NOSTG, QBUSY, SESSBUSY, and SYSBUSY conditions, standard system action is to suspend the task until the required resource becomes available, when execution of the associated command is resumed.

   This state occurs if neither a HANDLE CONDITION condition(label) nor a HANDLE CONDITION ERROR(label) has been executed.

   This state also occurs after executing a HANDLE CONDITION condition command without a label or PUSH HANDLE.

Alternative to the HANDLE command

The NOHANDLE, RESP, and RESP2 options (described below) are supplied as an alternative to the HANDLE command just described.

You are recommended to use these options because they allow you to write structured programs.

NOHANDLE option: You can use the NOHANDLE option with any command to specify that you want no action to be taken for any condition or attention identifier (AID) resulting from the execution of that command. In this way you can control the scope of the IGNORE CONDITION command to cover specified conditions for all commands (until a HANDLE CONDITION for the condition is executed). Similarly, you can control the scope of the NOHANDLE option to cover all conditions for specified commands.

RESP and RESP2 options: You can use the RESP and RESP2 options with any command to test the response to the execution of that command.

| If the HANDLE CONDITION overflow is not active, the overflow condition will not be raised and RESP or RESP2 will therefore indicate a normal response by default.

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 by means of DFHRESP, as follows:

```
EXEC CICS WRITEQ TS FROM(abc)
                    QUEUE(qname)
                    RESP(xxx)
     .
     .
If xxx=DFHRESP(NOSPACE) THEN ...
```

The above form of DFHRESP applies to both COBOL and PL/I. For assembler, the test would be:

```
CLC     xxx,DFHRESP(NOSPACE)
BE      ...
```

Because the use of RESP implies NOHANDLE, you must be careful when using RESP 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 will be ignored.

RESP2(yyy)

"yyy" is a fullword binary value that further qualifies the response to some commands. It is used in the INQUIRE and SET commands, and in the spool commands of the CICS interface to JES. These commands are primarily for the use of the system programmer, and are described in the *CICS/MVS Customization Guide.*

ERROR exceptional condition: Apart from the conditions associated with individual commands, there is a general condition named ERROR whose default action also is to terminate the task abnormally.

If no HANDLE CONDITION command is active for a condition, but one is active for ERROR, control will be passed to the label specified for ERROR.

A subsequent HANDLE CONDITION command (with or without a label) for a condition overrides the HANDLE CONDITION ERROR command for that condition.

Do not include commands in an error routine that may give rise to the same condition that caused the branch to the routine; take special care not to cause a loop on the ERROR condition.

You can avoid a loop by including a HANDLE CONDITION ERROR command as the first command in your error routine. Reinstate the original error action at the end of your error routine by including a second HANDLE CONDITION ERROR command.

Unsupported function: A task will be abended unconditionally if, in a command, you request an unsupported function, even if you have specified a HANDLE ERROR command or an IGNORE ERROR command, or have included the RESP condition in that command.

NOTAUTH exceptional condition: The NOTAUTH condition is a general condition that can be associated with individual commands. It is raised when a resource security check has failed.

The reasons for the failure are the same as for the AEY7 abend code, as described in the *CICS/MVS Messages and Codes* manual.

Summary of CONDITION commands and actions: CICS maintains a table of conditions referred to by HANDLE CONDITION and IGNORE CONDITION commands in an application program. Execution of these commands either updates the existing entry, or causes a new entry to be made if the condition has not yet been the subject of such a command. Each entry indicates one of the three states described earlier.

When the condition occurs, the following tests are made:

1. If the command has NOHANDLE or RESP, control returns to the next instruction in the application program except for the ALLOCATE, ENQ, GETMAIN, JOURNAL, READQ TD, and WRITEQ TS commands, where NOHANDLE will suspend execution of the application program until the specified resource becomes available. Otherwise the condition table is scanned to ascertain what is to be done.
2. If an entry for the condition exists, this determines the action.
3. If no entry exists and the default action for this condition is to suspend execution:
   - If the command has the NOSUSPEND or NOQUEUE option, control returns to the next instruction.
   - If the command does not have one of these options, the task is suspended.
4. If no entry exists and the default action for this condition is to abend, a second search is made for the ERROR condition and this entry, if found, determines the action. If it is not found, the task is abended.

The ALLOCATE, ENQ, GETMAIN, JOURNAL, READQ TD, and WRITEQ TS commands can give rise to conditions for which the default action is to suspend the execution of the application program until the specified resource becomes available. On these commands, the NOSUSPEND option is provided to inhibit this waiting and to cause an immediate return to the instruction in the application program following the command.

Throughout this manual, where appropriate, conditions are described together with the CICS default action at the end of a chapter, and a list of conditions that apply to a command is included within the syntax box for the command.

Some conditions can occur during the execution of any one of a number of unrelated commands; for example, IOERR can occur during file control operations, interval control operations, and others. If you want the same action for all occurrences, simply code a single HANDLE CONDITION IOERR command at the beginning of your program.

If you want different actions, you must code HANDLE CONDITION commands specifying different labels at appropriate points in the program, or you can specify the same label for all commands. You can test the EIBFN, EIBRCODE, and EIBRESP fields (in the EXEC interface block – EIB) to find out which condition has occurred and in which command. The EIB is described in Appendix A, "EXEC interface block" on page 339.

Handle exceptional conditions (HANDLE CONDITION)

```
HANDLE CONDITION
condition[(label)]...
```

condition[(label)]

'condition' specifies the name of the condition, and 'label' specifies the location within the program to be branched to if the condition occurs.

If you omit the condition, the default action for the condition is taken unless the default action is to terminate the task abnormally, in which case the ERROR condition occurs.

If you omit label, any HANDLE CONDITION command for the condition is deactivated and the default action for the condition is taken if the condition occurs.

You use this 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 HANDLE CONDITION command is executed before the command that may give rise to the associated condition.

At the end of this chapter is a list of the conditions that can be used in this command.

You cannot include more than sixteen conditions in the same command, and each condition should be separated by at least one space. You must specify additional conditions in further 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 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 executing, 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 or RESP option on a command.

When control passes to another program, the 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 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. (See "Chapter 4.4. Program control" on page 289 for information about logical levels.) This will not apply if macro-level links are used; current active conditions will remain active.

The following example shows you how to handle conditions, such as DUPREC, LENGERR, and so on, that can occur when you use a WRITE command to add a record to a data set. Let us 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
```

In an assembler-language application program, a branch to a label caused by a condition will restore the registers in the application program to their values in the program at the point where the command that caused the condition is issued. On MVS/XA, the addressing mode will be set to that in effect at the point where the HANDLE CONDITION command is issued.

In a PL/I application program, a branch to a label in an inactive procedure or in an inactive begin block, caused by a condition, will produce unpredictable results.

Ignore exceptional conditions (IGNORE CONDITION)

```
IGNORE CONDITION
condition condition ...
```

condition

specifies the name of the condition that is to be ignored.

You use this command 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) will be passed to your application program.

At the end of this chapter is a list of the conditions that can be used in this command.

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 executing, or until a HANDLE CONDITION command for the same condition is encountered, in which case the IGNORE CONDITION command is overridden.

You cannot code more than sixteen conditions in the same command, and each condition should be separated by at least one space. Additional conditions must be specified in further IGNORE CONDITION commands.

Suspend condition handling (PUSH and POP)

The PUSH and POP commands enable you to suspend all current HANDLE CONDITION, IGNORE CONDITION, HANDLE AID, HANDLE ABEND, and IGNORE commands. This can be useful, for example, during a branch to a subroutine embedded in a main program.

Normally, when a CICS program links to 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 the PUSH command to suspend existing HANDLE commands. The form of the PUSH command is:

```
PUSH HANDLE
```

Before returning control to the caller, a called program can restore the original commands using the POP command, which has the form:

```
POP HANDLE
```

You can nest PUSH...POP command sequences within a task. Each PUSH command stacks a set of specifications; the POP that follows it restores them.

List of exceptional conditions

The following list shows all the conditions that can occur during the execution of CICS commands. Each condition is followed by one or more commands during the execution of which the condition may occur. The numbers in parentheses are the numbers of the chapters that describe those commands. For the meaning of a condition and the default action associated with that condition, see the list of conditions at the end of the indicated chapter.

Condition	Command	Chapter
CBIDERR	ALLOCATE	(3.3)
	CONVERSE	(3.3)
	EXTRACT ATTACH	(3.3)
	SEND	(3.3)
CCERROR	SPOOLOPEN	(3.4)
	SPOOLWRITE	(3.4)
DISABLED	DELETE	(2.4)
	DELETEQ TD	(4.6)
	ENDBR	(2.4)
	READ	(2.4)
	READNEXT	(2.4)
	READPREV	(2.4)
	READQ TD	(4.6)
	RESETBR	(2.4)
	REWRITE	(2.4)
	STARTBR	(2.4)
	UNLOCK	(2.4)
	WRITE	(2.4)
	WRITEQ TD	(4.6)
DSIDERR	ENDBR	(2.4)
	DELETE	(2.4)
	READ	(2.4)
	READNEXT	(2.4)
	READPREV	(2.4)
	RESETBR	(2.4)
	REWRITE	(2.4)
	STARTBR	(2.4)
	UNLOCK	(2.4)
	WRITE	(2.4)
DSSTAT	ISSUE RECEIVE	(3.5)
DUPKEY	READ	(2.4)
	READNEXT	(2.4)
	READPREV	(2.4)
DUPREC	WRITE	(2.4)
	REWRITE	(2.4)
ENDDATA	RETRIEVE	(4.2)
ENDFILE	READNEXT	(2.4)
	READPREV	(2.4)
ENDINPT	RECEIVE	(3.3)
ENQBUSY	ENQ	(4.3)
ENVDEFERR	RETRIEVE	(4.2)
EOC	CONVERSE	(3.3)
	RECEIVE	(3.3)
	RECEIVE MAP	(3.2)
	RECEIVE PARTN	(3.2)
EODS	CONVERSE	(3.3)
	ISSUE RECEIVE	(3.5)

	RECEIVE	(3.3)
	RECEIVE MAP	(3.2)
	RECEIVE PARTN	(3.2)
EOF	CONVERSE	(3.3)
	RECEIVE	(3.3)
ERROR	General exceptional condition.	(1.5)
	Not included in the list of conditions in the syntax of individual commands.	
EXPIRED	DELAY	(4.2)
	POST	(4.2)
FUNCERR	ISSUE ABORT	(3.5)
	ISSUE ADD	(3.5)
	ISSUE END	(3.5)
	ISSUE ERASE	(3.5)
	ISSUE NOTE	(3.5)
	ISSUE QUERY	(3.5)
	ISSUE REPLACE	(3.5)
	ISSUE SEND	(3.5)
	ISSUE WAIT	(3.5)
IGREQCD	CONVERSE	(3.3)
	ISSUE SEND	(3.5)
	SEND	(3.3)
	SEND CONTROL	(3.2)
	SEND MAP	(3.2)
	SEND PAGE	(3.2)
	SEND TEXT	(3.2)
IGREQID	SEND CONTROL	(3.2)
	SEND MAP	(3.2)
	SEND TEXT	(3.2)
ILLOGIC	DELETE	(2.4)
	ENDBR	(2.4)
	READ	(2.4)
	READNEXT	(2.4)
	READPREV	(2.4)
	RESETBR	(2.4)
	REWRITE	(2.4)
	SPOOLCLOSE	(3.4)
	SPOOLOPEN	(3.4)
	STARTBR	(2.4)
	UNLOCK	(2.4)
	WRITE	(2.4)
INBFMH	CONVERSE	(3.3)
	RECEIVE	(3.3)
INVERRTERM	ROUTE	(3.2)
INVLDC	ROUTE	(3.2)
	SEND CONTROL	(3.2)
	SEND MAP	(3.2)
	SEND TEXT	(3.2)
INVMPSZ	RECEIVE MAP	(3.2)
	SEND MAP	(3.2)
INVPARTN	RECEIVE MAP	(3.2)
	RECEIVE PARTN	(3.2)
	SEND CONTROL	(3.2)
	SEND MAP	(3.2)
	SEND TEXT	(3.2)
INVPARTNSET	SEND PARTNSET	(3.2)
INVREQ	ALLOCATE	(3.3)
	ASSIGN	(1.6)
	CANCEL	(4.2)
	CONNECT PROCESS	(3.3)
	CONVERSE	(3.3)
	DELAY	(4.2)
	DELETE	(2.4)
	ENDBR	(2.4)
	ENTER	(5.3)
	EXTRACT ATTACH	(3.3)
	EXTRACT PROCESS	(3.3)
	EXTRACT TCT	(3.3)
	FREE	(3.3)
	ISSUE ABEND	(3.3)
	ISSUE CONFIRMATION	(3.3)
	ISSUE COPY	(3.3)
	ISSUE ENDFILE	(3.3)
	ISSUE ENDOUTPUT	(3.3)
	ISSUE EODS	(3.3)
	ISSUE ERASEAUP	(3.3)
	ISSUE ERROR	(3.3)
	ISSUE LOAD	(3.3)
	ISSUE PRINT	(3.3)
	ISSUE RESET	(3.3)
	POST	(4.2)
	READ	(2.4)
	READNEXT	(2.4)
	READPREV	(2.4)
	READQ TS	(4.7)
	RECEIVE	(3.3)
	RESETBR	(2.4)
	RETRIEVE	(4.2)
	RETURN	(4.4)
	REWRITE	(2.4)
	ROUTE	(3.2)
	SEND	(3.3)
	SEND CONTROL	(3.2)
	SEND MAP	(3.2)
	SEND PAGE	(3.2)
	SEND PARTNSET	(3.2)
	SEND TEXT	(3.2)
	SPOOLOPEN	(3.4)
	SPOOLCLOSE	(3.4)
	START	(4.2)
	STARTBR	(2.4)
	WAIT CONVID	(3.3)
	WAIT EVENT	(4.2)
	WAIT JOURNAL	(5.5)
	WRITE	(2.4)
	WRITEQ TS	(4.7)
INVTSREQ	RETRIEVE	(4.2)

IOERR	DELETE	(2.4)
	JOURNAL	(5.5)
	READ	(2.4)
	READNEXT	(2.4)
	READPREV	(2.4)
	READQ TD	(4.6)
	READQ TS	(4.7)
	RESETBR	(2.4)
	RETRIEVE	(4.2)
	REWRITE	(2.4)
	SPOOLCLOSE	(3.4)
	SPOOLOPEN	(3.4)
	SPOOLWRITE	(3.4)
	START	(4.2)
	STARTBR	(2.4)
	UNLOCK	(2.4)
	WAIT JOURNAL	(5.5)
	WRITE	(2.4)
	WRITEQ TD	(4.6)
	WRITEQ TS	(4.7)
ISCINVREQ	CANCEL	(4.2)
	DELETE	(2.4)
	DELETEQ TD	(4.6)
	DELETEQ TS	(4.7)
	ENDBR	(2.4)
	READ	(2.4)
	READNEXT	(2.4)
	READPREV	(2.4)
	READQ TD	(4.6)
	READQ TS	(4.7)
	RESETBR	(2.4)
	RETRIEVE	(4.2)
	REWRITE	(2.4)
	START	(4.2)
	STARTBR	(2.4)
	UNLOCK	(2.4)
	WRITE	(2.4)
	WRITEQ TD	(4.6)
	WRITEQ TS	(4.7)
ITEMERR	READQ TS	(4.7)
	WRITEQ TS	(4.7)
JIDERR	JOURNAL	(5.5)
	WAIT JOURNAL	(5.5)
LENGERR	CONNECT PROCESS	(3.3)
	CONVERSE	(3.3)
	DUMP	(5.4)
	GETMAIN	(4.5)
	ISSUE RECEIVE	(3.5)
	JOURNAL	(5.5)
	READ	(2.4)
	READNEXT	(2.4)
	READPREV	(2.4)
	READQ TD	(4.6)
	READQ TS	(4.7)
	RECEIVE	(3.3)
	RECEIVE PARTN	(3.2)
	RETRIEVE	(4.2)
	REWRITE	(2.4)
	SEND	(3.3)
	SPOOLWRITE	(3.4)
	WRITE	(2.4)
	WRITEQ TD	(4.6)
MAPERROR	SPOOLWRITE	(3.4)
MAPFAIL	RECEIVE MAP	(3.2)
	SPOOLWRITE	(3.4)
NAMEERROR	SPOOLOPEN	(3.4)
NOJBUFSP	JOURNAL	(5.5)
NONVAL	ISSUE LOAD	(3.3)
NOPASSBKRD	RECEIVE	(3.3)
NOPASSBKWR	SEND	(3.3)
NOSPACE	REWRITE	(2.4)
	SPOOLCLOSE	(3.4)
	SPOOLOPEN	(3.4)
	SPOOLWRITE	(3.4)
	WRITE	(2.4)
	WRITEQ TD	(4.6)
	WRITEQ TS	(4.7)
NOSPOOL	SPOOLCLOSE	(3.4)
	SPOOLOPEN	(3.4)
	SPOOLWRITE	(3.4)
NOSTART	ISSUE LOAD	(3.3)
NOSTG	GETMAIN	(4.5)
NOTALLOC	CONNECT PROCESS	(3.3)
	CONVERSE	(3.3)
	EXTRACT ATTACH	(3.3)
	EXTRACT PROCESS	(3.3)
	FREE	(3.3)
	ISSUE ABEND	(3.3)
	ISSUE CONFIRMATION	(3.3)
	ISSUE DISCONNECT	(3.3)
	ISSUE ERROR	(3.3)
	ISSUE SIGNAL	(3.3)
	POINT	(3.3)
	RECEIVE	(3.3)
	SEND	(3.3)
	WAIT CONVID	(3.3)
	WAIT TERMINAL	(3.3)
NOTAUTH	CANCEL	(4.2)
	DELETE	(2.4)
	DELETEQ TD	(4.6)
	DELETEQ TS	(4.7)
	ENDBR	(2.4)
	LINK	(4.4)
	LOAD	(4.4)
	JOURNAL	(5.5)
	READ	(2.4)
	READQ TD	(4.6)
	READQ TS	(4.7)

	READNEXT	(2.4)
	READPREV	(2.4)
	RELEASE	(4.4)
	RESETBR	(2.4)
	RETRIEVE	(4.2)
	RETURN	(4.4)
	REWRITE	(2.4)
	SPOOLCLOSE	(3.4)
	SPOOLOPEN	(3.4)
	SPOOLWRITE	(3.4)
	START	(4.2)
	STARTBR	(2.4)
	UNLOCK	(2.4)
	WAIT JOURNAL	(5.5)
	WRITE	(2.4)
	WRITEQ TD	(4.6)
	WRITEQ TS	(4.7)
	XCTL	(4.4)
NOTFND	CANCEL	(4.2)
	DELETE	(2.4)
	READ	(2.4)
	READNEXT	(2.4)
	READPREV	(2.4)
	RESETBR	(2.4)
	RETRIEVE	(4.2)
	STARTBR	(2.4)
NOTOPEN	DELETE	(2.4)
	ENDBR	(2.4)
	JOURNAL	(5.5)
	READ	(2.4)
	READNEXT	(2.4)
	READPREV	(2.4)
	READQ TD	(4.6)
	RESETBR	(2.4)
	REWRITE	(2.4)
	STARTBR	(2.4)
	UNLOCK	(2.4)
	WAIT JOURNAL	(5.5)
	WRITE	(2.4)
	WRITEQ TD	(4.6)
OVERFLOW	SEND MAP	(3.2)
PARTNFAIL	RECEIVE MAP	(3.2)
PGMIDERR	HANDLE ABEND	(5.2)
	LINK	(4.4)
	LOAD	(4.4)
	RELEASE	(4.4)
	XCTL	(4.4)
QBUSY	READQ TD	(4.6)
QIDERR	DELETEQ TD	(4.6)
	DELETEQ TS	(4.7)
	READQ TD	(4.6)
	READQ TS	(4.7)
	WRITEQ TD	(4.6)
	WRITEQ TS	(4.7)
QZERO	READQ TD	(4.6)
RDATT	CONVERSE	(3.3)
	RECEIVE MAP	(3.2)
	RECEIVE	(3.3)
RETPAGE	SEND CONTROL	(3.2)
	SEND MAP	(3.2)
	SEND PAGE	(3.2)
	SEND TEXT	(3.2)
ROLLEDBACK	SYNCPOINT	(5.6)
RTEFAIL	ROUTE	(3.2)
RTESOME	ROUTE	(3.2)
SELNERR	ISSUE ABORT	(3.5)
	ISSUE ADD	(3.5)
	ISSUE END	(3.5)
	ISSUE ERASE	(3.5)
	ISSUE NOTE	(3.5)
	ISSUE QUERY	(3.5)
	ISSUE REPLACE	(3.5)
	ISSUE SEND	(3.5)
	ISSUE WAIT	(3.5)
SESSBUSY	ALLOCATE	(3.3)
SESSIONERR	ALLOCATE	(3.3)
	CONVERSE	(3.3)
	EXTRACT ATTACH	(3.3)
	FREE	(3.3)
	ISSUE DISCONNECT	(3.3)
	ISSUE SIGNAL	(3.3)
	POINT	(3.2)
	RECEIVE	(3.2)
	SEND	(3.3)
	WAIT TERMINAL	(3.3)
SIGNAL	CONVERSE	(3.3)
	RECEIVE	(3.3)
	SEND	(3.3)
	WAIT CONVID	(3.3)
	WAIT SIGNAL	(3.3)
	WAIT TERMINAL	(3.3)
SYSBUSY	ALLOCATE	(3.3)
SYSIDERR	ALLOCATE	(3.3)
	CANCEL	(4.2)
	DELETE	(2.4)
	DELETEQ TD	(4.6)
	DELETEQ TS	(4.7)
	ENDBR	(2.4)
	READ	(2.4)
	READNEXT	(2.4)
	READPREV	(2.4)
	READQ TD	(4.6)
	READQ TS	(4.7)
	RESETBR	(2.4)

	RETRIEVE	(4.2)
	REWRITE	(2.4)
	START	(4.2)
	STARTBR	(2.4)
	UNLOCK	(2.4)
	WRITE	(2.4)
	WRITEQ TD	(4.6)
	WRITEQ TS	(4.7)
TERMERR	CONVERSE	(3.3)
	ISSUE ABEND	(3.3)
	ISSUE CONFIRMATION	(3.3)
	ISSUE COPY	(3.3)
	ISSUE DISCONNECT	(3.3)
	ISSUE EODS	(3.3)
	ISSUE ERASEAUP	(3.3)
	ISSUE ERROR	(3.3)
	ISSUE LOAD	(3.3)
	ISSUE PRINT	(3.3)
	ISSUE SIGNAL	(3.3)
	RECEIVE	(3.3)
	SEND	(3.3)
	WAIT SIGNAL	(3.3)
TERMIDERR	ISSUE COPY	(3.3)
	START	(4.2)
TRANSIDERR	START	(4.2)
TSIOERR	PURGE MESSAGE	(3.2)
	SEND CONTROL	(3.2)
	SEND MAP	(3.2)
	SEND PAGE	(3.2)
	SEND TEXT	(3.2)
UNEXPIN	ISSUE ABORT	(3.5)
	ISSUE ADD	(3.5)
	ISSUE END	(3.5)
	ISSUE ERASE	(3.5)
	ISSUE NOTE	(3.5)
	ISSUE QUERY	(3.5)
	ISSUE RECEIVE	(3.5)
	ISSUE REPLACE	(3.5)
	ISSUE SEND	(3.5)
	ISSUE WAIT	(3.5)
	RECEIVE MAP	(3.2)
WRBRK	CONVERSE	(3.3)
	SEND CONTROL	(3.2)
	SEND	(3.3)
	SEND MAP	(3.2)
	SEND PAGE	(3.2)
	SEND TEXT	(3.2)
WRONGSTAT	SPOOLOPEN	(3.4)

Chapter 1.6. 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 ADDRESS and ASSIGN commands to access such information; these commands are described in the following sections.

Not all fields are intended to be accessed by the application program; see the *CICS/VS Application Programmer's Reference Manual (Macro Level),* SC33-0079 for a list of the fields that are part of the application programming interface (the API) and that will remain valid from release to release. Details of each control block and its fields are contained in the *CICS/MVS Data Areas* manual.

When using the ADDRESS and ASSIGN commands, the fields in the API can be read but should not be set or used in any other way. Do not use any of the CICS fields as arguments in CICS commands, because these fields may be altered by the EXEC interface modules.

INQUIRE/SET commands

The INQUIRE and SET commands allow application programs to access information about CICS resources. The application program can retrieve and modify information for CICS data sets, terminals, system entries, mode names, system attributes, programs, and transactions.

The commands are fully described in the *CICS/MVS Customization Guide.*

EXEC interface block (EIB)

In addition to the usual CICS control blocks, each task in a command-level environment has a control block called the EXEC interface block (EIB) associated with it. The field names and the data types of the fields in this control block are described in Appendix A, "EXEC interface block" on page 339.

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 will be helpful when a dump is being used to debug a program.

Access to CICS storage areas (ADDRESS)

```
ADDRESS
option(ptr-ref)...
```

This command is used to obtain access to any of the following areas: the EXEC interface block (EIB), the common system area (CSA), the common work area (CWA), the terminal control table user area (TCTUA), and the transaction work area (TWA). No more than four options can be specified in one ADDRESS command.

ADDRESS command options

Product-Sensitive Programming Interface

CSA
allows access to control blocks addressed by the CSA. The pointer reference is set to the address of the CSA. The CSA gives access to all fields in CICS control blocks and storage areas.

End of Product-Sensitive Programming Interface

CWA
is used to pass information between application programs. The pointer reference is set to the address of the CWA. If a CWA does not exist, the pointer reference is set to X'FF000000'.

EIB
is used to obtain addressability to the EXEC interface block in application routines other than the first invoked by CICS (for which addressability to the EIB is provided automatically).

TCTUA
is used to pass information between application programs, but only if the same terminal is associated with the application programs involved (which can be in different tasks). The pointer reference is set to the address of the TCTUA. If a TCTUA does not exist, the pointer reference is set to X'FF000000'. The data area contains the address of the TCTUA of the principal facility, not that for any alternate facility that may have been allocated.

TWA
is used to pass information between application programs but only if they are in the same task. The pointer reference is set to the address of the TWA. If a TWA does not exist, the pointer reference is set to X'FF000000'.

An example of the use of the ADDRESS command is given in the next section. (Information can also be passed between programs using the COMMAREA option of the program control commands, described in "Chapter 4.4. Program control" on page 289.)

If an ADDRESS command is included in a COBOL program that is to be compiled using the optimization feature, it must be followed by SERVICE RELOAD statements to reload the BLL cell being used. (The SERVICE RELOAD statement is described on page 23.)

Values outside the application program (ASSIGN)

```
ASSIGN
option(data-area)...

Condition: INVREQ
```

This command is used to obtain values outside the local environment of the application program. The value obtained is assigned to the data area specified in the option.

The following values can be obtained:

- Lengths of storage areas
- Values needed when communicating with the 2980 General Banking Terminal System (copied from the TCTTE)
- Values needed during basic mapping support (BMS) operations
- Values needed during batch data interchange
- Information on terminal characteristics, such as screen size and supported features (copied from the TCTTE)
- Other information that may be useful to the application programmer (copied from various CICS control blocks).

A complete list of ASSIGN command options is given at the end of the chapter. Up to 16 options can be specified in one ASSIGN command.

Remote transactions

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.

However, if the application-owning region (AOR) and the terminal-owning region (TOR) have different security requirements, security-related options **must** return different (but usually equivalent) values to the AOR and the TOR. The options affected are OPERKEYS, OPSECURITY, and USERID. For details, see the *CICS/MVS Intercommunication Guide*.

Examples of ADDRESS and ASSIGN commands

The following example shows, in the different application programming languages, how the ADDRESS command is used to obtain access to the TWA, and how the ASSIGN command is used to obtain the length of the TWA. Included is a test for validity; if there is no TWA, the ASSIGN TWALENG command will obtain a length of zero.

ASM

```
DSWORKA    DSECT
WAPTR      EQU   08
           USING DSWORKA,WAPTR
:
COUNT      DS    H
:
DFHEISTG   DSECT
TWALENG    DS    H
CODE       CSECT
           EXEC CICS ASSIGN *
           TWALENG(TWALENG)
           CLC   TWALENG,=H'0'
           BNH   CONTINUE
           EXEC CICS ADDRESS
           TWA(WAPTR)
           LH    6,COUNT
           LA    6,1(6)
           STH   6,COUNT
CONTINUE   DS    0H
```

COBOL

```
WORKING-STORAGE SECTION.
77  TWALENG PIC S9(4) COMP.

LINKAGE SECTION.
01 BLLCELLS.
   02 FILLER PIC S9(8) COMP.
   02 WAPTR PIC S9(8) COMP.
01 WORKAREA.
   02 COUNT PIC S9(4) COMP.

PROCEDURE DIVISION.
    EXEC CICS ASSIGN TWALENG
    (TWALENG) END-EXEC
    IF TWALENG GREATER THAN 0 THEN
    EXEC CICS ADDRESS TWA(WAPTR)
    END-EXEC
    ADD 1 TO COUNT.
```

PL/I

```
DCL  TWALENG FIXED BIN(15);
DCL  1 WORKAREA BASED(WAPTR),
       2 COUNT FIXED BIN(15);
       .
       .
EXEC CICS ASSIGN TWALENG(TWALENG);
IF TWALENG>0 THEN DO;
   EXEC CICS ADDRESS TWA(WAPTR);
COUNT=COUNT+1;
END;
```

ASSIGN command options

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.

ABCODE
specifies a variable that is set to the current value of the abend code (abend codes are documented in the *CICS/MVS Messages and Codes* manual). If an abend has not occurred, the variable is set to blanks. The format of the value is a 4-byte character string.

APPLID
returns the value specified in the APPLID operand of the DFHSIT system macro for the system owning the transaction. The format of the value is an 8-byte character string.

In a CICS XRF environment, the value returned is the generic APPLID. An application program is unaffected by a takeover from the active to the alternate.

BTRANS
specifies that the value required is an 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. The format of the value is a 1-byte character.

COLOR
specifies that the value required is an 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. The format of the value is a 1-byte character.

CWALENG
specifies that the length of the CWA is required. If no CWA exists, a zero length is returned. No exceptional condition occurs. The format of the value is halfword binary.

DELIMITER
specifies that the value required is the data-link control character for a 3600, copied from TCTTEDLM. If the task is not initiated from a terminal, INVREQ occurs. The format of the value is a 1-byte character.

DESTCOUNT
This option has two uses:

- Following a BMS ROUTE command, it specifies that the value required is the number of different terminal types in the route list, and hence the number of overflow control areas that may be required. A zero value is returned if fast-path BMS only is in use.
- Within BMS overflow processing, it specifies that the value required is the relative overflow control number of the destination that has encountered overflow. If this option is specified when overflow processing is not in effect, the value obtained will be meaningless. If no BMS commands have been issued, INVREQ occurs.

The format of the value is halfword binary.

See “Routing and page overflow” on page 189.

DESTID
specifies that the value required is the identifier of the outboard destination, padded with blanks on the right to 8 characters. If this option is specified before a batch data interchange command has been issued in the task, INVREQ occurs. The format of the value is an 8-byte character string.

DESTIDLENG
specifies that the value required is the length of the destination identifier obtained by DESTID. If this option is specified before a batch data interchange command has been issued in the task, INVREQ occurs. The format of the value is halfword binary.

EXTDS
specifies that the value required is an indicator showing whether you can use the query structured field (X'FF') or not (X'00'), to enquire what features the device can support. Further information on the query structured field is given in Chapter 5 of the *CICS/OS/VS IBM 3270 Datastream Device Guide*, SC33-0232.

If this option is specified and there is no TCTTE for the task, INVREQ occurs. The format of the value is a 1-byte character. Individual feature capabilities such as color and highlight have their own assign options.

FACILITY
specifies that the value required is the identifier of the facility that initiated the transaction. The value is copied from the first 4 bytes pointed at by TCAFCAAA. If this option is specified and there is no allocated facility, INVREQ occurs.

Note: Always use the QNAME option (described on page 55) to get the name of the transient data intrapartition queue whose trigger level caused the transaction to be initiated. The format of the value is a 4-byte character string.

FCI
specifies that the value required is the facility control indicator, copied from TCAFCI, that 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. No exceptional condition occurs. The format of the value is a 1-byte character.

GCHARS
specifies that the value required is the graphic character set global identifier (the GCSGID). The value is a number between 1 and 65,534 representing the set of graphic characters that can be input or output at the terminal. If the task is not initiated from a terminal, INVREQ occurs. The format of the value is halfword binary.

GCODES
specifies that the value required is the code page global identifier (the CPGID). The value is a number between 1 and 65,534 representing the EBCDIC code page defining the code points for the characters that can be input or output at the terminal. If the task is not initiated from a terminal, INVREQ occurs. The format of the value is halfword binary.

HILIGHT
specifies that the value required is an indicator showing whether the terminal is defined as having the extended highlight capability (X'FF') or not (X'00'). If this option is specified and there is no TCTTE for the task, INVREQ occurs. The format of the value is a 1-byte character.

INPARTN
specifies that the value required is the name of the most recent input partition. If the task is not initiated from a terminal, INVREQ occurs. The format of the value is a 1-or 2-character name.

KATAKANA
specifies whether the principal facility supports KATAKANA. If the task is not initiated from a terminal, INVREQ occurs. The format of the value is a 1-byte character.

LDCMNEM
specifies that the value required is the logical device code (LDC) mnemonic of the destination that has encountered overflow. If this option is specified when overflow processing is not in effect, the value obtained will be meaningless. If no BMS commands have been issued, INVREQ occurs. The format of the value is a 2-byte character string.

LDCNUM
specifies that the value required is the LDC numeric value of the destination that has encountered overflow. This indicates the type of the LDC, such as printer or console. If this option is specified when overflow processing is not in effect, the value obtained will be meaningless. No exceptional condition occurs. The format of the value is a 1-byte character.

MAPCOLUMN
specifies that the value required is the 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 BMS routing is in effect, INVREQ occurs. The format of the value is halfword binary.

MAPHEIGHT
specifies that the value required is the height of the most recently positioned map. If no map has yet been positioned, or if BMS routing is in effect, or if the task is not initiated from a terminal, INVREQ occurs. A zero value is returned if fast-path BMS only is in use. The format of the value is halfword binary.

MAPLINE
specifies that the value required is the 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 BMS routing is in effect, or if the task is not initiated from a terminal, INVREQ occurs. A zero value is returned if fast-path BMS only is in use. The format of the value is halfword binary.

MAPWIDTH
specifies that the value required is the width of the most recently positioned map. If no map has yet been positioned, or if BMS routing is in effect, or if the task is not initiated from a terminal, INVREQ occurs. A zero value is returned if fast-path BMS only is in use. The format of the value is halfword binary.

MSRCONTROL
specifies that the value required is an indicator showing whether the terminal supports magnetic slot reader (MSR) control (X'FF') or not (X'00'). If the task is not initiated from a terminal, INVREQ occurs. The format of the value is a 1-byte character.

NETNAME
specifies that the value required is the name of the logical unit in the VTAM network. The format of the value is an 8-byte character string. If the principal facility is not a local terminal, the value returned is a null string.

If the task is not initiated from a terminal, INVREQ occurs.

NUMTAB
specifies that the value required is the number of the tabs required to position the print element in the correct passbook area of the 2980. If this option is

specified and there is no TCTTE for the task, INVREQ occurs. The format of the value is a 1-byte character.

OPCLASS
specifies that the value required is the operator class, copied from TCTTEOCL. If this option is specified and there is no TCTTE for the task, INVREQ occurs. The format of the value is a 3-byte character string.

OPERKEYS
specifies that the value required is the 8 bytes representing the transaction security keys, copied from fields in the TCTTE. For a remote terminal, the value returned may, for security reasons, be forced to the default value of 1 (see the *CICS/MVS Intercommunication Guide*). If this option is specified and there is no TCTTE for the task, INVREQ occurs. The format of the value is an 8-byte character string.

OPID
specifies that the value required is the operator identification, copied from TCTTEOI. If this option is specified and there is no TCTTE for the task, INVREQ occurs. The format of the value is a 3-byte character string.

OPSECURITY
specifies that the value required is the first 3 bytes of the field returned by the OPERKEYS option. This option is provided to maintain compatibility with previous releases.

OUTLINE
specifies that the value required is an 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. The format of the value is a 1-byte character.

PAGENUM
specifies that the value required is the current page number for the destination that has encountered an overflow. If this option is specified when overflow processing is not in effect, the value obtained will be meaningless. A zero value is returned if fast-path BMS only is in use. If no BMS commands have been issued, INVREQ occurs. The format of the value is halfword binary.

PARTNPAGE
specifies that the value required is the name of the partition that most recently caused page overflow. A blank value is returned if partitions are not in use, or if fast-path BMS is in use. If no BMS commands have been issued, INVREQ occurs. The format of the value is a 1-or 2-character name.

PARTNS
specifies that the value required is an indicator showing whether the terminal supports partitions (X'FF') or not (X'00'). If the task is not initiated from a terminal, INVREQ occurs. The format of the value is a 1-byte character.

PARTNSET
specifies that the value required is the name of the application partition set. A blank value is returned if there is no application partition set. If the task is not initiated from a terminal, INVREQ occurs. The format of the value is a 1-through 6-character name.

PRINSYSID
applies only when the principal facility is one of the following:

- An MRO session to another CICS system
- An LU6.1 session to another CICS or IMS system
- An LU6.2 (APPC) session to another CICS system, or to another APPC system or device.

PRINSYSID specifies that the value required is the name by which the other system is known in the local system; that is, the CONNECTION definition that defines the other system. For a single-session APPC device defined by a terminal definition, the returned value is the terminal identifier (TRMIDNT).

If the principal facility is not an MRO, LU6.1, or LU6.2 session, or if the task has no principal facility, INVREQ occurs.

The format of the value is a 4-byte character string.

Note: 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/MVS Intercommunication Guide*).

PS specifies that the value required is an indicator showing whether the terminal is defined as having the programmed symbols capability (X'FF') or not (X'00'). If this option is specified and there is no TCTTE for the task, INVREQ occurs. The format of the value is a 1-byte character.

QNAME
specifies that the value required is the 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 task initiation (ATI), INVREQ occurs. The format of the value is a 4-byte character string.

RESTART
specifies that the value required is an indicator showing whether a restart of the task (X'FF'), as opposed to a normal start of the task (X'00'), has occurred.

SCRNHT
specifies that the value required is the height of the 3270 screen defined for the current transaction. If this option is specified and there is no TCTTE for the task, INVREQ occurs. The format of the value is halfword binary.

SCRNWD

specifies that the value required is the width of the 3270 screen defined for the current transaction. If this option is specified and there is no TCTTE for the task, INVREQ occurs. The format of the value is halfword binary.

SIGDATA

specifies that the value required is the signal data received from a logical unit, copied from TCTESIDI. If this option is specified and there is no TCTTE for the task, INVREQ occurs. The format of the value is a 4-byte character string.

SOSI

specifies that the value required is an indicator showing whether the terminal is defined as having the mixed EBCDIC/DBCS fields capability (X'FF') or not (X'00'). The DBCS subfields within an EBCDIC field are delimited by SO (shift out) and SI (shift in) characters. If the task is not initiated from a terminal, INVREQ occurs. The format of the value is a 1-byte character.

STARTCODE

specifies that the value required is a code indicating how a transaction has been started. The format of the value is a 2-byte character string that can have the following values:

Code	Tx started by
QD	Transient data trigger level
S	START command (no data)
SD	START command (with data)
TD	Terminal input
U	User-attached task

STATIONID

specifies that the value required is the station identifier of a 2980. If this option is specified and there is no TCTTE for the task, INVREQ occurs. The format of the value is a 1-byte character.

SYSID

specifies that the value required is the name given to the local CICS system. 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. The format of the value is a 4-byte character string.

TCTUALENG

specifies that the value required is the length of the terminal control table user area (TCTUA). If no TCTUA exists, a zero length is returned. No exceptional condition occurs. The format of the value is halfword binary.

TELLERID

specifies that the value required is the teller identifier of a 2980. If this option is specified and there is no TCTTE for the task, the INVREQ condition occurs. The format of the value is a 1-byte character.

TERMCODE

specifies that the value required is a code giving the type and model number of the terminal associated with the task, copied from TCTTETT and TCTTETM. If the code returned in TCTTETT is TCTELU6, an EXEC CICS INQUIRE CONNECTION command can be executed to determine if this ISC session is using LU61 or APPC protocols. If this option is specified and there is no TCTTE for the task, INVREQ occurs. The format of the value is a 2-byte character string.

TWALENG

specifies that the value required is the length of the transaction work area (TWA). If no TWA exists, a zero length is returned. No exceptional condition occurs. The format of the value is halfword binary.

UNATTEND

specifies that the value required is a code indicating that the mode of operation of the terminal is unattended (X'FF') or attended (X'00'), copied from TCTEMOP. If this option is specified and there is no TCTTE for the task, INVREQ occurs. The format of the value is a 1-byte character.

USERID

specifies that the value required is the user identifier of whoever is signed on. This option will return a blank string when there is no user identifier. If the task is not initiated from a terminal, INVREQ occurs. The format of the value is an 8-byte character string. For a remote terminal, the value returned may, for security reasons, be blanks (see the *CICS/MVS Intercommunication Guide*).

VALIDATION

specifies that the value required is an indicator showing whether the terminal is defined as having the validation capability (X'FF') consisting of the mandatory fill, mandatory enter, and trigger attributes. No validation capability is indicated by X'00'. If this option is specified and there is no TCTTE for the task, INVREQ occurs. The format of the value is a 1-byte character.

Chapter 1.7. Execution (command level) diagnostic facility

The execution (command level) diagnostic facility (EDF) enables you to test a command level application program online without modifying the program or the program preparation procedure. EDF intercepts execution of the application program at various points and displays information about it at these points. Also displayed are any screens sent by the application program, so that you can converse with the application program during testing just as a user would on the production system.

EDF can only be used to test user application programs; it cannot be used for system transactions that use the command level interface. User application programs that are to be debugged using EDF must be assembled or compiled with the EDF translator option, which is the default. If you specify NOEDF, the program cannot be debugged using EDF.

EDF runs as a CICS transaction. You start it by a transaction identifier (CEDF) or by a PF key named in the program control table (PCT). You must also ensure that the programs and maps that are used by EDF are specified in the processing program table (PPT). EDF uses temporary storage and BMS. You can use EDF only from a 3270 terminal that has a screen width of 80 columns or more and a screen depth of 24 lines or more.

EDF is a command level diagnostic aid only, and unpredictable results may occur if macros are coded, or the terminal control table (TCT) is in application programs monitored by EDF.

TCAM (a data stream access method) is supported by EDF, but only in dual screen mode, and provided that the terminals are not pooled.

VM PASSTHRU is not supported by EDF when testing in single screen mode.

When the CEDF initialization screen is being displayed and a COBOL program is the first to be executed, this program is locked and any attempt to use it will cause a wait until CEDF initialization is complete. At this time, if the task is abended the program will remain locked until a master terminal NEWCOPY command is executed to unlock it.

When using single screen mode with CEDF, automatic message journaling should not be specified for CEDF or for the user transaction.

If you want to test an application program that uses partitions, or which does its own request unit (RU) chaining (RU chaining is described in "Chaining of input data" on page 226), you must run EDF on a terminal other than the terminal on which that application program is executing. In other words, EDF must be used in dual screen mode as described in "Invoking EDF" on page 58.

If a SEND LAST command is issued, EDF is terminated before the command is executed.

Functions of EDF

During execution of a transaction in debug mode, EDF intercepts the execution of the application program at the following points:

1. At transaction initialization:

   After the EXEC interface block (EIB) has been initialized, but before the application program is given control.

2. At the start of the execution of every EXEC CICS and EXEC DLI command:

   After the initial trace entry has been made, but before the requested action has been performed.

3. At the end of the execution of every command (except ABEND, XCTL, and RETURN):

   After the requested action has been performed, but before the HANDLE CONDITION mechanism is invoked and before the response trace entry is made.

4. At program termination.
5. At normal task termination.
6. When an ABEND occurs.
7. At abnormal task termination.

At all the above points of interception, EDF displays the current status, by identifying the cause of interception. In addition:

- At point 1, EDF displays the contents of the fields in the EIB.
- At point 2, EDF displays the command, including keywords, options, and argument values. The command is identified by transaction identifier, program name, the hexadecimal offset within the program and, if the program has been translated with the DEBUG translator option, the line number of the command as given in the translator source listing.
- At point 3, EDF displays the same as at point 2, plus the response from command execution.
- At points 6 and 7, EDF displays the values of the fields in the EIB and the following items:
  - The abend code
  - If the abend code is ASRA (that is, a program interrupt has occurred), the PSW at the time of interrupt, and the source of the interrupt as indicated by the PSW

- If the PSW indicates that the instruction giving rise to the interrupt is within the application program, the offset of that instruction.

You can also display any of the following:

- The values of the fields in the EXEC interface block (EIB) and the DL/I interface block (DIB).
- The program's working storage in hexadecimal and character form.
- The last ten displays, including all argument values, responses, and so on.
- The contents (in hexadecimal) of any address location within the CICS region.

At any of these points of interception, you can interact with the application program in the following ways:

- If the current command is being displayed before it is executed, you can modify any argument value by overtyping the value that is displayed on the screen. Alternatively, you can suppress execution of the command (that is, convert it to a null operation), but you cannot add or delete options.
- If the current command is being displayed after it has been executed, you can modify certain argument values and the response code by overtyping the displayed value or response with the required value or response.
- You can modify the program's working storage and most fields of the EIB and DIB.
- You can request a display of the contents of any temporary storage queue.
- You can switch off debug mode (except at point 2) and continue running the application normally. Alternatively, you can force an abend.
- You can request command displays to be suppressed until one or more of a set of specific conditions is fulfilled. These conditions are:
  - A specific named command is encountered.
  - Any exceptional condition occurs for which the system action is to raise ERROR.
  - A specific exceptional condition occurs.
  - The command at a specific offset or on a specific line number (assuming the program had been translated with the DEBUG option) is encountered.
  - An abend occurs.
  - The task terminates normally.
  - The task terminates abnormally.
  - Any DL/I error status occurs.
  - A specific DL/I error status occurs.

Security rules

If a security key has been defined for EDF, the user transaction must have the same type of security. If the security is external, the transaction must be defined to that security manager.

To invoke EDF, you must have a security key that matches the security key defined for EDF in the PCT. In addition, to test a particular transaction, you must have a security key that matches the security key for that transaction. If this condition is not satisfied, the EDF session is terminated immediately.

By default, resource level security checks will be made during execution of the transaction under test unless EDF has been redefined as not requiring these checks. If such checks indicate that you are not allowed access to the resource, your transaction will raise the NOTAUTH condition. Unless you allow for this by means of a HANDLE CONDITION command, your transaction will be abended.

Installing EDF

To ensure that EDF is available on the test system, the system programmer must make one group entry in the PCT and one group entry in the PPT (see either the *CICS/MVS Resource Definition (Online)* manual or the *CICS/MVS Resource Definition (Macro)* manual for details of constructing a PCT and PPT).

EDF can send messages greater than 4K bytes in length. If you are using VTAM, ensure that your NCP (network control program) can handle data of this length.

Invoking 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"). You cannot use single screen mode if the transaction to be tested makes use of extended attributes or partitions.

You start EDF in single screen mode either by:

- Entering transaction code CEDF or
- Pressing the appropriate PF key (if one has been defined for EDF).

Next, you start the transaction to be tested by:

- Pressing the CLEAR key to clear the screen
- Entering the transaction code of the transaction to be tested.

You start EDF in dual screen mode by entering:

```
CEDF xxxx
```

on the current terminal. This terminal must be in TRANSCEIVE status (that is, it can both send and receive data).

Here "xxxx" is the 4-character identifier of the terminal (termid) on which the transaction to be tested is being run. (This identifier is as defined in the TRMIDNT operand of the DFHTCT TYPE=TERMINAL system macro.)

If a command level transaction is already running on that terminal, EDF will associate itself with that transaction; otherwise it will associate itself with the next command level transaction started at that terminal.

The above also applies to a single system. If the transaction running on the terminal has been transaction routed, EDF will not associate itself with it, nor with any other transaction that has been routed. EDF will associate itself with the next command level transaction that runs on the system to which the terminal is connected.

You must include the identifier of the session (sessionid) when you want to test a transaction that is attached across an MRO or LU6.1 session. Alternatively, you must provide the sessionid to the system on which the attached transaction is running. All CICS commands executed by the attached transaction will be tested.

You can include the identifier of the system (sysid) when you want to test transactions attached across LU6.2 sessions. In this case, EDF will associate itself with the first transaction attached across an LU6.2 session belonging to the specified system.

You can enter CEDF from a formatted screen. The effect is the same as if you had pressed the PF key, that is, the terminal at which CEDF is entered is put into EDF mode. (No message is issued, so the formatted screen remains intact.)

The full format of the command to initiate or terminate an EDF session is:

```
CEDF [termid|sysid|sessionid]
[,ON|,OFF]
```

If you omit the terminal identifier, the terminal at which the CEDF transaction is initiated is assumed.

You cannot define CEDF to be a remote transaction. The only way to test a transaction running in a connected system is by means of the CRTE routing transaction. You use CRTE to set up a routing session with the connected system. You can then use your terminal in single screen mode, entering CEDF to invoke EDF within the routing session. You cannot use PA or PF keys in a routing session.

You cannot use EDF in dual-screen mode if the transaction under test, or the terminal that invokes it, is owned by a different system.

EDF displays

An example of a typical EDF display is given in Figure 12 on page 60. The five lines at the foot of the screen provide a menu indicating the effect of the ENTER and PF keys for that particular display. If the terminal does not have PF keys, the same effect can be obtained by positioning the cursor under the required instruction on the screen and pressing the ENTER key. The cursor can be correctly positioned by using the tab keys.

Although the menu may change from one display to another, no function will move from one key to another as a result of a menu change.

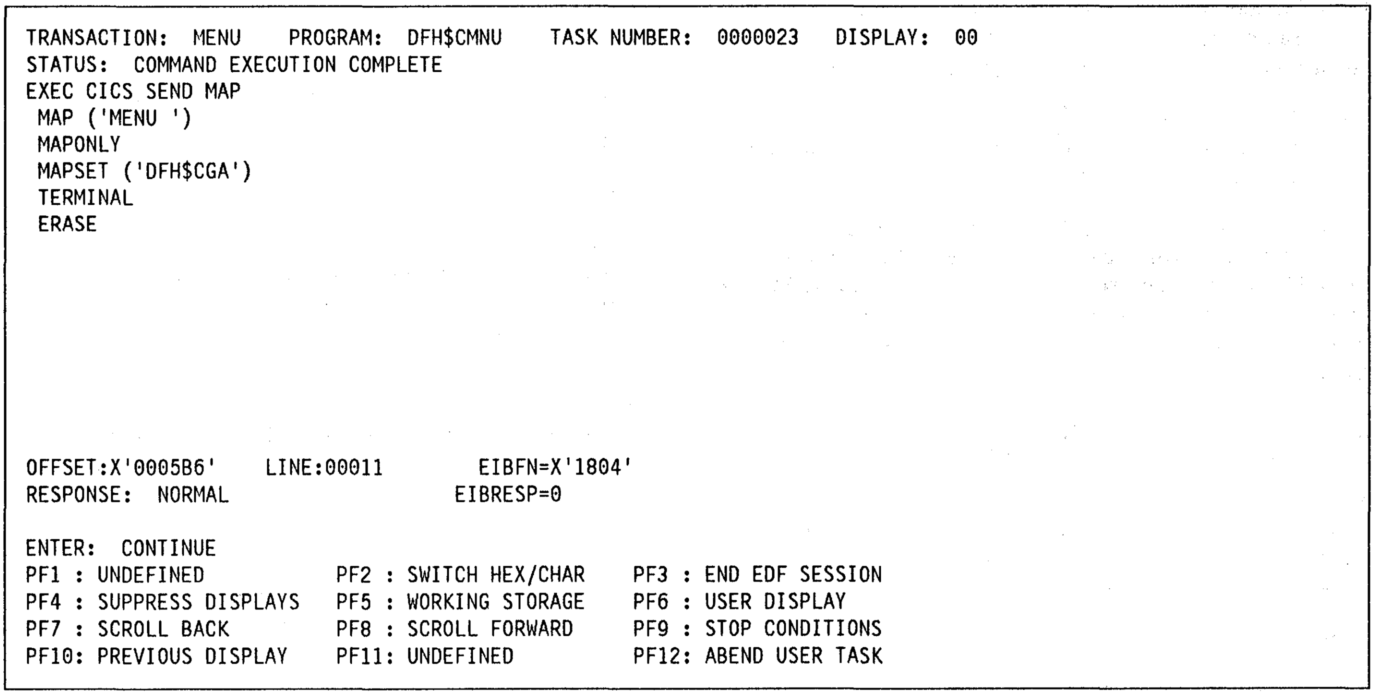
If the ENTER key is pressed while the cursor is not positioned within the menu, the function specified for the ENTER key is performed.

EDF uses the line immediately above the menu to display messages to the user.

Up to ten displays are remembered and can be redisplayed later. The number at the top right of the screen indicates the current display number; it is possible to recall any of the last ten displays, which are numbered −01, −02, and so on, by overtyping this number. Alternatively, PF10 and PF11 can be used to step back and forward one display at a time. PF10 and PF11 become undefined if there are no further displays backward or forward respectively.

Argument values can be displayed in character or hexadecimal format. If character format is requested, numeric arguments are shown in signed numeric character format. Each argument value is restricted to one line of the display; if the value is too long, only the first few bytes are displayed, followed by "..." to indicate that the value is incomplete. If the argument is displayed in hexadecimal format, the address of the argument is also displayed. This enables the user to display the argument value in full by requesting a display of that location and scrolling if necessary.

The user can overtype any screen area at which the cursor stops when the tabbing keys are pressed, such as the response field. For example, the response can be changed from "NORMAL" to "ERROR" or some other exceptional condition, so as to test the program's error handling at this point in the program. A list of areas that can be overtyped is given later under "Overtyping EDF displays" on page 63.



```
TRANSACTION:  MENU    PROGRAM:  DFH$CMNU    TASK NUMBER:  0000023   DISPLAY:  00
STATUS:  COMMAND EXECUTION COMPLETE
EXEC CICS SEND MAP
 MAP ('MENU ')
 MAPONLY
 MAPSET ('DFH$CGA')
 TERMINAL
 ERASE









OFFSET:X'0005B6'    LINE:00011        EIBFN=X'1804'
RESPONSE:  NORMAL                   EIBRESP=0

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: ABEND USER TASK
```

Figure 12. Typical EDF display

The response of EDF to a user request is in accordance with the following order of priority:

1. If the CLEAR key is used, EDF redisplays the screen with any changes ignored.
2. If invalid changes are made, EDF accepts any valid changes and redisplays the screen with a diagnostic message.
3. If the display number is changed, EDF accepts any other changes and displays the requested display.
4. If a PF key is used, or the ENTER key is pressed when the cursor is in the PF key definition area, EDF accepts any changes and performs the action requested by the PF key.
5. If the ENTER key is pressed while the cursor is not in the PF key definition area, and the screen has been modified (other than the REPLY field), EDF redisplays the screen with changes included.
6. If the ENTER key is pressed while the cursor is not in the PF key definition area, and the screen has not been modified (other than the REPLY field) and if the ENTER key means CONTINUE, execution of the user transaction continues, otherwise if the ENTER key means CURRENT DISPLAY, EDF redisplays the current status display.

Terminal sharing between transaction and EDF

When both EDF and the user transaction are sharing the same terminal, EDF restores the user transaction's display at the following times:

- When the transaction requires input from the operator
- When the transaction's display is changed
- At the end of the transaction
- When EDF displays are suppressed
- When USER DISPLAY is requested.

When a SEND command is followed by a RECEIVE command, the display sent by the SEND command appears twice; once when the SEND command is executed, and again when the RECEIVE command is executed. It is not necessary to respond to the SEND command but, if a response is made, EDF will remember it and redisplay it when the screen is restored for the RECEIVE command. The response passed to the transaction is that which is made to the RECEIVE command.

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 will be seen when the SEND command is executed, but not when the screen is restored.

For same terminal use, when EDF restores the transaction display on a device that uses color, programmed symbols, or extended highlighting, the attributes will no longer be present and the display will be in monochrome with no programmed symbols, or extended highlighting.

If the inbound reply mode in the application program is set to character (to enable the attribute setting keys), EDF will reset this mode causing these keys to be disabled.

When EDF restores the transaction display, it locks the keyboard until the transaction issues a RECEIVE command, at which time EDF frees the keyboard.

If the EDF session is terminated part way through the transaction, EDF restores the screen with the keyboard locked if the last send/receive to the terminal is a RECEIVE command; otherwise, the keyboard is unlocked. This will usually, but not always, match the normal behavior of the transaction.

Program function (PF) keys

The following list explains the meanings of the program function (PF) key settings. Where a terminal has 24 PF keys, EDF treats PF13 through PF24 as duplicates of PF1 through PF12 respectively.

ABEND USER TASK
terminates the task. 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. If 'NO' is entered, the task will be abended without a dump.

Abend codes beginning with the character A are reserved for use by CICS. Use of a CICS abend code may cause unpredictable results.

This function cannot be used 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. The queue name can be changed by using CEBR commands. The CEBR transaction is described in "Chapter 1.9. Temporary storage browse" on page 73.

CONTINUE
causes the user transaction to continue unless the screen has been modified. In the latter case, EDF redisplays the screen with changes incorporated.

CURRENT DISPLAY
displays the screen that was being displayed before the user started examining other displays, such as remembered displays, unless the screen has been modified. In the latter case, EDF redisplays the screen with changes incorporated.

DIB DISPLAY
shows the contents of the DIB; see "DL/I interface block (DIB)" on page 105 for a description of the fields in the DIB.

EIB DISPLAY
shows the contents of the EIB and COMMAREA (if any); see Appendix A, "EXEC interface block" on page 339 for a description of the fields in the EIB.

END EDF SESSION
ends the debugging session, and takes the terminal out of debug mode. The user transaction continues.

NEXT DISPLAY
used when examining displays, to step on to the next remembered display. Repeated use stops at the current display, when the 'next display' key is no longer available.

PREVIOUS DISPLAY
shows the latest remembered display. Repeated use stops at the earliest remembered display. Further use merely causes the earliest remembered display to be redisplayed.

REGISTERS AT ABEND
displays storage containing the values of the registers in the event of an ASRA abend. The layout of the storage is as follows:

- PSW at abend (8 bytes)
- Register values (0 through 15).

In some (very rare) cases, when a second program check occurs in the system before EDF has captured the values of the registers, this function will not appear on the menu of the abend display. If this happens, a second test run will generally prove to be more informative.

REMEMBER DISPLAY
places a display that would not normally be remembered, such as an EIB display, in the memory. (Normally, only the command displays are remembered.) The memory can hold up to ten displays. All pages associated with the display are remembered (and can be scrolled when recalled) except for storage displays where only the page currently displayed is remembered.

SCROLL BACK
scrolls a command or EIB display backward. A plus sign (+) against the first option or field indicates there are more options or fields preceding.

SCROLL BACK FULL
scrolls a working storage display a full screen backward, displaying lower addresses.

SCROLL BACK HALF
scrolls a working storage display half a screen backward, displaying lower addresses.

SCROLL FORWARD
scrolls a command or EIB display forward. A plus sign (+) against the last option or field indicates there are more options or fields following.

SCROLL FORWARD HALF
scrolls a working storage display half a screen forward, displaying higher addresses.

SCROLL FORWARD FULL
scrolls a working storage display a full screen forward, displaying higher addresses.

STOP CONDITIONS
displays, as shown in Figure 13 on page 63, a skeleton menu with which the user can specify one or more conditions that will cause EDF to stop the user transaction, and start redisplaying commands, after displays have been suppressed by the SUPPRESS DISPLAYS function.

These functions are used to reduce the amount of operator intervention required to check out a program that is partly working.

The transaction can be stopped:

- When a specified type of command is reached.
- When a specified exceptional or error condition occurs during execution of a command.
- When a specified offset or line is reached.
- At transaction abend.
- At normal task termination.
- At abnormal task termination.

The line number, which will be available on the source listing if the program has been translated using the DEBUG translator option, must be specified exactly as it appears on the listing, including leading zeros, and must be the line on which a command starts.

The offset specified must be the offset of the BALR instruction corresponding to the command.

The correct line can be determined easily from the translator output listing. The offset can be determined from the code listing produced by the assembler or compiler.

For transactions that contain DLI commands, the qualifier CICS on the command line can be overtyped with DLI to specify a DLI command. Also, the transaction can be stopped when a specified error status, or any error status, occurs.

DL/I error status codes can be entered only when the DLI command (not a CICS command) is about to be executed. The transaction can be stopped when a specified DL/I error status code is entered. For a list of valid status codes that may be entered in this 2-byte field, see page 105.

SUPPRESS DISPLAYS
suppresses all EDF displays until the next stop condition occurs.

SWITCH HEX/CHAR
switches the display between hexadecimal and character representation. This is a mode switch; subsequent displays will stay in the chosen mode until the next time this key is pressed. This switch has no effect on previously remembered displays, stop condition displays, and working storage displays.

UNDEFINED
means that this key is not available with this type of display.

USER DISPLAY
shows what the user would see if the terminal was not in EDF mode. Hence, this function is usable only for same terminal checkout.

WORKING STORAGE
displays the program's working storage, in a form similar to that of a dump listing, that is, in both hexadecimal and character representation. When this key is used, two additional scrolling keys are provided, and other PF keys allow the EIB (and the DIB if a DL/I command has been processed by EDF) to be displayed.

The meaning of "working storage" depends on the programming language of the application program, as follows:

ASM
The storage defined in the current DFHEISTG DSECT.

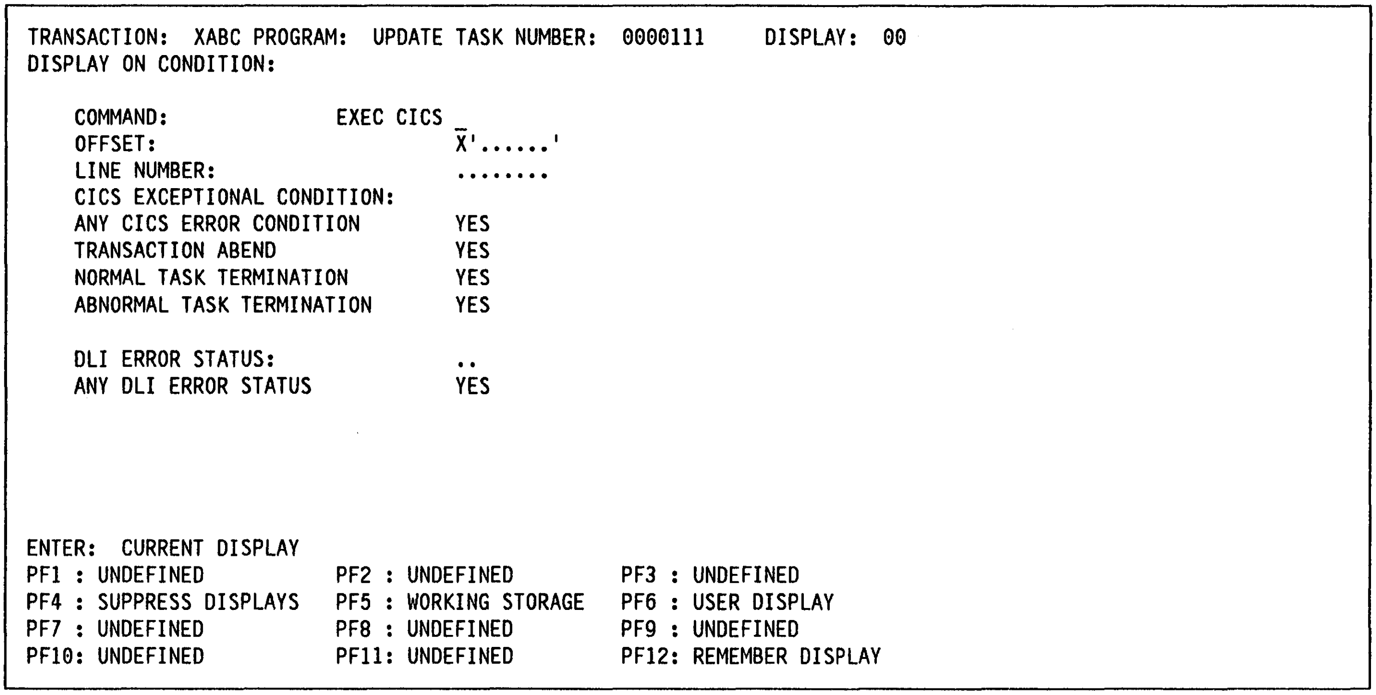
COBOL
All data storage defined in the working-storage section of the program.

PL/I
The dynamic storage area (DSA) of the current procedure.

Except for COBOL programs, working storage starts with a standard-format save area, that is, registers 14-12 are stored at offset 12 and register 13 at offset 4.

Working storage can be changed at the screen; either the hexadecimal section or the character section may be used. Also, the ADDRESS field at the head of the display can be overtyped with a hexadecimal address; storage starting at that address will then be displayed when ENTER is pressed. This allows any location in the partition to be examined. Further information on the use of overtyping is given under "Overtyping EDF displays" on page 63.

If the storage examined is not part of the user's working storage (which is unique to the particular transaction under test), the corresponding field on the screen is inhibited to prevent the user from overwriting storage that can affect more than one task in the program.



```
TRANSACTION:  XABC PROGRAM:  UPDATE TASK NUMBER:  0000111     DISPLAY:  00
DISPLAY ON CONDITION:

    COMMAND:              EXEC CICS _
    OFFSET:                         X'......'
    LINE NUMBER:                    ........
    CICS EXCEPTIONAL CONDITION:
    ANY CICS ERROR CONDITION        YES
    TRANSACTION ABEND               YES
    NORMAL TASK TERMINATION         YES
    ABNORMAL TASK TERMINATION       YES

    DLI ERROR STATUS:               ..
    ANY DLI ERROR STATUS            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 13. "Stop-Conditions" display

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.

At the beginning and end of a task, working storage is not available. In these circumstances, EDF generates a blank storage display so that the user can still examine any storage area in the region by overtyping the address field.

Overtyping EDF displays

As mentioned above, certain areas of an EDF display can be overtyped. These areas can be identified by use of the tab keys; the cursor stops only at fields that can be overtyped (excluding fields within the menu).

- Any command can be overtyped with 'NOOP' or 'NOP' before execution; this suppresses execution of the command. Use of the ERASE EOF key, or overtyping with blanks, will give the same effect. When the screen is redisplayed with NOOP, the original verb line can be restored by erasing the whole verb line with the ERASE EOF key.
- 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. Overtyping must not extend beyond the argument value displayed. Any modification that is not overtyping of the displayed value is ignored (no diagnostic message being generated). When an argument is displayed in hexadecimal format, the address of the argument location is also displayed.
- Numeric values always have a sign field, which can be overtyped with a minus or a blank only.
- The response field can be overtyped with the name of any exceptional 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 will take whatever action has been prescribed for the specified response.
- The EIBRESP field can be overtyped with any desired hexadecimal value when it is displayed as part of the EXEC interface block, but not when it is part of a command display.

When a field representing a data area of a program is overtyped, the entered value is placed directly into the application program's storage. On the other hand, before execution of a command, when a field representing a data value (which may possibly be a constant) is overtyped, a copy of the field is used; thus, other parts of the program that might use the same constant for some unrelated purpose will not be affected by the change. If, for example, the map name is overtyped before executing a SEND MAP command, the map actually used temporarily is the map with the entered name; but the map name displayed on response will be the original map name. (The 'previous display' key can be used to display the map name actually used.)

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 by a period. When overtyping a period, the user must be aware that the storage may in fact contain a character other than a period. The user may not overtype any character with a period; if this is done, the change is ignored and no diagnostic message is issued. Similarly, when a value is displayed in hexadecimal format, overtyping with a blank character is ignored and no diagnostic message is issued.

When storage is displayed in both character and hexadecimal format and changes are made to both, the value of the hexadecimal field will take precedence should the changes conflict; no diagnostic message is issued.

If invalid data is entered, the result is as follows, regardless of the action requested by the user:

- The invalid data is ignored;
- A diagnostic message is displayed;
- The alarm is sounded if the terminal has the alarm feature.

EDF does not translate lowercase characters to uppercase. If uppercase translation is not specified for the terminal in use, the user must take care to enter only uppercase characters.

Checking pseudoconversational programs

On termination of the task, EDF displays a message saying that the task is terminated and prompting the user to specify whether or not debug mode is to continue into the next task. This is to allow realistic debugging of pseudoconversational programs. If the terminal came out of debug mode between the tasks involved, each task would start with fresh EDF settings, and the user would not be able, for example, to display screens remembered from previous tasks.

Program labels

Some commands, such as HANDLE CONDITION, require the user to specify a program label. The form of the display program labels depends on the programming language in use:

- For assembler language, the offset of the program label is displayed; for example, ERROR (X'00030C')
- For COBOL, a null argument is displayed: for example, ERROR ( )
- For PL/I, the address of the label constant is displayed; for example, ERROR (X'001D0016').

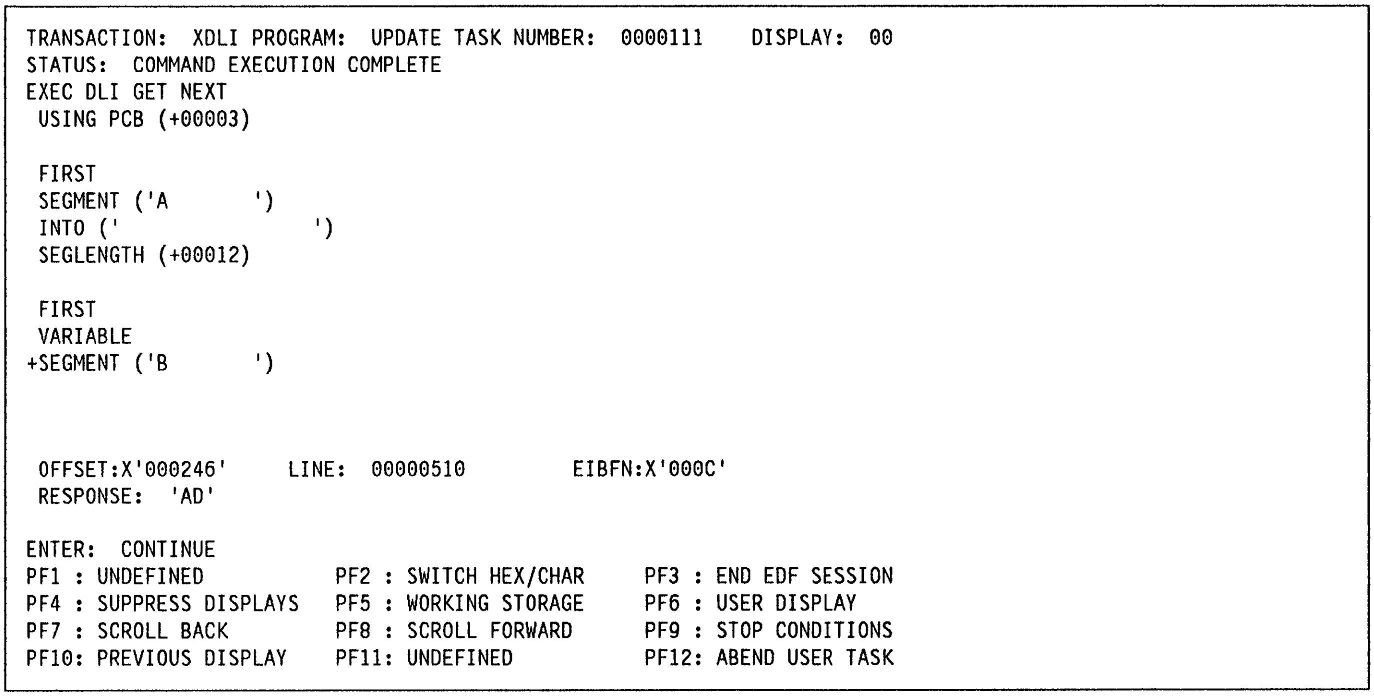
If no label value is specified on a HANDLE CONDITION command, EDF displays the condition name alone.

EDF and EXEC DLI commands

EDF supports EXEC DLI commands in the same way as it supports EXEC CICS commands. However, the following minor differences should be noted:

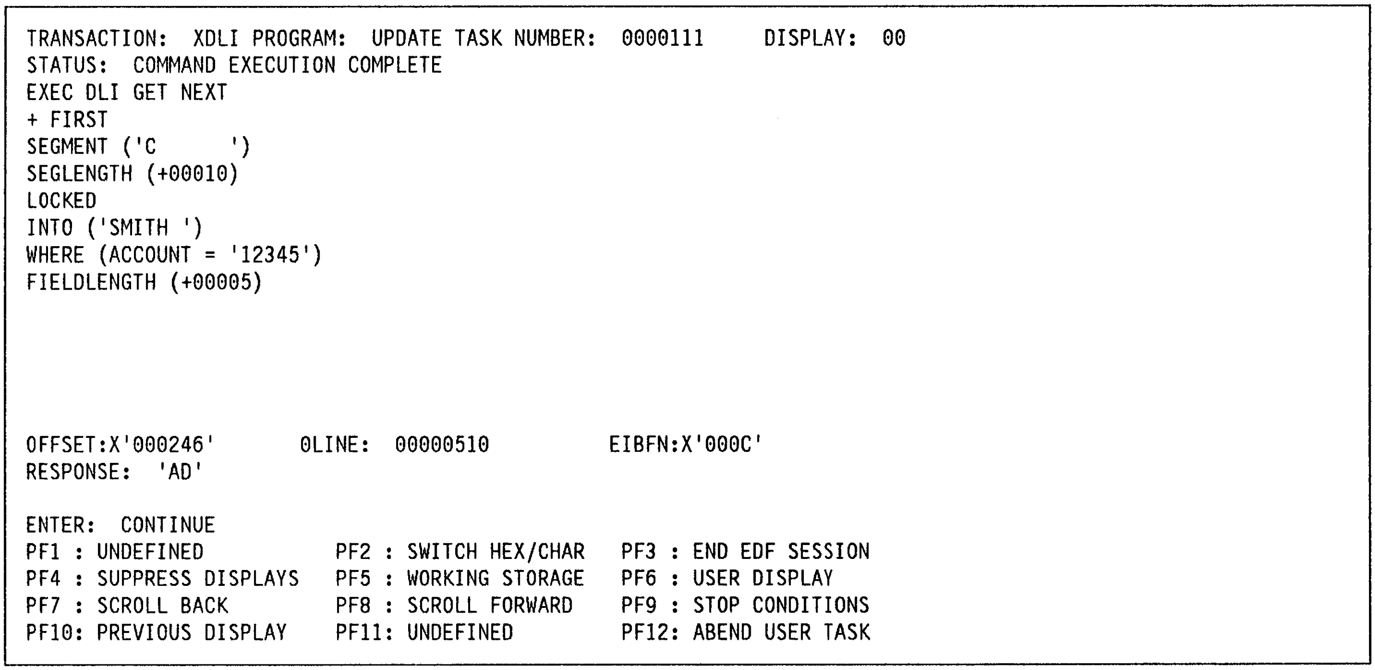
- The two-character DL/I status code appears in the RESPONSE field and the EIBRCODE field is not displayed. The status code can be displayed in character or hexadecimal format. If the status code is changed to an invalid value, or to a value that would have caused DL/I to abend the user task, a warning message is issued before continuing the user task.
- For commands that generate more than one CALL statement, the offset is that of the last CALL.
- For the WHERE option, only the keyfield value (the component following each comparison operator) can be converted to hexadecimal. The address shown for this option is that of the keyfield value. All the components of a WHERE option, including comparison and boolean operators, can be overtyped.
- For transactions that contain EXEC DLI commands, the DL/I interface block can be displayed, and additional stop conditions can be specified.

Examples of typical displays for an EXEC DLI command are given in Figure 14 on page 65, and in Figure 15 on page 65.



```
TRANSACTION:  XDLI PROGRAM:  UPDATE TASK NUMBER:  0000111    DISPLAY:  00
STATUS:  COMMAND EXECUTION COMPLETE
EXEC DLI GET NEXT
 USING PCB (+00003)

 FIRST
 SEGMENT ('A       ')
 INTO ('                ')
 SEGLENGTH (+00012)

 FIRST
 VARIABLE
+SEGMENT ('B       ')



 OFFSET:X'000246'     LINE:  00000510        EIBFN:X'000C'
 RESPONSE:  'AD'

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: ABEND USER TASK
```

Figure 14. First page of typical EXEC DLI display



```
TRANSACTION:  XDLI PROGRAM:  UPDATE TASK NUMBER:  0000111     DISPLAY:  00
STATUS:  COMMAND EXECUTION COMPLETE
EXEC DLI GET NEXT
+ FIRST
SEGMENT ('C      ')
SEGLENGTH (+00010)
LOCKED
INTO ('SMITH ')
WHERE (ACCOUNT = '12345')
FIELDLENGTH (+00005)





OFFSET:X'000246'      0LINE:  00000510         EIBFN:X'000C'
RESPONSE:  'AD'

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: ABEND USER TASK
```

Figure 15. Second page of typical EXEC DLI display

Chapter 1.8. Command level interpreter

The command level interpreter enables CICS commands to be entered, syntax-checked, and executed interactively at a 3270 terminal. The interpreter performs a dual role in the operation of a CICS system:

- For the application programmer, it provides a reference to the syntax of the whole of the CICS command level application programming interface (excluding DL/I). Most of the commands can be carried through to execution, and the results of execution can be displayed. However, the interpreter cannot be used to execute commands that refer to partitions. This is because the display cannot be restored after the screen has been partitioned.
- For the system programmer, it provides a means of interaction with the system. For example, a corrupted database record can be "repaired", a temporary storage queue can be created or deleted, and so on. It provides a useful extension to the facilities provided by the CEMT master terminal transaction.

Invoking the command level interpreter

The command level interpreter is a CICS application program and runs as a CICS transaction. It is started by a 'CECI' or 'CECS' transaction identifier, followed optionally by the command.

The general format is:

```
CECI|CECS [command]
```

where 'command' can be any of the CICS commands (except EXEC DLI) described in this manual.

The use of CECI will give the full facilities of the interpreter right through to execution of the command.

For example, entering:

```
CECI READ FILE('FILEA')
```

will give the screen display shown in Figure 16 on page 68. A severe error message (indicated by S) is displayed near the bottom of the screen.

If you are trying this command using the pregenerated system, as described in the *CICS/MVS Installation Guide*, you must first sign on as one of the operators defined in the sample sign-on table, with RSLKEY=1. See also "Security rules" on page 72.

Modifying the command input to:

```
READ FILE('FILEA') RIDFLD('009000')
```

will give the screen display shown in Figure 17 on page 69. The error message has disappeared because the requested record identification field has been supplied.

The command is now ready to be executed, and this is achieved simply by pressing the ENTER key. The display shown in Figure 18 on page 70 will appear showing the result of execution.

It is possible to prevent unauthorized access by the interpreter to resources such as data sets. See the security rules later in this chapter.

A question mark (?) before the command always gives the command syntax check display and prevents command execution.

The use of CECS forces a question mark before the command. This always gives the command syntax check display and prevents command execution. In a system where security is important, CECS can be made more widely available than CECI.

Screen layout

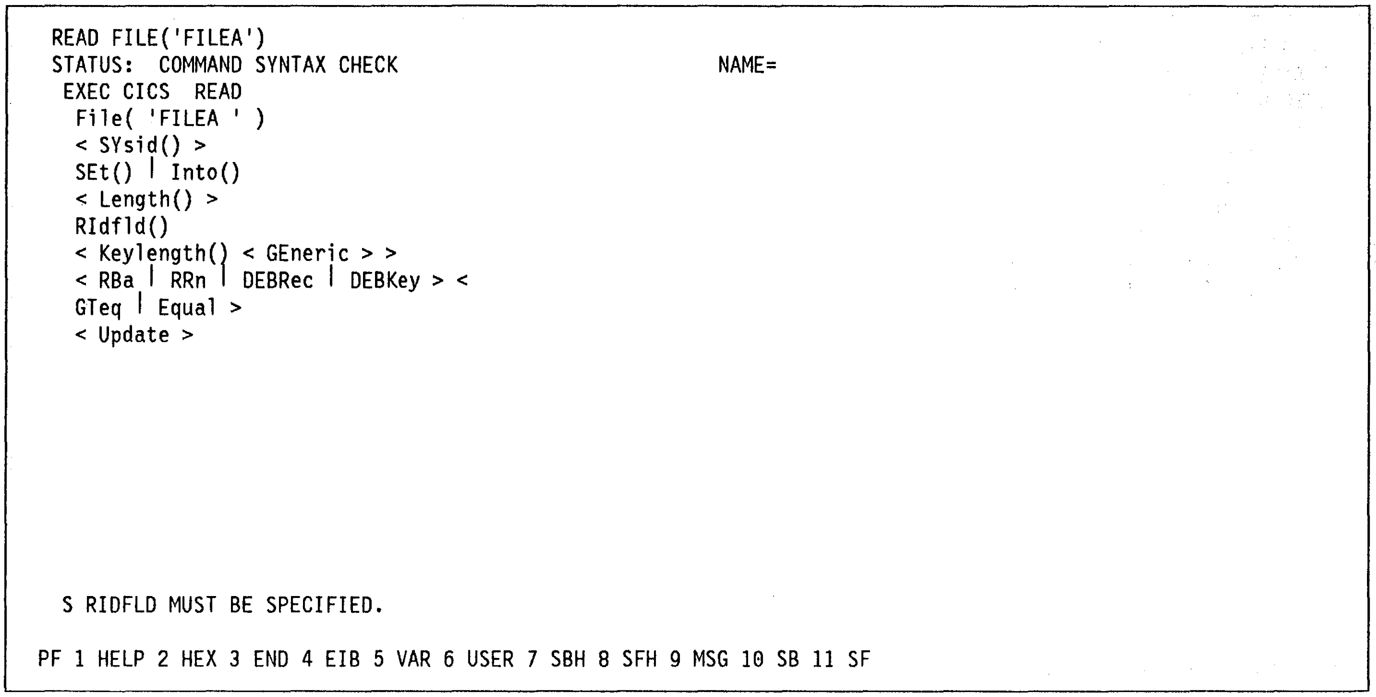
The command interpreter uses a basic screen layout of four areas, as shown in Figure 16 on page 68. These areas are:

- Command input area (the first line of the screen)
- Status area (the second line of the screen)
- Information area (21 lines on a 24 x 80 display)
- PF key values area (the last line of the screen).

Command input area

This is the first line of the screen. The command whose syntax is to be checked or which is to be executed, is entered on this line, either in the normal format described in "Chapter 1.2. Command format and argument values" on page 5 and as shown throughout this manual, or in an abbreviated or condensed form that reduces the number of keystrokes involved. The condensed form of the command is obtained as follows:

- The keywords EXEC CICS are optional.
- The options of a command can be abbreviated to any number of characters sufficient to make them unique. Valid abbreviations are shown in uppercase characters in syntax displays.
- The quotes around character strings are optional, and all strings of characters will be treated as character-string constants unless they are preceded by an ampersand (&) in which case they are treated as variables, as described on page 70.



```
  READ FILE('FILEA')
  STATUS:  COMMAND SYNTAX CHECK                         NAME=
   EXEC CICS  READ
    File( 'FILEA ' )
    < SYsid() >
    SEt() | Into()
    < Length() >
    RIdfld()
    < Keylength() < GEneric > >
    < RBa | RRn | DEBRec | DEBKey > <
    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
```

Figure 16. "Command syntax check" display

- Options of a command that receive a value from CICS when the command is executed are called 'receivers', and need not be specified. The value received from CICS will be included in the syntax display after the command has been executed.

The following example shows the condensed form of a command. The file control command:

```
EXEC CICS READ FILE('FILEA')
RIDFLD('009000') INTO(data-area)
```

can be entered on the command input line, as:

```
READ FIL(FILEA) RID(009000)
```

Here, the INTO option is a receiver (as defined above), and can be omitted.

Status area

This is the second line of the screen. It will contain one of the following:

- COMMAND SYNTAX CHECK
- ABOUT TO EXECUTE COMMAND
- COMMAND EXECUTION COMPLETE (or COMMAND NOT EXECUTED)
- EIB DISPLAY
- VARIABLES
- ERROR MESSAGES
- EXPANDED AREA

This status line describes the type of information in the immediately following information area of the display.

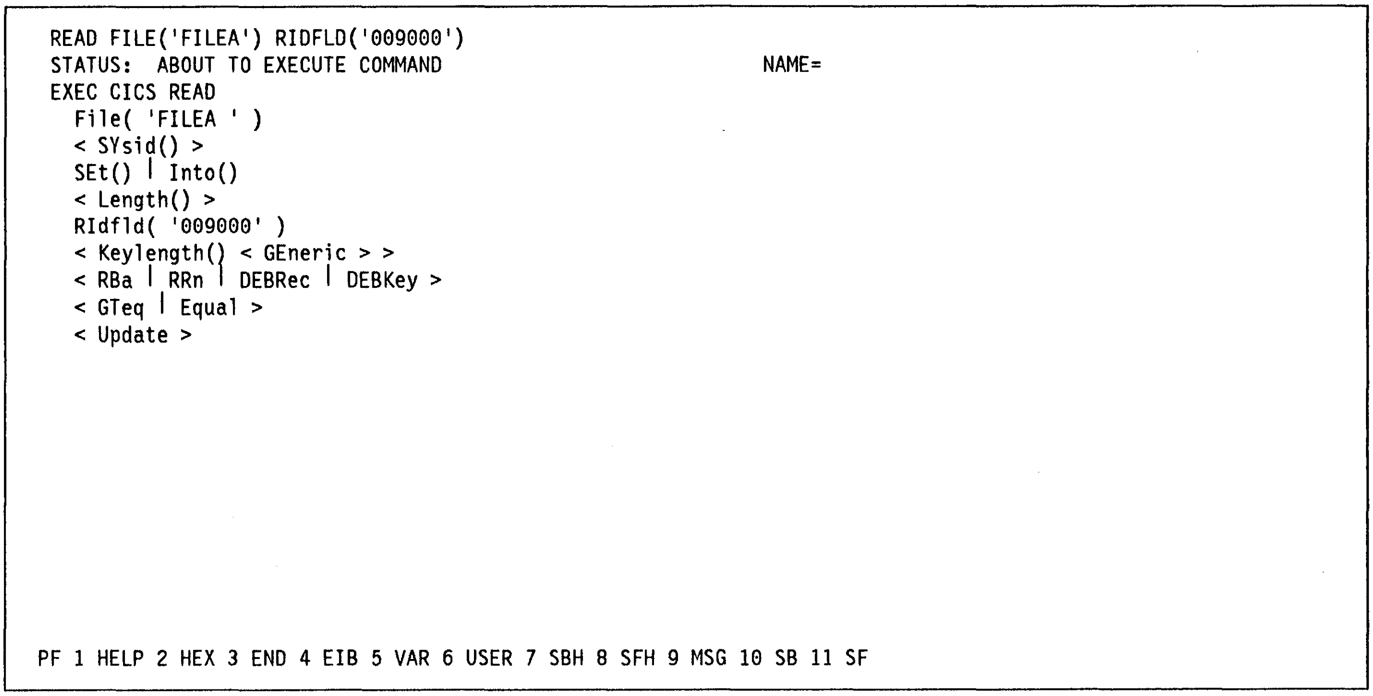
Information area

This area consists of the remainder of the screen between the 'command input' and 'status' areas at the top, and 'PF key values' at the bottom of the screen. This area is used to display the syntax of the entered command, error message information, the response to execution, and any other information that can be obtained by using the PF keys or the cursor.

A line at the bottom of this area is reserved for messages that describe errors in the conversation with the user (for example, 'INVALID PACKED DECIMAL'). These messages are intensified to attract attention.

Command syntax check: When this status message appears (as shown in Figure 16), it indicates that the command that has been entered on the command input line has been syntax checked but is not about to be executed. This will always be the status for CECS or for CECI with a question mark before the command. It is also the status when the syntax check of the command gives severe error messages and for those commands which are not executable (for example, HANDLE CONDITION and HANDLE AID).

The information area of the display for 'command syntax check', 'about to execute command', and 'command execution complete' contains information common to all three displays.



```
READ FILE('FILEA') RIDFLD('009000')
STATUS:  ABOUT TO EXECUTE COMMAND                     NAME=
EXEC CICS READ
  File( 'FILEA ' )
  < SYsid() >
  SEt() | Into()
  < Length() >
  RIdfld( '009000' )
  < Keylength() < GEneric > >
  < RBa | RRn | DEBRec | DEBKey >
  < 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 17. 'About to execute command' display

The full syntax of the command is displayed together with error information at the foot of the display. Options in the syntax panel are intensified to show those specified on the command input line, those assumed by default, and any 'receivers'.

You can modify the command on the command input line at any time by overtyping and pressing ENTER.

When an argument is to be displayed in character format, some of the characters may not be displayable (including lowercase characters). CECI replaces each nondisplayable character by a period. When overtyping a period, you must be aware that the storage may in fact contain a character other than a period. You cannot overtype any character with a period; if you do, the change is ignored and no diagnostic message is issued. Similarly, when a value is displayed in hexadecimal format, overtyping with a blank character is ignored and no diagnostic message is issued.

If you need to overtype a character with a period, you can do so by switching the display to hexadecimal format, using PF2, and overtyping with hex 4B.

If the command has more options than can be held in one display, a plus sign (+) will appear at the left-hand side of the last option of the current display to indicate that there are more. These can be displayed by using one of the scrolling PF keys.

The syntax display differs slightly from the syntax shown throughout the manual in the following ways:

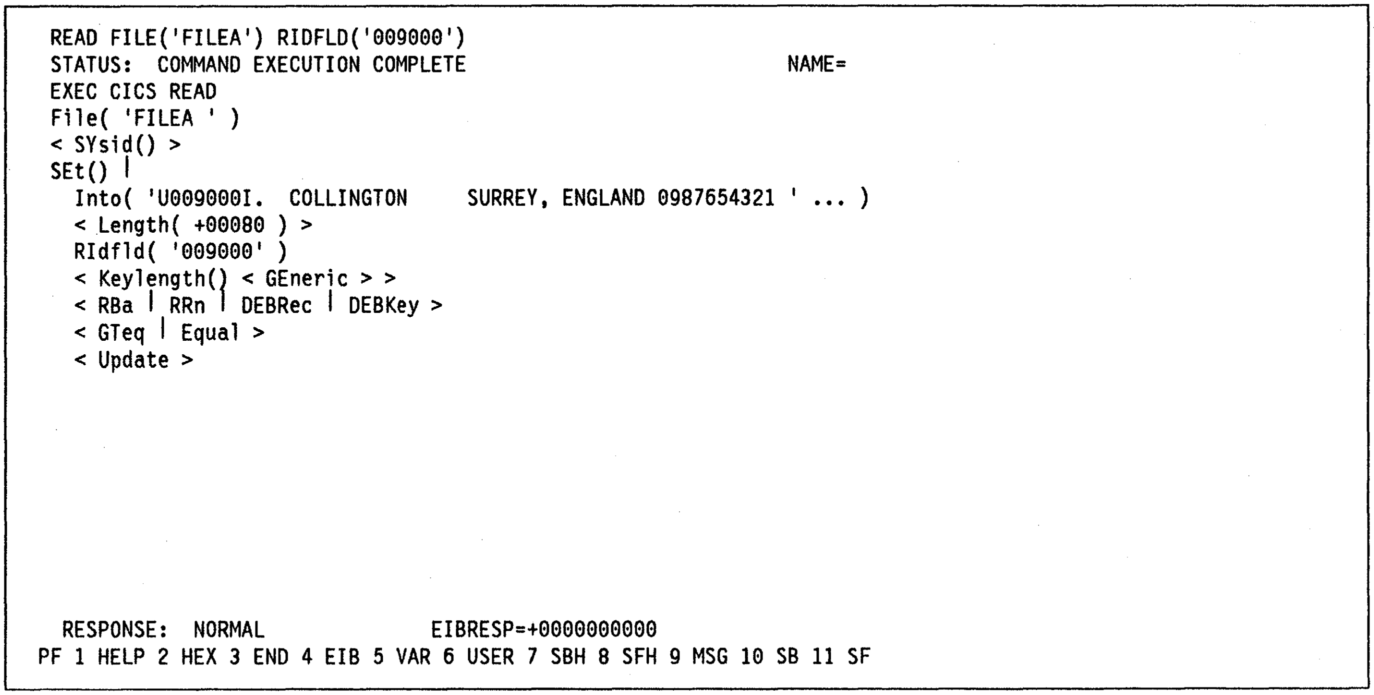
- Square brackets [ ] are replaced by the less-than and greater-than symbols < >.
- Braces { } are not used. If a mandatory option is omitted, an error message will be displayed and execution will not proceed until the option has been specified.
- Parentheses ( ) are used to indicate that an option requires a value or data field but none has been specified.

The error information consists either of a single error message or an indication of the number and severity of the messages generated.

The NAME= field on the syntax display can be used to create a variable containing the current command. (See "Variables" on page 70.)

About to execute command: This display (as shown in Figure 17) appears when none of the reasons for stopping at 'command syntax check' apply. Option values can be modified by overtyping them in the syntax panel.

This is a temporary modification for the duration of the command and does not affect the command input line. It is similar to the modification of option values that is possible with EDF when debugging an application program.



```
   READ FILE('FILEA') RIDFLD('009000')
   STATUS:  COMMAND EXECUTION COMPLETE                         NAME=
   EXEC CICS READ
   File( 'FILEA ' )
   < SYsid() >
   SEt() |
     Into( 'U009000I.  COLLINGTON     SURREY, ENGLAND 0987654321 ' ... )
     < Length( +00080 ) >
     RIdfld( '009000' )
     < Keylength() < GEneric > >
     < RBa | RRn | DEBRec | DEBKey >
     < GTeq | Equal >
     < Update >









    RESPONSE:  NORMAL              EIBRESP=+0000000000
  PF 1 HELP 2 HEX 3 END 4 EIB 5 VAR 6 USER 7 SBH 8 SFH 9 MSG 10 SB 11 SF
```

Figure 18. "Command execution complete" display

Command execution complete: This display (as shown in Figure 18) appears in response to the ENTER key after an 'about to execute command' display. The command has been executed and the results are displayed on the screen. Any 'receivers', whether specified or not, together with their CICS-supplied values, are displayed intensified. If the value of an option is too long for the line, only the first part will be displayed followed by "..." to indicate there is more. Positioning the cursor, using the tab key at the start of the option value, and pressing ENTER will produce an expanded display of the whole option value.

Also displayed at the foot of the information area is the appropriate response code (for example, NORMAL) together with the contents of the EIBRESP field of the EIB.

Note: CECI will return INVREQ for some commands, even if the selected options are correct, because CECI checks every option of the invoked command, some of which may be invalid at invocation. The command executed is the command on the command line together with all receiver parameters.

Because CECI is an interactive transaction, all commands that execute under its control are processed as if the NOHANDLE option were active, thus forcing all responses back to CECI when execution has completed.

Variables: This display will show, in response to pressing key PF5, all the variables associated with the current interpreter session, showing for each, its name, length, and value.

Normally, the value supplied for an option in the command input area is taken as a character string constant. However, there is sometimes a requirement for this value to be represented by a variable. The command interpreter will recognize a value as a variable only if it is preceded by an ampersand (&).

A variable is required when two associated commands are to be connected through the values supplied in their options; for example, READ INTO(data-area) UPDATE and REWRITE FROM(data-area). A variable can be used to make the data area in the FROM option the same as that in the INTO option.

A variable is also useful when the values of 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 will enable a command to be accommodated.

Variables can also be used to contain commands, and variable names can be entered in a command input line that contains complete or partial commands.

Variables are deleted at the end of an interpreter session unless action has been taken to save them, for example, in temporary storage, as described below.

Variables, which can be of data type character, fullword, halfword, or packed decimal, can be created as follows:

1. By naming the variable in a receiver. The variable will be created when the command is executed. The data type is implied by the type of receiver.
2. By adding one or more new entries to the list of variables already defined. This list is displayed by pressing key PF5. The display shows all defined variables giving, for each, its name, length in bytes, and its value. The value is displayed in character form but PF2 can be used to switch from character to hexadecimal. An expanded display of each variable can be obtained by positioning the cursor under the & of the name and pressing ENTER. To create a new character variable, enter its name and its length and press ENTER. The variable will be initialized to blanks, which can then be overtyped. For a fullword, halfword, or packed variable, enter F, H, or P in the length field. These fields are initialized to zero.

   Variable names, lengths, and their values can be modified by overtyping. Variables can be deleted by positioning the cursor under the & of the name and pressing ERASE EOF. Variables can be copied by obtaining the expanded display of the variable and overtyping the name field.
3. By associating a variable name with the value of an option. Positioning the cursor, using the tab key at the start of the line of the syntax display, and pressing ENTER will produce an expanded display of the whole option value. A variable name can now be assigned to the data so displayed.
4. By entering a name in the NAME= field of the syntax panel. This will create a variable containing the current command.

Three variables are provided initially. The first, &DFHC, is a sample. The second, &DFHW, contains a temporary storage WRITEQ command, and the third, &DFHR, contains a READQ command. It is possible to write a command to temporary storage by entering &DFHC in the NAME= field of the syntax panel, entering &DFHW in the command input line, and executing the WRITEQ command. In this way, a list of commands can be written. The command list can be read and executed by alternately entering &DFHR and &DFHC in the command input line.

Expanded area: This display will use the whole of the information area of the screen to display areas selected by means of 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 will then give the expanded area display. The scrolling keys can be used to display all the information if it exceeds a full screen.

PF key values area

The single line at the foot of the screen provides a menu indicating the effect of the ENTER and PF keys for the display. Continuation of interpretation depends entirely upon use of the ENTER key; unless this key is pressed, no further action will occur.

The PF keys are self explanatory; if the terminal has no PF keys, the same effect can be obtained by positioning the cursor under the required item in the menu by means of the tab keys and pressing ENTER. The following PF keys are available:

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: SWITCH HEX/CHAR
switches the display between hexadecimal and character representation. This is a mode switch; all subsequent displays will stay in the chosen mode until the next time this key is pressed.

PF3: END SESSION
ends the current session of the interpreter.

PF4: EIB DISPLAY
shows the contents of the EXEC interface block (EIB); see Appendix A, "EXEC interface block" on page 339 for a description of the fields in the EIB.

PF5: VARIABLES
shows all the variables associated with the current command interpreter session, giving for each its name, length, and value.

PF6: USER DISPLAY
shows what the user would see if the terminal had been executing a transaction that contained the commands that have been executed using the interpreter.

PF7: SCROLL BACK HALF
scrolls half a screenful backward.

PF8: SCROLL FORWARD HALF
scrolls half a screenful forward.

PF9: EXPAND MESSAGES
shows all the messages generated during the syntax check of a command.

PF10: SCROLL BACK
scrolls backward.

PF11: SCROLL FORWARD
scrolls forward.

PF12: UNDEFINED
means that this key is not available with this type of display.

Terminal sharing

When the command being interpreted is one that uses the screen that the interpreter is using, the command interpreter will manage the sharing of the screen between the interpreter display and the user display.

The user display will be restored:

- When the command being executed requires input from the operator.
- When the command being executed is about to modify the user display.
- When USER DISPLAY is requested.

Thus when a SEND command is followed by a RECEIVE command, the display sent by the SEND command appears twice: once when the SEND command is executed, and again when the RECEIVE command is executed. It is not necessary to respond to the SEND command but, if a response is made, the interpreter will remember it and redisplay it when the screen is restored for the 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 a SEND command is executed.

Program control

The interpreter is itself a CICS application program, and the execution of certain program control commands may cause different results from an application program containing those commands. For example, an EXEC CICS ABEND command will be intercepted by the interpreter rather than abending the interpreter (unless the CANCEL option is specified).

If the interpreter is used to link to a program, the interpreter will not be aware of modifications to the user display made by that program. If the interpreter executes an XCTL command, control will be transferred to that program and this will conclude the interpreter session.

Security rules

To invoke the command interpreter, the user must have a security key that matches the security key defined in the PCT.

The command level interpreter transaction identifier (CECI) specifies, by default, that resource level security checking is required for any resources referenced with the interpreter. This checking applies to data sets, transient data queues, temporary storage queues, programs, transaction identifiers of the START command, and journal file identifiers.

If the resource security level specified in the appropriate CICS table (for example, the FCT for a data set) is not matched by the authorization obtained from a sign-on, the resource security check fails, and the response to the command will be the NOTAUTH condition (EIBRESP=70). This response is given on the 'command execution complete' display.

Installing the command level interpreter

To ensure that the command interpreter is available on the system, the system programmer must make one group entry in the PPT and in the PCT. (See the *CICS/MVS Resource Definition (Online)* manual or *CICS/MVS Resource Definition (Macro)* manual for information on constructing a PPT and a PCT.)

Chapter 1.9. Temporary storage browse

You use the browse transaction (CEBR) to browse the contents of CICS temporary storage queues.

You start the CEBR transaction directly by entering the transaction identifier CEBR, and, optionally, a queue name. You end the transaction by pressing PF3.

You can also start the CEBR transaction from EDF. Press PF5 to obtain the **working storage** display, then PF2 to invoke CEBR. (When you press PF3 to terminate CEBR, after invoking it from EDF, CICS reinstates the EDF working storage display.)

CEBR begins by generating the display shown in Figure 19 on page 74. As you can see from the figure, the display shows the contents of a temporary storage queue associated with the invoking terminal. That is, the transaction's initial display refers to a queue called CEBRxxxx (where 'xxxx ' is your terminal identifier).

You use CEBR commands (see below) or the PF keys to process the queue. You can also use CEBR to copy transient data queues to temporary storage, although you cannot read an output extrapartition transient data queue.

The CEBR transaction allows you to browse, copy, and delete data on queues. You can also browse the output from VS COBOL II library routines. Before you enable the transaction, therefore, consider the possible consequences of using it. In particular, ensure that data cannot be browsed by unauthorized personnel. In other words, you should employ **resource level security**.

Using the transaction

If you invoke the browse transaction from a terminal with the identifier L77A, you receive the display shown in Figure 19 on page 74.

When this display appears, continue the transaction by entering one of the CEBR commands into the command line at the top of the screen.

The PF keys help you view the queue. There is a list of these keys at the bottom of each CEBR display. If your terminal does not have PF keys, you can simulate their use by placing the cursor under the key description at the bottom of the display and pressing ENTER.

If you want a full list of the commands that you can type when CEBR is active, initiate the transaction by typing just CEBR, then pressing PF1. This produces a HELP display. You return from this to the main CEBR panel by pressing the ENTER key.

CEBR commands

Here is a list of the CEBR commands:

QUEUE xxxxxxxx
names a queue that you want to become the "current queue ". The value that you specify can be in hexadecimal, for example, QUEUE X'C134'. CEBR responds by displaying the data that is in the named queue.

TERMINAL xxxx
changes the name of the queue. The four characters represented by "xxxx " (the term-id) become the last four characters of the new queue name.

PURGE
erases the contents of the queue being browsed. If the queue is recoverable, terminate the browse before using the PURGE command, otherwise an abend will occur. Do not use PURGE to erase the contents of an internally generated queue, such as a BMS logical message.

TOP
shows the first page of this queue.

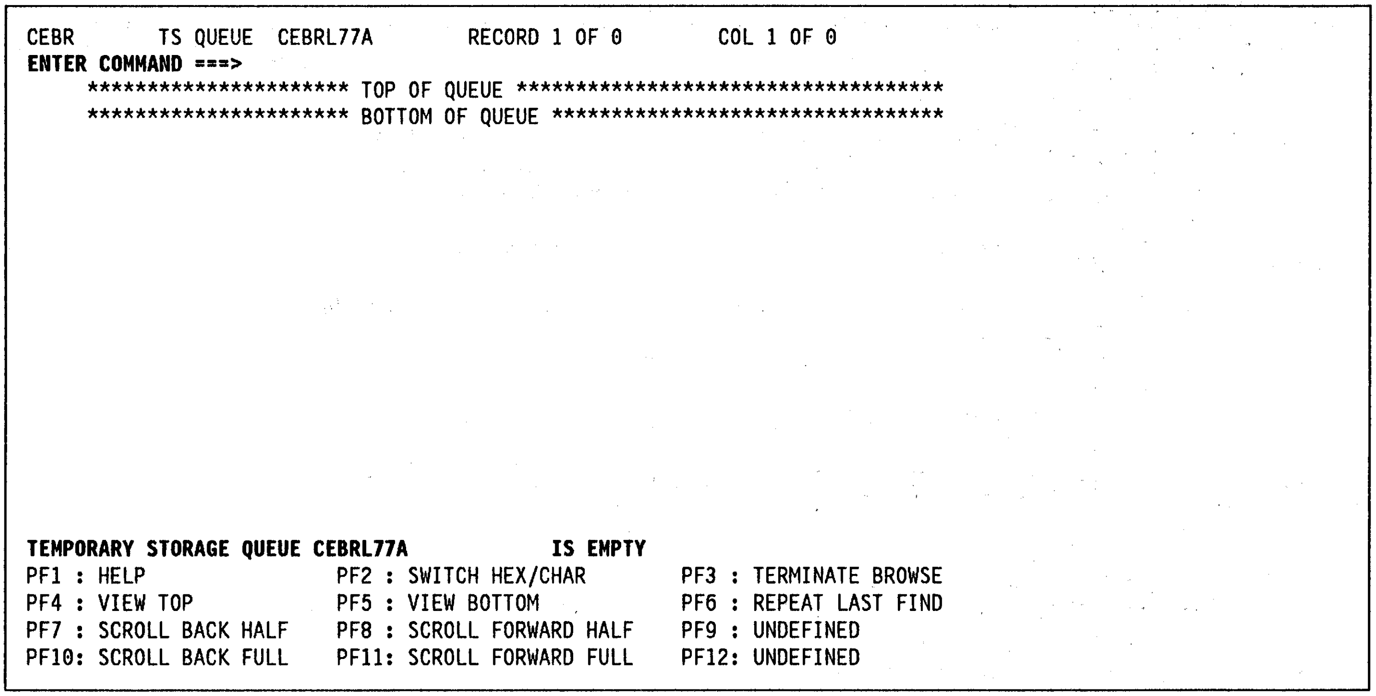
BOTTOM
shows the last page of this queue.

FIND /string
finds the next occurrence of the specified string, making the line containing the string the second on the display page. '/ ' is a delimiting character. It does not have to be '/ ', but must not be a character that appears in the search argument. If there are blank characters in the string, you must terminate it with the delimiting character that started it.

LINE nnnn
makes the specified line the second line on the displayed page.

COLUMN nnnn
moves the displayed area to this column of the queue.

GET xxxx
transfers the named transient data queue to temporary storage. This allows 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.



```
CEBR       TS QUEUE  CEBRL77A        RECORD 1 OF 0        COL 1 OF 0
ENTER COMMAND ===>
     ********************** TOP OF QUEUE ************************************
     ********************** BOTTOM OF QUEUE *********************************















TEMPORARY STORAGE QUEUE CEBRL77A            IS EMPTY
PF1 : HELP                PF2 : SWITCH HEX/CHAR        PF3 : TERMINATE BROWSE
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 19. Initial display produced by the browse transaction

PUT xxxx
transfers the temporary storage queue that is being browsed to the named transient data queue. You can use this command to prepare data for printing. “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.

Resource definition

If you want to use the temporary storage browse transaction in your installation, you must generate a CICS system that includes EDF in the PCT and the PPT. To do this, code the EDF option of the FN operand of both the DFHPCT TYPE=GROUP and DFHPPT TYPE=GROUP system macros.

To limit access to restricted data, specify RSLC=YES in your DFHPCT entry for the transaction. This will at least ensure that users of the transaction can only browse queues with a resource level given by RSL=PUBLIC. Code a DFHTST TYPE=SECURITY system macro for each queue that can be browsed. (See the *CICS/MVS Resource Definition (Online)* manual and *CICS/MVS Resource Definition (Macro)* manual for further information on these topics.)

Part 2. Files and databases

Chapter 2.1. General description of file control facilities 77

Chapter 2.2. File control — VSAM considerations 87

Chapter 2.3. File control — BDAM considerations 89

Chapter 2.4. File control — commands, options, and conditions 93

Chapter 2.5. DL/I services (EXEC DLI command) 101

Chapter 2.6. DL/I services (DL/I CALL statement) 117

Chapter 2.7. DL/I batch programs (shared DB) 129

Introduction to files and databases

CICS transactions can access files and databases, which can be on either a local or remote system.

Files are processed by the CICS file control program, which allows you to read, add, update, delete (VSAM only), and browse records in VSAM and BDAM data sets. When you access files through the file control program, you do not have such considerations as buffer management, blocking and deblocking, and access method dependencies. File control is described in Chapters 2.1 through 2.4.

DL/I databases give you a greater degree of data independence than file control does. You are presented with a logical view of the database in terms of a hierarchy of segments. DL/I offers you facilities for manipulating these segments and you do not need to know how they are organized.

DL/I databases are processed by the IBM Information Management System/Virtual Storage (IMS/VS) licensed program, or by IMS/ESA Version 1.3.

CICS has two programming interfaces to DL/I: the EXEC DLI interface, and the DL/I CALL interface.

The CICS-DL/I interface invoked by means of the EXEC DLI command is described in "Chapter 2.5. DL/I services (EXEC DLI command)" on page 101.

The CICS-DL/I interface invoked by means of the DL/I CALL statement is described in "Chapter 2.6. DL/I services (DL/I CALL statement)" on page 117.

You are recommended to use the 'EXEC DLI interface' because it is simpler to use, can be used with EDF, and can be used in programs executing "above the line" in a 31-bit environment.

The restrictions that an IMS application programmer must observe when running a DL/I batch application program in a shared database environment under CICS are described in "Chapter 2.7. DL/I batch programs (shared DB)" on page 129.

Chapter 2.1. General description of file control facilities

CICS file control provides the application programmer with facilities to read, update, add, delete, and browse data in a data set. To do this, CICS processes files. To CICS, a file is a logical view of a data set and is identified by an 8-character file name. Many files can refer to the same physical data set. CICS file control and CICS application programs are concerned only with files.

The operating system access methods manage the external physical data sets, and CICS file control communicates with these access methods on behalf of application programs.

In general, the application programmer does not need to be concerned with the type of data set nor with the precise physical organization of data in the data set.

A CICS application program reads data from a data set and writes data to a data set by way of a file referencing that data set. The application accesses individual records in the data set. Each such request to access a record is made by means of a CICS command, described in detail in "Chapter 2.4. File control — commands, options, and conditions" on page 93.

To access a record, the application program identifies a file that references the data set as well as the record within the data set. In addition, the application program must specify the area of storage into which the record is to be read or from which it is to be written.

Using CICS file control, you can access data sets that are managed by the following standard operating system access methods:

- Virtual Storage Access Method (VSAM)
- Basic Direct Access Method (BDAM).

The data sets handled by these access methods are called VSAM data sets and BDAM data sets. They are described in the next two sections.

VSAM data sets

CICS supports access to any of the three types of VSAM data set, namely:

- Key-sequenced data set (KSDS)
- Entry-sequenced data set (ESDS)
- Relative record data set (RRDS).

Key-sequenced data set

A **key-sequenced data set (KSDS)** is one in which each of its records is identified by means of a key. The **key** of any record is stored as a field in a predefined position as part of the record. Each key value must be unique in the data set. When the data set is initially loaded with data and when new records are added, the physical order of the records is determined by the collating sequence of the key field. This also determines the order in which records are retrieved when the data set is read in sequence (browsed).

To enable VSAM to determine the physical location of a record in a KSDS, VSAM creates and maintains an index that relates the key of each record with the record's relative location in the data set. When a record is added to or deleted from a KSDS, the index is updated to reflect the change.

Any record in a KSDS may also be identified by its address relative to the beginning of the data set. However, this address, known as the **relative byte address** (RBA), may not remain constant: it may change whenever records are added to or deleted from the data set.

Entry-sequenced data set

An **entry-sequenced data set (ESDS)** is one in which each record is identified by its RBA. Records are stored in an ESDS in the order in which they are initially loaded into the data set. Further records added to an ESDS are always stored after the last record in the data set. Records may not be deleted from an ESDS, nor may their lengths be altered. After a record has been stored in an ESDS, its RBA will remain unchanged. When browsing through an ESDS, records are retrieved in the order in which they were added to the data set.

Relative record data set

A **relative record data set (RRDS)** consists of a series of fixed-length slots that have been predefined to VSAM and in which records may be stored. A record in an RRDS is identified by the **relative record number** (RRN) of the slot in which it is stored. When a new record is added to an RRDS, VSAM assigns the next sequential number in the data set or the number supplied with the request.

Unlike records in a KSDS or an ESDS, records in an RRDS must be of fixed length, equal to the size of a slot in the RRDS.

VSAM data set organization

VSAM data sets are stored on direct-access storage devices (DASD), sometimes called auxiliary storage. The space allocated to a VSAM data set is divided by VSAM into **control areas**, which are further divided into **control intervals**. Each control interval is of fixed predefined size and will, in general, contain a number of **records**. When VSAM reads a record on behalf of CICS file control from a data set, it reads the whole control interval containing the record. The control interval is thus the unit of data transmission between virtual and auxiliary storage.

VSAM allows a KSDS or ESDS to be defined so as to permit records to extend over more than a single control interval. These are called **spanned** records. CICS file control imposes no restrictions on processing spanned records: application programs can access spanned and nonspanned records in exactly the same way.

VSAM paths and bases

It may be convenient for an application program to identify records in a data set in terms of a secondary (alternate) key instead of the primary identification described above. An **alternate key** is defined in the same way as the primary key in a KSDS, as a field of fixed length and fixed position within the record. Any number of alternate keys may be defined and, unlike the primary or base key, an alternate key need not have a unique value.

As an example of primary and alternate keys, consider a KSDS containing records for employees in an organization. Each record is identified by a primary key, defined as the employee number. The employee name, which need not be unique, is used as an alternate key. The employee's department might be defined as a further alternate key.

VSAM allows alternate keys to be defined for key-sequenced and entry-sequenced data sets (though not for relative record data sets). When the data set is created, a secondary or **alternate index** (AIX) is built for each alternate key in the record and is related to the primary or base data set. To access records using the alternate key, a further VSAM object, an **alternate index path** must also be defined. The path then behaves as if it were a KSDS in which records are accessed using the alternate key.

When a record is updated by way of a path, the corresponding AIX is updated to reflect the change. If the record is updated directly by way of the base or by a different path, the AIX will be updated only if, when created, it was defined to VSAM as belonging to what is termed the **upgrade set** of the base data set.

A CICS application program need not be aware of whether the file it is accessing is a path or the base. In a running CICS system, accesses to a single base data set can be made by way of the base and by any of the paths defined over it, as long as each such access route has been defined to CICS in the file control table (FCT).

If the same control interval or the same record in the base data set is simultaneously updated by more than one request, a VSAM exclusive control conflict or a CICS record locking conflict may occur. CICS then causes the request that suffered the conflict to wait until the conflict is resolved before allowing the request to be completed.

VSAM share options

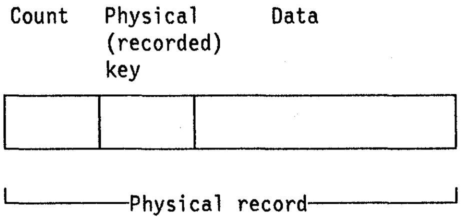
Every data set defined to VSAM is associated with a share options attribute that can take a value of 1, 2, 3, or 4, and which defines how the data set is to be shared among users. In the CICS environment, different FCT entries referring to the same base data set represent different users.

For a data set defined with share options 1 or 2, if the file is defined to be recoverable (LOG=YES in the FCT), CICS will ensure that integrity of data is preserved.

A data set defined with a share option of 3 or 4 can also be updated concurrently through more than one FCT entry. However, CICS is then unable to ensure that integrity of data is preserved.

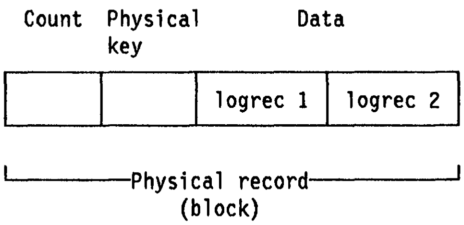
BDAM data sets

CICS supports access to keyed and nonkeyed BDAM data sets. BDAM support makes use of the physical nature of a record on a DASD device. BDAM data sets consist of unblocked records with the following format:



Keyed BDAM files have a physical key identifying the BDAM record. The count area contains the physical key length, the physical data length, and the record's data location. Nonkeyed BDAM files have a zero key length in the count area of the physical record.

CICS may define a further structure on top of BDAM data sets. It introduces the concept of blocked data sets:



The data portion of the physical record may itself consist of logical records. To CICS, the whole structure is defined to be a block: the physical key now identifies the block. CICS will support the retrieval of logical records from the data part of the block. CICS also supports the concept of unblocked records, in which case the structure reverts to the original BDAM concept of one logical record per physical record.

To retrieve a physical record from a BDAM file under CICS, a record identification field (RIDFLD) has to be defined in a CICS file control command to specify how the physical record should be retrieved. This may be done using the physical key, by relative address, or by absolute address.

If the data set is defined to CICS to be blocked, the physical record is seen by CICS as a block. Individual records within the block can be retrieved (deblocked) in two addressing modes: by key or by relative record. To deblock by key, the key of the logical record (that is, the key contained in the logical record) is used to identify which record is required from the block. To deblock by relative record, the record number in the block (relative to zero) of the record to be retrieved, is used. The key or relative record number used for deblocking is specified in a subfield of the RIDFLD option used when accessing CICS BDAM files. The addressing mode for CICS BDAM data sets is set in the FCT using the RELTYPE keyword.

Record identification and BDAM record access are described in more detail in "Chapter 2.3. File control – BDAM considerations" on page 89.

Data set identification

You use the **FILE** option in a file control command to specify the symbolic name of an entry in the FCT which identifies the file that, in turn, refers to the data set to be accessed. The FCT entry must be defined in the CICS system and must have been associated with a physical data set before the command can be executed.

This association is done in any of the following ways:

- The data set is defined by a job control statement for the CICS job. In this case, the FCT entry name is also used as the DD name of the job control statement defining the data set. The data set will normally be allocated to CICS at the time of CICS job initiation and will remain allocated through to CICS termination.
- No job control statement for the data set is included. Instead, the 44-character physical data set name is specified in the FCT by means of the DSNAME option. This, together with the DISP option, enables CICS to allocate the data set dynamically as part of OPEN processing (both for explicit OPEN requests and for implicit requests resulting from the execution of a file access command). The data set will be dynamically deallocated by CICS at the time the file is closed.
- The data set name and the disposition of the data set are specified neither by job control nor in the FCT. Their values are set dynamically, either by means of the EXEC CICS SET command in an application program, or by issuing a CEMT command from the operator's terminal, specifying DSNAME, and one of the disposition options OLD or SHARE. At the time these settings are made, the values are simply recorded in the FCT; they are subsequently used for dynamic allocation that takes place as part of OPEN processing.

Any job control specifications take precedence over assembled or dynamically set values in the FCT. Such settings may be made but if, at the time of the OPEN, the data set is found to have been allocated already by means of job control, any dynamic settings will be overridden.

Accessing data sets from CICS application programs

The following sections describe the facilities available to application programs for accessing data sets by means of files. The description is presented in terms of VSAM data sets, though most of the facilities apply equally to BDAM data sets. Where there are differences, or particular considerations that are appropriate only to BDAM, these are noted in the text and in "Chapter 2.3. File control – BDAM considerations" on page 89.

Retrieving records

Direct reading: A record in the data set is read by means of the **READ FILE** command. The CICS file is associated with an external VSAM or BDAM data set. The command must include sufficient information to identify the record to be retrieved and must also specify whether the record is to be read into an area of storage provided by the application program, or into a CICS buffer acquired by file control. In the latter case, the address of the data in the CICS buffer is returned to the program.

Direct reading from a KSDS: When reading from a VSAM KSDS, the record to be retrieved is usually identified by specifying the full key. It is, however, also possible to specify a partial (**generic**) key. CICS then retrieves from the data set the first record whose leftmost characters match the partial key. Alternatively, it is possible to retrieve the record in the data set whose key is greater than or equal to the full key provided with the command.

Finally, it is also possible to identify the record to be retrieved by providing a generic key together with the 'greater than or equal' (**GTEQ**) option. The NOTFND condition will be returned if no record with the key specified is found or, in the case of the GTEQ option, no record is found with a key greater than or equal to the specified key.

Direct reading from an ESDS: When reading from a VSAM ESDS, an individual record is identified by a relative byte address (RBA). For any record in an ESDS, its RBA cannot change; the application program is therefore able to keep an account of the values of the RBAs corresponding to the records it wishes to access. An access to a VSAM ESDS specifying an incorrect RBA, or an RBA where there is no record, will return the ILLOGIC condition.

Direct reading from an RRDS: When reading from a VSAM RRDS, the record to be retrieved is identified by its relative record number (RRN). Again, the application program needs to be aware of the RRN values of the records it is to retrieve. For records not present in the data set, 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 defined, and if the alternate index path is defined to CICS by an entry in the FCT, a record in the base data set can be retrieved by means of an alternate key. In this case, the generic option and the greater than or equal option may also be used in exactly the same way as for a read from a KSDS using the primary key.

If the alternate key provided in a READ command is not unique, the first record in the data set having that key is read and the DUPKEY condition is returned. To retrieve other records having the same alternate key, a browse operation has to be started at this point.

Sequential reading (browsing): Records may be read in sequence (browsed) from the data set. A browse is initiated using the **STARTBR** command in which a particular record is identified in the same way as for a direct read. However, the STARTBR command identifies only the starting position for the browse; no record is retrieved.

The **READNEXT** command reads records sequentially from the data set, beginning at the starting point provided by the STARTBR command. The field specified in the RIDFLD option on the READNEXT command is updated by CICS with the complete key, relative byte address, or relative record number of the record retrieved each time a READNEXT command is executed.

Also available for use on VSAM data sets is the **READPREV** command. This behaves in the same way as a READNEXT command, except that records are read sequentially **backward** from the starting point provided by the STARTBR command.

Browsing through a KSDS: When browsing through a VSAM KSDS, a generic key may be used on the STARTBR command. However, a browse initiated in this way may only continue forward through a data set. The INVREQ condition is raised if a READPREV is attempted during a browse initiated using a generic key.

The 'key equal to' and 'key greater than or equal to' options may be used on the STARTBR command. The default, unlike the default on a direct read command, is the 'key greater than or equal to' option. If no record can be found matching the key specified with the STARTBR command, the NOTFND condition is returned.

Only after the successful execution of a STARTBR command can a READNEXT or READPREV command be executed successfully.

A forward browse through a VSAM KSDS can be started at the beginning of the data set by specifying a key of hexadecimal zeros, or by specifying options of GENERIC and KEYLENGTH(0) on the STARTBR or RESETBR command. In the latter case, the RIDFLD keyword is required although its value is not used. Similarly, a complete key of hexadecimal 'FF's on a STARTBR command will point to the last record in the data set ready for a backward browse.

Browsing through an ESDS: When browsing through a VSAM ESDS, the GTEQ option is invalid on the STARTBR command. If no record is found matching the RBA specified in the STARTBR command, the ILLOGIC condition is raised. As for a VSAM KSDS, keys of hexadecimal zeros and 'FF's on a STARTBR command enable a forward browse to start at the first record, and a backward browse to start from the last record respectively.

Browsing through an RRDS: When browsing through a VSAM RRDS, the GTEQ option can be used on a STARTBR command and is set by default even though, on a direct READ, use of this option has no effect. A direct-read GTEQ command specifying an RRN that does not exist will return NOTFND because only the EQUAL option is taken. However, a STARTBR GTEQ using the same RRN will complete successfully, and set a pointer to the relevant position in the data set for the start of the browse. The first record in the data set is identified using hexadecimal '1' and the last record by hexadecimal 'FF's.

Browsing by way of a path: Browsing may also be performed by way of an alternate index path to a VSAM KSDS or an ESDS. The browse is performed in exactly the same way as for a VSAM KSDS, but the alternate key is used. The records are thus retrieved in alternate key order.

When nonunique alternate keys are involved, a browse operation will retrieve all records with the same alternate key. The READNEXT command will retrieve records in the order in which they were added to the data set. (READPREV could be used, but the records will be returned in the same order as for READNEXT). When switching from a direct read to a browse, the first record having a nonunique key is retrieved twice: once for the READ command, and again for the first READNEXT command. 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 will be returned upon retrieval of the first two, but not the third. The application program can be designed to revert from browsing to direct reading when the DUPKEY condition no longer occurs.

Ending the browse: An attempt to browse past the last record in a data set will raise the ENDFILE condition. A browse is terminated using the **ENDBR** command. This command should always be issued before an update operation is performed on the same data set (read update, delete with RIDFLD, or write), before a syncpoint, or before task termination. Failure to do so will lead to unpredictable results, including self-inflicted deadlock.

A browse can be reset at any time using the **RESETBR** command. The command can be used to define a new starting position for the browse, or it can be used to change the type of search argument (key, relative byte address, or relative record number) employed.

Simultaneous browse operations: CICS allows a transaction to perform more than one browse on the same data set at the same time. To distinguish between browse operations, the REQID option is included on each browse command.

Skip-sequential processing: For files accessing VSAM data sets, it is possible to browse using skip-sequential processing by altering the key, RBA, or RRN, in the RIDFLD option of the READNEXT or READPREV command to that of the next record required. This may even be done on the first READNEXT or READPREV after a STARTBR or RESETBR command. This procedure allows quick direct access to records in a VSAM data set by reducing index search time.

The RIDFLD option on the READNEXT or READPREV command must be in the same form (key, RBA, RRN) as that used in the STARTBR command or last RESETBR command. If generic keys are used 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 READNEXT command has the same effect as a RESETBR, because the keylength has been changed. To continue browsing from this new point, remove the KEYLENGTH option from subsequent READNEXT commands.

If a 'key equal to' search is specified on a STARTBR or RESETBR command, a READNEXT command using skip-sequential processing may result in the NOTFND condition being raised.

Specifying record length: A file may be defined in the FCT as containing either fixed-length or variable-length records. Fixed-length records may be defined only if the VSAM Access Method Services definition also specifies fixed-size records (average size equals maximum size), and also if all the records in the data set are of that length.

For direct reading and browsing, if the file is defined as containing fixed-length records, and if the application program provides an area into which the record is to be read, that area must be of the defined fixed length. If the file contains variable-length records, the command must also specify the length of the area provided. For fixed-length records and for records retrieved into the CICS-provided buffer, the length argument need not be specified, although it may be useful to do so to check that the record being read is not too long for the available data area. If the length argument is provided, CICS uses the length field to return the actual length of the record retrieved.

Updating records

A record to be updated must first be retrieved using a READ command with the **UPDATE** option. The record is identified in exactly the same way as for a direct read. After the record has been modified by the application program, it is written back to the data set using the **REWRITE** command.

In the case of a VSAM KSDS or ESDS, the record may, as with a direct read, be accessed either by way of an FCT entry that refers to the base, or to a path defined over it.

The REWRITE command cannot identify the record being rewritten. In any one transaction CICS allows only a single update to a given data set to be in progress at any time; the record being rewritten is therefore identified by the previous READ UPDATE command.

A record that has been retrieved as part of a browse operation may not be updated during the browse. The application program must end the browse, read the desired record with a direct command, and perform the update. Failure to do this 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 a CICS-provided buffer. If the record is read into the CICS buffer, it may then be copied into application program storage and rewritten from that storage, or it may be modified and rewritten direct from the CICS buffer.

For a VSAM KSDS, the base key in the record must not be altered when the record is modified. Similarly, if the update is being made by way of a VSAM path, the alternate key used to identify the record must not be altered either, although other alternate keys may be altered. If the file definition allows variable-length records, the length of the record may be changed.

The length of records in an ESDS or an RRDS can never be altered.

Specifying record length: For a file defined as containing fixed-length records, the length of record being rewritten must equal the length defined to VSAM in the Access Method Services definition. For variable-length records, the length must be specified with both the READ and the REWRITE commands.

Deleting records: In the case of a KSDS`or RRDS, instead of rewriting the record, the application program may issue a **DELETE** command to erase it from the data set. As in the case of the REWRITE command, the record to be deleted must not be identified within the DELETE command but is, by default, the most recently read record. If the RIDFLD option is included in this form of the DELETE command, an INVREQ condition is returned to the application program.

The application program may wish to complete the update operation without rewriting or deleting the record. This can be done by means of the **UNLOCK** command. This command releases any CICS storage acquired for the READ and completes the VSAM request by issuing a VSAM ENDREQ command.

Deleting records

As described above, a record in a VSAM KSDS or RRDS may be deleted by first retrieving it for update and then issuing a DELETE command. It is also possible to delete a record in a single operation, again using the **DELETE** command. In this case, the record to be deleted must be identified as part of the command. The record is identified in the same way as when reading a record, except that the GTEQ option may not be used.

Records may never be deleted from an ESDS, irrespective of whether the ESDS is being accessed by way of the base or by a path.

If a full key is provided with the DELETE command, a single record with that key is deleted. This means that, if the data set is being accessed by way of an AIX path that allows nonunique alternate keys, only the first record with that key is deleted. At the completion of such an operation, if further records exist with the same alternate key, the DUPKEY condition is returned.

Deleting groups of records (generic delete): If a generic key is provided with the DELETE command, instead of deleting a single record, all the records in the data set whose keys match the generic key will be deleted with the single command. 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 an AIX path, the records deleted are all those whose alternate keys match the generic key.

Adding records

New records are added to a data set by means of the **WRITE** command, via a file referring to that data set. They must always be written from an area provided by the application program.

Adding to a KSDS: When a record is added to a VSAM KSDS, the base key of the record identifies the position in the data set where the record will 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 WRITE command.

A record added to a KSDS by way of an AIX path is also inserted into the data set in the position determined by the base key. However, the command must also include the AIX key as the record identifier.

Adding to an ESDS: A record added to an ESDS is always added to the end of the data set. It is not possible to insert a record in an ESDS between existing records. After the operation is completed, the relative byte address in the data set where the record was placed is returned to the application program.

When adding a record to an ESDS by way of an AIX path, the record is also placed at the end of the data set. The command must include the AIX key in the same way as for a KSDS path.

Adding to an RRDS: To add a record to an RRDS, the WRITE command must include the relative record number (RRN) as a record identifier. The record is then stored in the data set in the position corresponding to the RRN.

Specifying record length: When writing to a fixed-length VSAM file, the record length must be the same as the value specified at the time the data set was created. In this case, the application program need not include the length with the command, although it may be convenient to do so to check that the length agrees with that originally defined to VSAM. If the file is defined as containing variable-length records, the command must also include the length of the record.

Sequential adding of records (MASSINSERT): A group of records may be added to any VSAM data set using the mass sequential insertion operation. This is more efficient than issuing separate WRITE commands. A mass sequential insertion operation consists of a series of WRITE commands. Each command identifies the record to be added in exactly the same way as for a direct WRITE, but also includes the **MASSINSERT** option. When using the mass insert operation to add records to a KSDS or RRDS, or to a KSDS or ESDS by way of a path, the keys or RRN values of consecutive records must be in ascending, but not necessarily consecutive, order.

The operation must be completed by issuing an UNLOCK command to ensure that all the records are written to the data set and the position is released. A READ command will not necessarily retrieve a record that has been added by an incomplete mass insert operation. If an UNLOCK command is not issued, the mass sequential insertion operation will be completed when a syncpoint is issued, or at task termination.

Review of file control command options

CICS application programs read data from, and write data to, external data sets by way of files referencing those data sets. Whenever you retrieve a record using the READ command, add a record using the WRITE command, delete a record using the DELETE command (except in the case when you have read the record for update first), or initiate a browse using the STARTBR command, you identify the record by means of the RIDFLD option. Further, during a browse using READNEXT or READPREV commands, you must include this option to provide a means for CICS to return the identifier of each record retrieved.

RIDFLD option

RIDFLD identifies a field containing the record identification appropriate to the access method and the type of data set being accessed. The detailed formats are described in Chapters 2.2 and 2.3.

When retrieving records from a VSAM KSDS, or from a VSAM KSDS or ESDS by way of an alternate index path, or when identifying a starting position for a browse in this type of data set, you can include one or both of the further options GTEQ and GENERIC with the command. These options are used when the RIDFLD option by itself may not identify a specific record in the data set.

When executing READNEXT or READPREV commands, the application program would not normally set the RIDFLD field. After each command, CICS updates this field with the actual identifier of the record retrieved. The application may, however, alter the RIDFLD value to identify a new position from which the browse is to continue.

INTO and SET options

When you retrieve a record using the READ, READNEXT, or READPREV commands, the record is retrieved and placed in main storage in accordance with the INTO or SET option that you have specified.

- INTO specifies the area in main storage into which the record is to be placed. For fixed-length records, you need not include the LENGTH option. If you do, the length specified must be the same as the defined length, otherwise the LENGERR condition will occur. For variable-length records you must always specify, in the LENGTH option, the maximum length of record that your application program will accept, otherwise the LENGERR condition will occur. This condition will also occur if the record exceeds this maximum length, in which case the record is truncated to that length. After the record has been retrieved, if the LENGTH option has been included, the data area specified in this option is set to the actual record length (before any truncation occurred).
- SET specifies a pointer reference that is set to the address of the buffer in main storage acquired by CICS and large enough to hold the record. When using SET, the LENGTH option need not be included. If it is included, the data area specified is set to the actual record length after the record has been retrieved.

FROM option

When you add records using the WRITE command, or update records using the REWRITE command, you must specify the record to be written by means of the FROM option. FROM specifies the area in main storage which contains the record to be written. In general, this area will be part of storage owned by the application program. On a REWRITE command, the FROM area will usually, though not necessarily, be the same as the corresponding INTO area specified on the READ UPDATE command. The length of the record may be changed when rewriting to a variable-length VSAM KSDS.

The LENGTH option must always be included when writing to a variable-length file. If the value specified exceeds the maximum allowed in the data set definition, the LENGERR condition will be returned when the command is executed. When writing to a fixed-length file, CICS uses the length specified in the Access Method Services data set definition as the length of the record to be written. The LENGTH option need not therefore be included. If it is, its value is checked against the defined value and LENGERR is returned if the values are not equal.

Preventing transaction deadlocks

The application programmer should be aware of the need to design applications in such a way as to prevent the occurrence of transaction deadlocks. A deadlock may occur when one transaction needs exclusive use of some resource (for example, a particular record in a data set) that is already held by a second transaction. The first transaction will wait for the resource to become available but, if the second transaction is not in a position to release it because it, in turn, is waiting on some other resource held by the first, both are deadlocked and the only way of breaking the deadlock is to cancel one or both transactions.

A transaction may have to wait for a resource for a number of different reasons while executing file control commands. First, for both VSAM and BDAM data sets, any record that is in the process of being modified is held in exclusive control by the access method for the duration of the request. (In the case of VSAM, not only the record but the complete control interval containing the record is held in exclusive control). Secondly, if a transaction has modified a record in a recoverable file, that record is locked by CICS to the transaction even after the request that performed the change has completed. That transaction may continue to access and modify the same record; other transactions, however, are obliged to wait until the transaction releases the lock either by issuing a sync point request or by terminating.

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 encountering a set of circumstances which results in a deadlock situation; for this reason, it is important to recognize the possibility of deadlock at an early stage of the application program design.

The following are examples of different types of deadlock:

- Two transactions are running concurrently and are modifying records within a single recoverable file, through the same FCT entry, in the following manner:

  ```
  Trans.1:  READ UPDATE rec.1 UNLOCK 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 (even though it has completed the READ UPDATE with an UNLOCK). Transaction 2 has similarly acquired the record lock for record 2. Each transaction is then in a deadlock state because it wishes to acquire the lock held by the other transaction.

- Two transactions are running concurrently and 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
  ```

  In this case the record locks have been acquired on different files as well as different records, however the deadlock is similar to the first example.

- Two transactions are running concurrently and are modifying a single recoverable KSDS, through the same FCT entry, with the following sequence of operations:

  ```
  Trans.1:  READ UPDATE rec.1

  Trans.2:  DELETE rec.3

  Trans.1:  WRITE rec.3

  Trans.2:  READ UPDATE rec.2
  ```

  Suppose records 1 and 2 are stored in the same control interval (CI). The first READ UPDATE has acquired VSAM exclusive control of the CI containing record 1. The DELETE operation has completed and has acquired the CICS record lock on record 3. The WRITE operation is forced to wait for the lock on record 3 to be released before it can complete the operation. Finally, the last READ UPDATE is forced to wait for the VSAM exclusive control lock held by transaction 1 to be released.

- A transaction is in progress of browsing through a VSAM data set that uses shared resources (LSRPOOLID not equal to NONE in the FCT). Before completing the request with an ENDBR, the transaction issues a further request to update the current record or another record that happens to be in the same control interval. Because VSAM already holds exclusive control of the control interval on behalf of the first request, the second request is forced to wait indefinitely: the transaction has produced its own deadlock.

To reduce the opportunity for deadlock, CICS recognizes certain situations that may lead to it and prevents them occurring by returning the INVREQ condition to the application program. For example, CICS does not allow a transaction to issue a READ UPDATE request to a particular file if a previous READ UPDATE has not yet been completed with a REWRITE, DELETE or UNLOCK command.

CICS does not, however, detect every situation that may cause a deadlock. The application programmer can avoid deadlocks by following these rules:

1. All applications that update (modify) multiple resources should do so in the same order. For instance, if a transaction is updating more than one record in a data set, it can do so in ascending key order (or ascending alternate key order). A transaction that is accessing more than one file should always do so in the same predefined sequence of files.
2. An application that issues a READ UPDATE command should follow it with a REWRITE, DELETE without RIDFLD, or UNLOCK to release position before performing any other operation on the data set referenced by that file.
3. A sequence of WRITE MASSINSERT requests must use ascending keys (or RRN values). The sequence must terminate with the UNLOCK command to release position. No other operation on the data set should be performed before the UNLOCK command has been issued.
4. An application must end all browses on a data set by means of ENDBR commands (thereby releasing position) before issuing a READ UPDATE, WRITE, or DELETE with RIDFLD to the data set.

KEYLENGTH option for remote data sets

In general, execution of file control commands requires the RIDFLD and KEYLENGTH options to be specified. KEYLENGTH may be specified explicitly in the command, or it may be determined implicitly from the FCT.

For files accessing remote data sets (that is, those for which SYSID has been specified), KEYLENGTH should be specified only if RIDFLD specified a key. If the remote data set is being browsed, KEYLENGTH is not required for the READNEXT or READPREV commands.

For a remote BDAM data set, where the DEBKEY or DEBREC options have been specified, KEYLENGTH (when specified explicitly) should be the total length of the key (that is, all specified subfields).

Chapter 2.2. File control — VSAM considerations

Record identification

You identify records in VSAM data sets by **key**, by **relative byte address (RBA)**, or by **relative record number (RRN)**.

To distinguish which format of record identification is to be used, the RBA and RRN options can be used on most file commands that access VSAM data sets. The options effectively define the format of the record identification field (RIDFLD). If neither the RBA nor the RRN option is specified, the RIDFLD option should contain a key to be used for accessing a VSAM KSDS, or a VSAM KSDS or ESDS by way of a path.

The RBA option specifies that the RIDFLD contains the relative byte address of the record to be accessed. A relative byte address is used to access a VSAM ESDS, and it may also be used to access a VSAM KSDS. However, if a KSDS is accessed in this way, the RBA of the record may change during the transaction as a result of another transaction adding records to, or deleting records from, the same data set.

The RRN option specifies that the RIDFLD contains the relative record number (the first record in a data set being numbered 1) of the record to be retrieved.

Operations involving use of VSAM keys may specify either a complete key or a generic (partial) key. (The one exception to this rule is when a record is written to a VSAM KSDS. In this instance, the complete key **must** be specified in the RIDFLD option of the command.) When a generic key is used, its length must be specified in the KEYLENGTH option, and the GENERIC option must also be specified on the command. A generic key cannot have a keylength equal to the full keylength. That is, a generic key is defined to be of a length that is strictly less than that of the complete key.

For both complete and generic keys, the GTEQ option may also be specified on certain commands. The command then positions at, or applies to, the record in the data set with the next higher key if a matching key cannot be found. When a data set is being accessed by way of an alternate index path, the record identified is the one with the next higher alternative when a matching record cannot be found.

The application programmer should always, even when using generic keys, use an area of storage for the RIDFLD whose length is equal to the length of the complete key. This is because during a browse operation, after retrieving a record, CICS copies into the RIDFLD area the actual identifier of the record retrieved. In some cases, CICS will return to the application program a complete key, even when a generic key was specified on the command. An example of this is a generic browse through a VSAM KSDS where the complete key is returned to the application program on each READNEXT and READPREV command.

CICS locking of VSAM records in recoverable files

In the previous chapter, the prevention of transaction deadlocks was described in terms of the record locks CICS acquires whenever records in a recoverable file are modified. The locks are held on behalf of the transaction performing the change until the transaction issues a sync point request or terminates (at which time a syncpoint is automatically performed). For VSAM recoverable file processing, further considerations have to be borne in mind.

Whenever a VSAM record is modified, CICS file control locks the record by means of a CICS ENQUEUE request using the primary record identifier as the enqueue argument. If a record is modified by way of a path, the enqueue uses the base key or the base RBA as argument. This means that CICS will permit only one transaction at a time to perform its request, the other transactions having to wait until the first has reached a syncpoint.

For the READ UPDATE, REWRITE-related commands, the record lock is acquired as soon as the READ UPDATE has been issued. For a DELETE command that has not been preceded by a READ UPDATE, or for a WRITE command, the record lock is acquired at the time the command is executed. For a WRITE MASSINSERT command, which consists of a series of commands, a separate record lock is acquired at the time each individual WRITE command is performed. Similarly, for a DELETE GENERIC command, each record deleted acquires a separate lock on behalf of the transaction issuing the request.

The record locks described above are known as **update locks** because they are acquired whenever a record is updated (modified). A further type of lock known as a **delete lock** is also acquired by file control whenever a DELETE, a WRITE, or a WRITE MASSINSERT operation is being performed. A DELETE operation therefore acquires two separate locks on the record being deleted.

The delete lock, separate from the update lock, is required because of the method used by file control to implement WRITE operations. In advance of executing a WRITE or WRITE MASSINSERT command to a KSDS or RRDS, file control finds and locks the empty range into which the new record or records are to be inserted. The empty range is locked by identifying the next existing record in the data set and acquiring its delete lock.

The empty range is locked in order to prevent other requests simultaneously adding records into the empty range. Furthermore, the end of the empty range must not be changed while the add operation is in progress. If another transaction issues a request to add records into the empty range or to delete the record at the end of the range, the delete lock will force the transaction to wait until the WRITE or WRITE MASSINSERT is completed. The record held with a delete lock may, however, be updated by another transaction during the WRITE operation.

Unlike an update lock, a delete lock is held only for the duration of a DELETE, WRITE, or WRITE MASSINSERT operation. A MASSINSERT that adds records to the file into more than one empty range will release the previous delete lock as it moves into a new empty range.

Chapter 2.3. File control — BDAM considerations

Record identification

You identify records in BDAM data sets by a **block reference**, a **physical key** (keyed data set), and a **deblocking argument** (blocked data set). The record identification (specified in the RIDFLD option) contains a subfield for each, which, when used, must be in the above order. The subfields are as follows:

Block reference — one of the following:

- Relative block address: 3-byte binary, beginning at relative block number 0 (RELTYPE=BLK).
- Relative track and record (hexadecimal format): 2-byte TT, 1-byte R (RELTYPE=HEX).

  The 2-byte TT begins at relative track 0. The 1-byte R begins at relative record 1.
- Relative track and record (zoned decimal format): 6-byte TTTTTT, 2-byte RR (RELTYPE=DEC).
- Actual (absolute) address: 8-byte MBBCCHHR (RELTYPE operand omitted).

The type of block reference being used must be specified in the RELTYPE operand of the DFHFCT TYPE=FILE system macro that defines the CICS file to be associated with the BDAM data set.

Physical key — required only if the data set has been defined to contain recorded keys. If used, it must immediately follow the block reference. Its length must be the same as the length specified in the BLKKEYL operand of the DFHFCT TYPE=FILE system macro that defines the CICS file to be associated with the BDAM data set.

Deblocking argument — required only if specific records are to be retrieved from a block. If used, it must follow immediately the physical key (if present) or the block reference. If omitted, an entire block will be retrieved.

The deblocking argument may be either a key (specify the DEBKEY option on a READ or STARTBR command), in which case its length must be the same as that specified in the KEYLEN operand of the DFHFCT TYPE=FILE system macro, or it may be a relative record number (specify the DEBREC option on a READ or STARTBR command), in which case it is a 1-byte binary number (first record=0).

The examples in Figure 20 on page 90 assume a physical key of 4 bytes and a deblocking argument of 3 bytes.

Browsing records from BDAM data sets

The record identification field must contain a block reference (for example, TTR or MBBCCHHR) that conforms to the addressing method defined for the data set. Processing begins with the specified block and continues with each subsequent block until the browse is terminated.

If the data set contains blocked records, processing begins at the first record of the first block and continues with each subsequent record, regardless of the contents of the record identification field. That is, CICS uses only the information held in the TTR or MBBCCHHR subfield of the RIDFLD to identify the record. All other information, such as physical key and relative record, or logical key, is ignored. On completion of the READNEXT command, the RIDFLD is updated by CICS with the complete identification of the record retrieved.

For example, assume a browse is to be started with the first record of a blocked, keyed data set, and deblocking by logical key is to be performed.

Before issuing the STARTBR command, the TTR (assuming that is the addressing method) of the first block should be placed in the record identification field. After the first READNEXT command, the record identification field might contain X'0000010504', where 000001 represents the TTR value, 05 represents the block key, and 04 represents the logical record key.

As another example, assume that a blocked, non-keyed data set is being browsed using relative record deblocking and the second record from the second physical block on the third relative track is read by a READNEXT command. Upon return to the application program, the record identification field contains X'0000020201', where 000002 represents the track, 02 represents the block, and 01 represents the number of the record in the block relative to zero.

The DEBREC and DEBKEY options must be specified on the STARTBR command when browsing blocked data sets, for the correct contents to be returned by CICS in the RIDFLD. Specifying DEBREC on the STARTBR command will cause the relative record number to be returned. The DEBKEY option specified on the STARTBR command will cause the logical record key to be returned.

```
Byte |0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
     ------------------------------------------------
     |RELBLK#| N|                                     Search by relative block;
                                                      deblock by relative record

     |RELBLK#| KEY |                                  Search by relative block;
                                                      deblock by key

     |T  T  R|  PH-KEY   |  KEY  |                    Search by relative track
                                                      and record and key;
                                                      deblock by key

     |M  B  B  C  C  H  H  R| N |                     Search by actual address;
                                                      deblock by relative record

     |T  T  T  T  T  T  R  R|  PH-KEY   |  KEY  |     Search by zoned decimal
                                                      relative track and record
                                                      and key; deblock by key

     |T  T  R|  KEY  |                                Search by relative track
                                                      and record; deblock by key
```

Figure 20. Examples of record identification

The omission of DEBREC or DEBKEY when browsing a blocked file has the following effect. The logical record is retrieved from the block, the length parameter is set equal to the logical record length, **but** the RIDFLD is not updated with the full identification of the record. This method should not be used. This should be contrasted with the omission of the DEBREC or DEBKEY option from the READ command when reading from a blocked BDAM data set. In this case, the **whole** block is retrieved, and the length parameter is set equal to the length of the block.

Adding records to BDAM data sets

When adding records to a BDAM data set, the following considerations and restrictions apply:

1. When adding undefined or variable-length records (keyed or non-keyed), the track on which each new record is to be added must be specified. If space is available on the track, the record is written following the last previously written record, and the record number is placed in the 'R' portion of the record identification field of the record. The track specification may be in any of the acceptable formats except relative block. If zoned decimal relative format is used, the record number is returned as a 2-byte zoned decimal number in the seventh and eighth positions of the record identification field.

   The extended search option allows the record to be added to another track if no space is available on the specified track. The location at which the record is added is returned to the application program in the record identification field being used.

   When adding records of undefined length, the length of the record must be specified in the LENGTH option. When an undefined record is retrieved, the application program must determine its length.

2. When adding keyed fixed-length records the data set must first be formatted with dummy records or "slots" into which the records may be added. A dummy record is signified by a key of hexadecimal 'FF's. The first byte of data contains the record number.

3. When adding non-keyed fixed length records, the block reference must be given in the record identification field. The new records are written in the location specified, destroying the previous contents of that location.

4. When adding keyed fixed-length records, track information only is used to search for a dummy key and record which, when found, is replaced by the new key and record. The location of the new record is returned to the application program in the block reference subfield of the record identification field.

For example, for a record whose identification field is as follows:

```
0 3 0  ALPHA
T T R  KEY
```

the search will start at relative track 3. When control is returned to the application program, the record identification field will be as follows:

```
0 4 6  ALPHA
```

showing that the record is now record 6 on relative track 4.

5. When adding variable length blocked records, a 4-byte record description field (RDF) must be included in each record. The first 2 bytes specify the length of the record (including the 4-byte RDF); the other 2 bytes consist of zeros.

BDAM exclusive control

When a blocked record is read for update, CICS maintains exclusive control of the containing block. An attempt to read a second record from the block before the first is updated (by a REWRITE command), or before exclusive control is released (by an UNLOCK command), will cause a deadlock.

Chapter 2.4. File control — commands, options, and conditions

This chapter shows the syntax of each file control command, describes the purpose and format of each command and its options, and gives a list of the exceptional conditions that can arise during execution of a file control command. These commands are concerned with CICS files. A CICS file refers to an external VSAM or BDAM data set. Many CICS files may refer to the same data set.

Read a record (READ)

```
READ
FILE[1](name)
{INTO(data-area)|SET(ptr-ref)}
[LENGTH(data-area)][2]
RIDFLD(data-area)
[KEYLENGTH(data-value)[3][GENERIC][4]]
[SYSID(name)]
[RBA[4]|RRN[4]|DEBKEY[5]|DEBREC[5]]
[GTEQ|EQUAL][4].
[UPDATE]

Conditions: DISABLED, FILENOTFOUND[6],
DUPKEY[4], ILLOGIC[4], INVREQ, IOERR,
ISCINVREQ, LENGERR, NOTAUTH, NOTFND,
NOTOPEN, SYSIDERR

[1] DATASET is also accepted, but
    FILE is the preferred term
[2] Mandatory with SYSID, and with
    INTO when reading variable-length
    records
[3] Mandatory with SYSID unless RBA
    or RRN is coded also, in which
    case it is invalid
[4] VSAM only
[5] Blocked BDAM only
[6] DSIDERR is equivalent
```

You use this command to read a record, via a file, from a direct access data set on a local or remote system.

If you include the UPDATE option, you must identify the record to be updated by the record identification field specified in the RIDFLD option. Immediately upon 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 per data set within a transaction at any given time. Further, to avoid deadlock when accessing a VSAM data set, your next command to the data set should be a REWRITE, DELETE without RIDFLD, or UNLOCK.

The following example shows you how to read a record from a data set, via a file named 'MASTER', into a specified data area:

```
EXEC CICS READ
     INTO(RECORD)
     FILE('MASTER')
     RIDFLD(ACCTNO)
```

The following example shows you how to read a record for update from a VSAM data set using a generic key and specifying a greater-or-equal key search.

```
EXEC CICS READ
     INTO(RECORD)
     LENGTH(RECLEN)
     FILE('MASTVSAM')
     RIDFLD(ACCTNO)
     KEYLENGTH(4)
     GENERIC
     GTEQ
     UPDATE
```

Write a record (WRITE)

```
WRITE
FILE[1](name)
FROM(data-area)
[LENGTH(data-value)][2]
RIDFLD(data-area)
[KEYLENGTH(data-value)][3]
[SYSID(name)]
[RBA|RRN][4]
[MASSINSERT][4]

Conditions: DISABLED, FILENOTFOUND[5],
DUPKEY[4], DUPREC, ILLOGIC[4], INVREQ,
IOERR, ISCINVREQ, LENGERR, NOTFND[6],
NOSPACE, NOTAUTH, NOTOPEN, SYSIDERR

[1] DATASET is also accepted, but
    FILE is the preferred term
[2] Mandatory with SYSID, and with
    FROM when writing variable-length
    records
[3] Mandatory with SYSID unless RBA or
    RRN is coded also, in which case
    it is invalid
[4] VSAM only
[5] DSIDERR is equivalent
[6] means trying to write to a BDAM
    track address that is not
    defined for the dataset
```

You use this command to write a record to a direct access data set on a local or remote system. For example:

```
EXEC CICS WRITE
     FROM(RECORD)
     LENGTH(DATLEN)
     FILE('MASTER')
     RIDFLD(KEYFLD)
```

For a VSAM entry-sequenced data set (ESDS), the record is always added at the end of the data set. VSAM 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.

For a VSAM KSDS, the record is added in the location specified by the associated key; this location may be anywhere in the data set. For VSAM data sets, the key in the record and the key in the RIDFLD identification field must be the same.

Records for ESDS and KSDS data sets can be either fixed length or variable length. Those for a relative record data set must be fixed length. MASSINSERT operations must proceed with ascending keys, and must be terminated by an UNLOCK before any other request to the same data set.

Update a record (REWRITE)

```
REWRITE
FILE[1](name)
FROM(data-area)
[LENGTH(data-value)][2]
[SYSID(name)]

Conditions: DISABLED, FILENOTFOUND[4],
DUPREC, ILLOGIC[3], INVREQ, IOERR,
ISCINVREQ, LENGERR, NOSPACE,
NOTAUTH, NOTOPEN, SYSIDERR

[1] DATASET is also accepted, but
    FILE is the preferred term
[2] Mandatory with SYSID, and with
    FROM when rewriting variable
    length records
[3] VSAM only
[4] DSIDERR is equivalent
```

You use this command to update a record in a direct-access data set on a local or remote system. You must always precede this command with a READ UPDATE to read the record to be updated. For example:

```
EXEC CICS REWRITE
     FROM(RECORD)
     FILE('MASTER')
```

For VSAM data sets, you must not change the key field in the record.

Delete a record (DELETE) — VSAM only

```
DELETE
FILE[1](name)
[RIDFLD(data-area)[2]
  [KEYLENGTH(data-value)[3]
    [GENERIC [NUMREC(data-area)]]]]
[SYSID(name)]
[RBA|RRN]

Conditions: DISABLED, FILENOTFOUND[4],
DUPKEY, ILLOGIC, INVREQ,
IOERR, ISCINVREQ,
NOTAUTH, NOTFND, NOTOPEN,
SYSIDERR

[1] DATASET is also accepted, but
    FILE is the preferred term
[2] Mandatory with GENERIC
[3] Mandatory with SYSID unless RBA
    or RRN is coded also, in which
    case it is invalid
[4] DSIDERR is equivalent
```

You use the DELETE command to delete a record from a KSDS or RRDS data set on a local or remote system. You must identify the record to be deleted in the RIDFLD option.

You can also delete a record that has been retrieved for update (by a READ UPDATE command), instead of rewriting it, by this command. In this case, you must not specify the RIDFLD option.

You can delete groups of records in a similar way, except that you identify the group by the GENERIC option.

The following example shows you how to delete a group of records in a VSAM data set:

```
EXEC CICS DELETE
     FILE('MASTVSAM')
     RIDFLD(ACCTNO)
     KEYLENGTH(4)
     GENERIC
     NUMREC(NUMDEL)
```

Release exclusive control (UNLOCK)

```
UNLOCK
FILE[1](name)
[SYSID(name)]

Conditions: DISABLED, FILENOTFOUND[3],
ILLOGIC[2], IOERR, ISCINVREQ, NOTAUTH,
NOTOPEN, SYSIDERR

[1] DATASET is also accepted, but
    FILE is the preferred term
[2] VSAM only
[3] DSIDERR is equivalent
```

You use this command to release exclusive control position made in response to a READ command with the UPDATE option. You use it if you retrieve a record for update, and then decide that you do not want to update the record after all. However, for a data set for which the system programmer has specified auto logging, the resource remains under the task control enqueue until either a sync point command is executed or the task is terminated. The record can be in a data set on a local or remote system.

You can also use this command to terminate a VSAM WRITE MASSINSERT operation.

Start browse (STARTBR)

```
STARTBR
FILE[1](name)
RIDFLD(data-area)
[KEYLENGTH(data-value)[2][GENERIC][3]]
[REQID(data-value)]
[SYSID(name)]
[RBA[3]|RRN[3]|DEBKEY[4]|DEBREC[4]]
[GTEQ|EQUAL][3]

Conditions: DISABLED, FILENOTFOUND[5],
ILLOGIC[3], INVREQ, IOERR, ISCINVREQ,
NOTAUTH, NOTFND, NOTOPEN, SYSIDERR

[1] DATASET is also accepted, but
    FILE is the preferred term
[2] Mandatory with GENERIC or SYSID
    unless RBA or RRN is coded
    also, in which case
    it is invalid
[3] VSAM only
[4] Blocked BDAM only
[5] DSIDERR is equivalent
```

You use this command to specify the record in a data set, on a local or remote system, at which you want the browse to start. No records will be read until a READNEXT command (or, for VSAM only, a READPREV command) is executed.

Read next record during a browse (READNEXT)

```
READNEXT
FILE[1](name)
{INTO(data-area)|SET(ptr-ref)}
[LENGTH(data-area)][2]
RIDFLD(data-area)
[KEYLENGTH(data-value)][3]
[REQID(data-value)]
[SYSID(name)]
[RBA|RRN][4]

Conditions: DISABLED, FILENOTFOUND[6],
DUPKEY[4] ENDFILE, ILLOGIC[4], INVREQ,
IOERR, ISCINVREQ, LENGERR, NOTAUTH,
NOTFND, NOTOPEN, SYSIDERR[5]

[1] DATASET is also accepted, but
    FILE is the preferred term
[2] Mandatory with SYSID, and with
    INTO when reading variable
    length records
[3] Mandatory with SYSID unless RBA
    or RRN is coded also, in which
    case it is invalid
[4] VSAM only
[5] Not raised following successful
    STARTBR to remote system if
    that system becomes unavailable.
    Abend AZI4 issued instead.
[6] DSIDERR is equivalent
```

You use this command to read records in sequential order from a data set on a local or remote system. You can also use it during VSAM skip sequential processing,

If the NOTFND condition occurs during a browse, you must include a RESETBR command to reset, or an ENDBR command to terminate, the browse.

A READNEXT command following a READPREV command will read the same record as that read by the READPREV command.

Read previous record during a browse (READPREV) — VSAM only

```
READPREV
FILE[1](name)
{INTO(data-area)|SET(ptr-ref)}
[LENGTH(data-area)][2]
RIDFLD(data-area)
[KEYLENGTH(data-value)][3]
[REQID(data-value)]
[SYSID(name)]
[RBA|RRN]

Conditions: DISABLED, FILENOTFOUND[5],
DUPKEY, ENDFILE, ILLOGIC, INVREQ,
IOERR, ISCINVREQ, LENGERR, NOTAUTH,
NOTFND, NOTOPEN, SYSIDERR[4]

[1] DATASET is also accepted, but
    FILE is the preferred term
[2] Mandatory with SYSID, and with
    INTO when reading variable
    length records
[3] Mandatory with SYSID unless RBA
    or RRN is coded also, in which
    case it is invalid
[4] Not raised following successful
    STARTBR to remote system if
    that system becomes unavailable.
    Abend AZI4 issued instead.
[5] DSIDERR is equivalent
```

You use this command only to read records in reverse sequential order from a VSAM data set on a local or remote system.

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 data set, otherwise the NOTFND condition will be raised for the READPREV command.

A READPREV command following a READNEXT command will read the same record as that read by the READNEXT command.

If you want to restart a browse using the RESETBR command, you must supply a complete key. If the key you supply does not exist, the NOTFND condition will be raised.

Reset start of browse (RESETBR)

```
RESETBR
FILE[1](name)
RIDFLD(data-area)
[KEYLENGTH(data-value)[2][GENERIC]]
[REQID(data-value)]
[SYSID(name)]
[GTEQ|EQUAL][3]
[RBA|RRN][3]

Conditions: DISABLED, FILENOTFOUND[4],
ILLOGIC[3], INVREQ, IOERR, ISCINVREQ,
NOTAUTH, NOTFND, NOTOPEN, SYSIDERR

[1] DATASET is also accepted, but
    FILE is the preferred term
[2] Mandatory with SYSID unless RBA
    or RRN coded also, in which case
    it is invalid
[3] VSAM only
[4] DSIDERR is equivalent
```

You can include this command at any time prior to issuing any other browse command. It is similar to an ENDBR — STARTBR sequence (but with less function), and gives the BDAM user the sort of skip sequential capability that is available to VSAM users through use of the READNEXT command.

End browse (ENDBR)

```
ENDBR
FILE[1](name)
[REQID(data-value)]
[SYSID(name)]

Conditions: DISABLED, FILENOTFOUND[3],
ILLOGIC[2], INVREQ, ISCINVREQ,
NOTAUTH, NOTOPEN, SYSIDERR

[1] DATASET is also accepted, but
    FILE is the preferred term
[2] VSAM only
[3] DSIDERR is equivalent
```

You use this command to end a browse on a data set on a local or remote system. An ENDBR is required only after a successful STARTBR.

Always issue an end browse (ENDBR) command before performing any update operations on the same data set (READ UPDATE, DELETE with RIDFLD, or WRITE), and before a sync point.

File control options

FILE(name)
specifies the name of the file to be accessed. The file, in turn, will be associated with an external data set. The name must be alphanumeric, up to 8 characters in length, and must have been defined in the file control table (FCT).

If SYSID is specified, the data set to which this file refers is assumed to be on a remote system irrespective of whether or not the name is defined in the FCT. Otherwise, the FCT entry will be used to determine if the data set is on a local or remote system.

Note: DATASET(name) is also accepted, but FILE is the preferred term.

DEBKEY (blocked BDAM only)
specifies that deblocking is to occur by key. If neither DEBREC nor DEBKEY is specified, deblocking does not occur.

If KEYLENGTH is specified, its value must be the sum of the lengths of all three subfields comprising the key.

DEBREC (blocked BDAM only)
specifies that deblocking is to occur by relative record (relative to zero). If neither DEBREC nor DEBKEY is specified, deblocking does not occur.

If KEYLENGTH is specified, its value must be the sum of the lengths of all three subfields comprising the key.

EQUAL (VSAM only)
specifies that the search will be satisfied only by a record having the same key (complete or generic) as that specified in the RIDFLD option.

FROM(data-area)
specifies the record that is to be written to the data set referred to by this file.

GENERIC (VSAM only)
specifies that the search key is a generic key whose length is specified in the KEYLENGTH option. The search for a record is satisfied when a record is found that has the same starting characters (generic key) as those specified.

GTEQ (VSAM only)
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 will satisfy the search.

INTO(data-area)
specifies the data area into which the record retrieved from the data set is to be written.

KEYLENGTH(data-value)
specifies the length (halfword binary) of the key that has been specified in the RIDFLD option, except when RBA or RRN is specified, in which case it is invalid. This option must be specified if GENERIC is specified, and it can be specified whenever a key is specified. However, if the length specified is different from the length defined for the data set and the operation is not generic, the INVREQ condition occurs.

The INVREQ condition also occurs if a READ, DELETE, or STARTBR command specifies GENERIC, and the KEYLENGTH is not less than that specified in the VSAM definition.

If KEYLENGTH is omitted from a READNEXT or READPREV command used in a generic browse, normal browsing occurs.

If KEYLENGTH is included in a READNEXT or READPREV command used in a generic browse, a new browse is started using the keylength specified and the key in the RIDFLD option. If the data set is remote, and SYSID is specified, specifying KEYLENGTH will not start a new generic browse on a READNEXT or READPREV command.

If KEYLENGTH(0) is used with the object of reading the first record in the data set, the GTEQ option must also be specified, otherwise the NOTFND condition will be raised. GTEQ is the default for STARTBR and RESETBR, but not for READ.

The use of this option with remote data sets is discussed further in "KEYLENGTH option for remote data sets" on page 85.

LENGTH(parameter)
specifies the length (as a halfword binary value) of the record to be retrieved or written by the READ, READNEXT, READPREV, WRITE and REWRITE commands. On completion of a retrieval operation, (READ, READNEXT, READPREV) the LENGTH parameter is set to the length of the retrieved record.

This option must be specified with SYSID. It must also be specified with the INTO and FROM options on file control 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 or written is not too long for the available data area
- When reading or browsing fixed-length records into an area longer than the record being accessed, the LENGERR condition will be raised for assembler language, PL/I, and VS COBOL II applications if the LENGTH option is not specified.

When reading or 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.

For a READ, READNEXT or READPREV command with the INTO option, the LENGTH parameter must be a data area that specifies the largest record the program will accept. 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 is raised. In this case, the LENGTH data-area is set to the length of the record prior to truncation.

For a READ, READNEXT or READPREV command with the SET option, the LENGTH option need not be specified but, if it is, the parameter must be a data area.

For a WRITE or REWRITE command, the parameter must specify a data value that is the actual length of the record that is to be written. When writing fixed-length records, the LENGTH option need not be specified. However, if it is, its value is compared against the record length defined for the data set and the LENGERR condition is raised if the values are not equal.

MASSINSERT (VSAM only)
specifies that the WRITE command is part of a mass-insert operation.

NUMREC(data-area)
specifies a halfword binary data area that is set to the number of records that have been deleted.

RBA (VSAM only)
specifies that the record identification field specified in the RIDFLD option contains a relative byte address.

REQID(data-value)
specifies as a halfword binary value a unique request identifier for a browse, used to control multiple browse operations on a data set. 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 relative record number (for VSAM data sets), or a block reference, physical key, and deblocking argument (for BDAM data sets). For a relative byte address or a relative record number, the format of this field must be fullword binary. When adding records to a keyed data set, the field must contain the complete key.

If you are executing the READ command under CEDF without specifying the KEYLENGTH option, and if the data set is initially closed, RIDFLD will be displayed with a default value of 4 bytes on the 'About to execute' panel. On the 'Execution complete' panel, however, RIDFLD will be displayed with the length of the key defined for the data set.

Because no keylength was specified with the command, the key length from the FCT is used and, because the data set remains closed until the execution of the command, the FCT will not have been updated to reflect the true key length. The default value of 4 bytes will therefore be displayed on the 'About to execute' panel.

RRN (VSAM only)
specifies that the record identification field specified in the RIDFLD option contains a relative record number. This option should only be used with files referencing relative record data sets.

SET(ptr-ref)
specifies the pointer reference which is to be set to the address of the retrieved record.

In assembler language, if the DUPKEY exceptional condition occurs, the register specified will not have been set, but can be loaded from DFHEITP1.

The pointer reference is valid until the next READ command for the same file or until completion of a corresponding REWRITE or DELETE command in the case of READ UPDATE SET. If the user wishes to retain the data within the field addressed by the pointer, it should be moved to the user's own area.

SYSID(name)
specifies the name of the system whose resources are to be used for intercommunication facilities. The name may be up to four characters in length.

When this option is specified, LENGTH and KEYLENGTH must be specified in some situations where normally they need not be, as follows.

If neither RBA nor RRN is specified, KEYLENGTH must be specified; it cannot be found in the FCT.

If SET is not specified, LENGTH must either be specified explicitly or must be capable of being defaulted from the INTO or FROM option using the length attribute reference in assembler language, or STG and CSTG in PL/I. LENGTH must be specified explicitly in COBOL.

UPDATE
specifies that the record is to be obtained for updating or (for VSAM only) deletion. If this option is omitted, a read only operation is assumed.

File control exceptional conditions

DISABLED
occurs if 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 command or by the CEMT transaction.

DSIDERR
equivalent to FILENOTFOUND.

Default action: terminate the task abnormally.

DUPKEY (VSAM only)

occurs if a record is retrieved by way of an alternate index with the NONUNIQUEKEY attribute, and another alternate index 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.

In assembler language, if the SET option is being used, the register specified will not have been set, but can be loaded from DFHEITP1.

Default action: terminate the task abnormally.

DUPREC

occurs if an attempt is made to add a record to a data set, via a file, or to an alternate index with the UNIQUEKEY attribute, in which the same key already exists.

It may also occur when an attempt is made to add a record to a data set whose upgrade set has an alternate index with the UNIQUEKEY attribute, if the corresponding alternate key already exists in the alternate index.

Default action: terminate the task abnormally.

ENDFILE

occurs if an end-of-file condition is detected during a browse.

Default action: terminate the task abnormally.

FILENOTFOUND

occurs if a file name referred to in the FILE option cannot be found in the FCT.

Default action: terminate the task abnormally.

ILLOGIC (VSAM only)

occurs if a VSAM error occurs that does not fall within one of the other CICS response categories. Further information is available in the EXEC interface block (see Appendix A, "EXEC interface block" on page 339 for details).

Default action: terminate the task abnormally.

INVREQ

occurs if any of the following situations exist:

- A requested file control operation is not allowed according to the file entry specification in the FCT.
- A REWRITE command, or a DELETE command without the RIDFLD option, is issued for a file for which no previous READ UPDATE command has been issued.
- A READNEXT, READPREV, ENDBR, or RESETBR command is issued for a file for which no previous STARTBR command has been issued, or for which a STARTBR was not successful.
- A READPREV command is issued for a file for which the previous STARTBR command has the GENERIC option.
- The KEYLENGTH option is specified (but the GENERIC option is not specified), and the specified length does not equal the length defined for the data set to which this file refers.
- The KEYLENGTH and GENERIC options are specified, and the length specified in the KEYLENGTH option is either less than zero, or greater than or equal to the length of a full key.
- A DELETE command is issued for a file referring to a BDAM data set, or to a VSAM ESDS data set.
- A DELETE command with the RIDFLD option specified is issued for a file referencing a VSAM data set when a READ UPDATE command is outstanding.
- Following a READ UPDATE command for a file, a WRITE or READ UPDATE command is issued for a file referencing the same data set before exclusive control is released by a REWRITE, UNLOCK, or DELETE command.
- An attempt is made to start a browse with a REQID already in use for another browse.
- The type of record identification (for example, key or relative byte address) used to access a data set during a browse is changed by a READNEXT or READPREV command.
- For a WRITE command, when writing records containing imbedded keys, the key in the record area (FROM option) and the key in RIDFLD do not match.

Default action: terminate the task abnormally.

IOERR

occurs if there is an I/O error during a file control operation. An I/O error is any unusual event that is not covered by a CICS exceptional condition.

Further information is available in the EXEC interface block (see Appendix A, "EXEC interface block" on page 339 for details).

Default action: terminate the task abnormally.

ISCINVREQ

occurs when the remote system indicates a failure which does not correspond to a known condition.

Default action: terminate the task abnormally.

LENGERR

occurs if any of the following situations exist:

- The LENGTH option is not specified for a read (without the SET option specified) or write operation involving variable-length records.
- The length specified for a write operation exceeds the maximum record size; the record is truncated.
- The length of a record read during a read operation (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.

- An incorrect length is specified for a read or write operation involving fixed length records.

Default action: terminate the task abnormally.

NOSPACE
occurs if no space is available on the direct access device for adding records to a data set.

Default action: terminate the task abnormally.

NOTAUTH
occurs when a resource security check has failed. Use of SYSID will always raise the NOTAUTH condition when resource security level checking is in effect (RSLC=YES in the PCT). The reasons for the failure are the same as for abend code AEY7, as described in the *CICS/MVS Messages and Codes* manual.

Default action: terminate the task abnormally.

NOTFND
occurs if an attempt to retrieve or delete a record based on the search argument provided is unsuccessful. It may occur on a READPREV command immediately following a STARTBR command which specifies the key of a record that does not exist on the data set.

Default action: terminate the task abnormally.

NOTOPEN
occurs if one of the following situations exist:

- 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. This state can also be specified as the initial state by means of the FILSTAT parameter of the DFHFCT TYPE=FILE system macro.
- The requested file is still open and in use by other requests, but a close request against the file has been received. Existing users are allowed to complete.

This condition can occur only during the execution of the following commands:

- READ
- WRITE
- The first command in a WRITE MASSINSERT sequence
- DELETE
- The first command in a DELETE GENERIC sequence
- STARTBR

Other commands cannot raise this condition because they are part of an active request.

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 is raised.

This condition may also occur when a file control command refers to a file defined as REMOTE, where the remote system is a release of CICS earlier than 1.7. The condition can then occur in response to any file control command.

Default action: terminate the task abnormally.

SYSIDERR
occurs when the SYSID option specifies either a name which is not defined in the intersystem table or a system to which the link is closed.

Default action: terminate the task abnormally.

Chapter 2.5. DL/I services (EXEC DLI command)

DL/I is a general-purpose database control system that executes in a virtual-storage environment under MVS. DL/I simplifies the creation of databases by CICS application programs. It also simplifies the subsequent maintenance of those databases by CICS application programs.

Execution of an application program containing EXEC DL/I commands requires the installation of the IMS/VS licensed program (program number 5740-XX2) Version 1.3.

This chapter outlines the EXEC DLI commands that can be used in command level application programs that are used to access DL/I databases. These application programs can be written in assembler language, COBOL, or PL/I.

These commands have a syntax and format that are similar to CICS commands (they use EXEC DLI instead of EXEC CICS). Full details of the commands are given in the publication *IMS/VS Application Programming for CICS/OS/VS Users*.

For online application programs, the commands are translated by the appropriate command language translator (see "Chapter 1.3. Command language translator" on page 9) into calls to the CICS link edit stub. At execution, DFHEIP is invoked which in turn invokes a DL/I interface program to perform the requested operations.

For batch and shared database application programs, the DL/I link edit stub is invoked.

Application programs may reside above the 16-megabyte line but, for DL/I calls or EXEC DLI commands, the save area, parameter list, and all the parameters must be below the 16-megabyte line.

EXEC DLI command

The EXEC DLI command is similar to the EXEC CICS command, yet provides the same facilities as the existing call DL/I interface, as described in "Chapter 2.6. DL/I services (DL/I CALL statement)" on page 117. You can use a simpler, more flexible, and easier-to-read command format to request the same DL/I facilities. Perhaps as important, you can use EDF to test EXEC DLI commands in an application program; you cannot do that if you use DL/I calls. EDF is described in "Chapter 1.7. Execution (command level) diagnostic facility" on page 57.

CICS trace facilities record events for EXEC DLI requests in the same way as for DL/I call statements.

There are no exceptional conditions for DL/I commands, though you can code HANDLE ABEND commands to handle abends issued by DL/I, including those caused by error status codes.

This chapter describes the EXEC DLI commands, options, and arguments that you can code. Typically, the procedure for accessing a DL/I database is as follows:

1. Schedule your access to the database. That is, tell the system that you want access, and define the kind of access.
2. Perform operations involving the database. That is, read or update data, or request statistics.
3. Terminate access.
4. Request a checkpoint or other system services.

The range of EXEC DLI commands available differs according to the environment in which your program is to run. The following table shows which commands you can use in a given environment.

Command	**Online**	**Shared DB**	**Batch**
GET NEXT	YES	YES	YES
GET NEXT IN PARENT	YES	YES	YES
GET UNIQUE	YES	YES	YES
INSERT	YES	YES	YES
REPLACE	YES	YES	YES
DELETE	YES	YES	YES
STATISTICS	YES	NO	YES
LOAD	NO	NO	YES
CHECKPOINT	NO[1]	YES	YES
SCHEDULE	YES	NO	NO
TERMINATE	YES	NO	NO
ROLL	NO	NO	YES
ROLLBACK	NO	NO	YES
LOG	NO	NO	YES
RESTART	NO	NO	YES
SYMCHKP	NO	NO	YES

[1] use the EXEC DLI TERMINATE command.

General format of EXEC DLI command

The general format of the EXEC DLI command is as follows:

```
EXEC DLI function
[option[(argument)]]...
```

The following panels show the syntax of the EXEC DLI commands. For simplicity, the syntax panels omit the keywords EXEC DLI that should appear at the beginning of each command in your program.

Each function keyword has two forms: a full form, and an abbreviated form. Again for simplicity, the abbreviated forms are shown in the syntax panels, the full form being shown immediately following the panel.

This manual does not explain the meanings of the options of EXEC DLI commands. For such meanings, see *IMS/VS Application Programming for CICS/OS/VS Users.*

Schedule the PSB

```
SCHD
{PSB(name)|PSB((data-area))}
```

The full form of the command is SCHEDULE.

Get one or more segments

```
{GU|GN|GNP}
[USING PCB(integer-expr)]
[KEYFEEDBACK(data-area)]
  [FEEDBACKLEN(integer-expr)]

For each parent segment:
[VARIABLE]
[FIRST|LAST|CURRENT]
[SEGMENT(name)|SEGMENT((data-area))]
[SEGLENGTH(integer-expr)]
[OFFSET(integer-expr)]
[LOCKED]
[SETPARENT]
[INTO(data-area)]
[WHERE(where clause)
  [FIELDLENGTH(integer-expr)]]
[KEYS(data-area)
  KEYLENGTH(integer-expr)]

For the object segment:
[VARIABLE]
[FIRST|LAST]
[SEGMENT(name)|SEGMENT((data-area))]
[SEGLENGTH(integer-expr)]
[OFFSET(integer-expr)]
[LOCKED]
INTO(data-area)
[WHERE(where clause)
  [FIELDLENGTH(integer-expr)]]
[KEYS(data-area)
  KEYLENGTH(integer-expr)]
```

The full form of the command is GET UNIQUE, GET NEXT, or GET NEXT IN PARENT.

Insert one or more segments

```
ISRT
[USING PCB(integer-expr)]

For each parent segment:
[VARIABLE]
[FIRST|LAST|CURRENT]
[SEGMENT(name)|SEGMENT((data-area))]
[SEGLENGTH(integer-expr)]
[FROM(data-area)]
[WHERE(where clause)
  [FIELDLENGTH(integer-expr)]]
[KEYS(data-area)
  KEYLENGTH(integer-expr)]

For the object segment:
[VARIABLE]
[FIRST|LAST]
[SEGMENT(name)|SEGMENT((data-area))]
[SEGLENGTH(integer-expr)]
[OFFSET(integer-expr)]
FROM(data-area)
```

The full form of the command is INSERT.

Load a segment (batch only)

```
LOAD
[USING PCB(integer-expr)]
[VARIABLE]
[SEGMENT(name)|SEGMENT((data-area))]
FROM(data-area)
[SEGLENGTH(integer-expr)]
```

Replace one or more segments

```
REPL
[USING PCB(integer-expr)]

For each parent segment:

[VARIABLE]
[SEGMENT(name)|SEGMENT((data-area))]
[SEGLENGTH(integer-expr)]
[OFFSET(integer-expr)]
[FROM(data-area)]

For the object segment:

[VARIABLE]
[SEGMENT(name)|SEGMENT((data-area))]
[SEGLENGTH(integer-expr)]
[OFFSET(integer-expr)]
FROM(data-area)
```

The full form of the command is REPLACE.

Delete a segment

```
DLET
[USING PCB(integer-expr)]
[VARIABLE]
[SEGMENT(name)|SEGMENT((data-area))]
FROM(data-area)
[SEGLENGTH(integer-expr)]
```

The full form of the command is DELETE.

Return pool statistics

```
STAT
[USING PCB(integer-expr)]
INTO(data-area)
[LENGTH(integer-expr)]
[VSAM|NONVSAM]
[FORMATTED|UNFORMATTED|SUMMARY]
```

The full form of the command is STATISTICS.

Terminate access to the PSB

```
TERM
```

The full form of the command is TERMINATE.

When you issue a TERM command, the resources associated with the previously scheduled PSB, are released, and become available for scheduling by other tasks. (The issuing of a sync point is part of this operation.)

Request a basic checkpoint

```
CHKP
ID(data-area)|('char-expr')
```

The full form of the command is CHECKPOINT.

Back out updates and abend (batch only)

```
ROLL
```

Back out updates and return control (batch only)

```
ROLB
```

The full form of the command is ROLLBACK.

Write record to system log (batch only)

```
LOG
FROM(data-area)
[LENGTH(integer-expr)]
```

Request a symbolic checkpoint (batch only)

```
SYMCHKP
ID(data-area)|('char-expr')
[AREA1(data-area)
  [LENGTH1(integer-expr)]]
      ...
[AREA7(data-area)
  [LENGTH7(integer-expr)]]
```

The full form of the command is SYMBOLIC CHECKPOINT.

Invoke extended restart (batch only)

```
XRST
[ID(data-area)|('char-expr')]
[MAXLENGTH(integer-expr)]
[AREA1(data-area)
  [LENGTH1(integer-expr)]]
      ...
[AREA7(data-area)
  [LENGTH7(integer-expr)]]
RETRIEVE
USING PCB(integer-expr)
KEYFEEDBACK(data-area)
[FEEDBACKLEN(integer-expr)]
```

The full form of the command is RESTART.

General rules and conventions

As a general rule, you need only specify LENGTH parameters in COBOL application programs.

On the GET, INSERT, and REPLACE commands, you can repeat the segment-oriented keywords (that is, all those except USING PCB and KEYFEEDBACK) for each segment, but you must name them in hierarchical order, that is, the last segment named must be the object segment.

You must code keywords preceding the keyword SEGMENT immediately preceding the segment to which they apply, but within themselves they may be coded in any order.

Similarly, you must code keywords that follow the keyword SEGMENT immediately following the segment to which they apply, but within themselves they may be coded in any order.

You cannot code either FIRST or CURRENT on GET UNIQUE commands, but you must code the SEGMENT option for the object segment on GET UNIQUE commands.

You can specify the name of a segment, or a PSB, either as a literal or as a variable. If you specify it as a variable, you must code the name of the data area containing its current value in double parentheses, as shown in the syntax displays.

You cannot code the KEYS option and the WHERE option for the same segment. You can specify KEYS only once per command, and you must specify it in the highest level segment in the command.

A **where clause** has the following form:

```
WHERE(fieldname operator value
  [AND|OR fieldname operator value...])
[FIELDLENGTH(integer-expr1
  [,integer-expr2,...])]
```

'fieldname' must be the name of a field as defined in the database description (the DBD).

'value' is a reference to a data area containing the value to be compared. Alternatively, in an assembler or PL/I program, 'value' can be a literal character string in single quotes, for example:

```
WHERE (STATUS='SINGLE')
```

You can specify values for the WHERE operand as Boolean expressions. Such expressions help to identify the segment to be accessed. For example, you could use the following expression to identify, from a personnel database, all subjects with certain predefined attributes. You can specify up to 12 Boolean qualifiers in a single statement.

```
WHERE(AGE > SEVENTEEN AND
      AGE < TWENTYONE AND
      HAIR = BLONDE OR
      IQ > ONESIXZERO)
```

You must specify delimiters for COBOL and PL/I EXEC DLI commands, in the same way as EXEC CICS commands, by END-EXEC for COBOL and by a semicolon for PL/I, for example:

```
For COBOL:

EXEC DLI GU SEGMENT(SKILL)
         INTO(SKILLSTRUCT)
         WHERE(SKILLTYPE=PLUMBER)
         END-EXEC

For PL/I:

EXEC DLI GU SEGMENT(SKILL)
         INTO(SKILLSTRUCT)
         WHERE(SKILLTYPE=PLUMBER);
```

When coding EXEC DLI commands with a fieldname, you can use special characters (# or $) if single quotes are put around the name in parentheses.

When using EXEC DLI commands in assembler language application programs, you must obey the conventions for EXEC CICS commands (see "Coding conventions" on page 5) as well as the following rules that apply to expressions in the WHERE clause:

1. The WHERE clause must not contain macro variables (identified by the prefix '&') because '&' is treated as a Boolean operator by the CICS translator.
2. The operator following a comparison operator in a WHERE clause is restricted to one of the following:
   - A literal character string in single quotes
   - An assembler language relocatable expression not containing operators, for example:

WHERE(PTNO¬='0000')	valid
WHERE(PTNO<LAB3)	valid
WHERE(PTNO<LAB3(2,R1))	valid
WHERE(PTNO<LAB3+2)	not valid

DL/I interface block (DIB)

Whenever you make an EXEC DLI request, DLI responds by storing information in the DL/I interface block (the DIB) in your program. A DIB is inserted automatically into your program by the CICS command translator. The DIB contains the following named fields:

Fields of the DIB

Field	ASM	COBOL	PL/I
DIBSTAT	CL2	PIC XX	CHAR(2)
DIBSEGM	CL8	PIC X(8)	CHAR(8)
DIBSEGLV	CL2	PIC XX	CHAR(2)
DIBKFBL	H	PIC S9(4) COMP	FIXED BIN(15,0)

DIBSTAT
is the DL/I status code. It indicates the degree to which your DL/I request has been successful. The status code returned can be one of the following:

bb (blanks) request successful

GA crossed hierarchical boundary into higher level

GB end of data set; beyond last segment

GD segment position lost or insert had missing segment specification

GE segment not found

GG position set at start of database after GET with processing option GO

GK different segment type at same level returned

II segment to insert already exists

LB segment to load already exists (batch only)

NI secondary index to insert already exists (batch programs only)

TG TERM attempted when PSB not scheduled.

A full list of status codes is given in the application programming reference manual for IMS/VS.

Any other status code indicates that the DL/I interface program has found an unrecoverable error. Such an error abends your CICS transaction. The abend code generated by the error has the form DHxx, where xx is the DL/I status code.

DIBSEGM
is the name of the object segment or the lowest level parent segment actually retrieved.

DIBSEGLV
gives the hierarchical level of the object segment or lowest level parent segment actually retrieved.

DIBKFBL
is the halfword binary value, when KEYFEEDBACK has been specified, representing the actual length of the key returned to the KEYFEEDBACK area. It normally represents the concatenated key of the segment named in DIBSEGM, but may be truncated if the area you provide is not long enough.

Example of DL/I requests using EXEC DLI

The following example shows, in assembler language, COBOL, and PL/I, the use of the EXEC DLI command in a CICS application program to request DL/I services.

ASM example of EXEC DLI commands

```
*ASM XOPTS(CICS,DLI)
*
*  PROGRAM: SAMPLE
*           ASSEMBLER LANGUAGE EXEC DLI ONLINE PROGRAM
*
*  RELEASE DEPENDENCIES: CICS/MVS 2.1, IMS/VS 1.3
*
R2       EQU   2
R3       EQU   3
R4       EQU   4
R11      EQU   11
R12      EQU   12
R13      EQU   13
*
DFHEISTG DSECT
SEGKEYA  DS    CL4
SEGKEYB  DS    CL4
SEGKEYC  DS    CL4
SEGKEY1  DS    CL4
SEGKEY2  DS    CL4
CONKEYB  DS    CL8
SEGNAME  DS    CL8
SEGLEN   DS    H
PCBNUM   DS    H
AREAA    DS    CL80
AREAB    DS    CL80
AREAC    DS    CL80
AREAG    DS    CL250
AREASTAT DS    CL360
         COPY MAPSET
*******************************************************************
*  INITIALIZATION
*  HANDLE ERROR CONDITIONS IN ERROR ROUTINE
*  HANDLE ABENDS (DLI ERROR STATUS CODES) IN ABEND ROUTINE
*  RECEIVE INPUT MESSAGE
*******************************************************************
SAMPLE   DFHEIENT CODEREG=(R2,R3),DATAREG=(R13,R12),EIBREG=R11
         EXEC CICS HANDLE CONDITION ERROR(ERRORS)
         EXEC CICS HANDLE ABEND LABEL(ABENDS)
         EXEC CICS RECEIVE MAP ('SAMPMAP') MAPSET('MAPSET')
*        ANALYZE INPUT MESSAGE AND PERFORM NON-DLI PROCESSING
*******************************************************************
*  SCHEDULE PSB NAMED 'SAMPLE1'
*******************************************************************
         EXEC DLI SCHD PSB(SAMPLE1)
         BAL   R4,TESTDIB          CHECK STATUS
*******************************************************************
*  RETRIEVE ROOT SEGMENT AND ALL ITS DEPENDANTS
*******************************************************************
         MVC   SEGKEYA,=C'A300'
         EXEC DLI GU USING PCB(1) SEGMENT(SEGA) INTO(AREAA)            X
               SEGLENGTH(80) WHERE(KEYA=SEGKEYA) FIELDLENGTH(4)
         BAL   R4,TESTDIB          CHECK STATUS
```

ASM example of EXEC DLI commands (continued)

```
GNPLOOP  EQU   *
         EXEC DLI GNP USING PCB(1) INTO(AREAG) SEGLENGTH(250)
         CLC   DIBSTAT,=C'GE'      LOOK FOR END
         BE    LOOPDONE            DONE AT 'GE'
         BAL   R4,TESTDIB          CHECK STATUS
         B     GNPLOOP
LOOPDONE EQU   *
*******************************************************************
*  INSERT NEW ROOT SEGMENT
*******************************************************************
         MVC   AREAA,=CL80'DATA FOR NEW SEGMENT INCLUDING KEY'
         EXEC DLI ISRT USING PCB(1) SEGMENT(SEGA) FROM(AREAA)          X
               SEGLENGTH(80)
         BAL   R4,TESTDIB          CHECK STATUS
*******************************************************************
*  RETRIEVE 3 SEGMENTS IN PATH AND REPLACE THEM
*******************************************************************
         MVC   SEGKEYA,=C'A200'
         MVC   SEGKEYB,=C'B240'
         MVC   SEGKEYC,=C'C241'
         EXEC DLI GU USING PCB(1)                                      X
               SEGMENT(SEGA) WHERE(KEYA=SEGKEYA) FIELDLENGTH(4)        X
               INTO(AREAA)                                             X
               SEGLENGTH(80)                                           X
               SEGMENT(SEGB) WHERE(KEYB=SEGKEYB) FIELDLENGTH(4)        X
               INTO(AREAB)                                             X
               SEGLENGTH(80)                                           X
               SEGMENT(SEGC) WHERE(KEYC=SEGKEYC) FIELDLENGTH(4)        X
               INTO(AREAC)                                             X
               SEGLENGTH(80)
         BAL   R4,TESTDIB
*
*        UPDATE FIELDS IN THE 3 SEGMENTS
*
         EXEC DLI REPL USING PCB(1)                                    X
               SEGMENT(SEGA) FROM(AREAA) SEGLENGTH(80)                 X
               SEGMENT(SEGB) FROM(AREAB) SEGLENGTH(80)                 X
               SEGMENT(SEGC) FROM(AREAC) SEGLENGTH(80)
         BAL   R4,TESTDIB          CHECK STATUS
*******************************************************************
*  INSERT NEW SEGMENT USING CONCATENATED KEY TO QUALIFY PARENT
*******************************************************************
         MVC   AREAC,=CL80'DATA FOR NEW SEGMENT INCLUDING KEY'
         MVC   CONKEYB,=C'A200B240'
         EXEC DLI ISRT USING PCB(1)                                    X
               SEGMENT(SEGB) KEYS(CONKEYB) KEYLENGTH(8)                X
               SEGMENT(SEGC) FROM(AREAC) SEGLENGTH(80)
         BAL   R4,TESTDIB          CHECK STATUS
*******************************************************************
*  RETRIEVE SEGMENT DIRECTLY USING CONCATENATED KEY
*  AND THEN DELETE IT AND ITS DEPENDANTS
*******************************************************************
         MVC   CONKEYB,=C'A200B230'
         EXEC DLI GU USING PCB(1)                                      X
```

ASM example of EXEC DLI commands (continued)

```
               SEGMENT(SEGB)                                           X
               KEYS(CONKEYB) KEYLENGTH(8)                              X
               INTO(AREAB) SEGLENGTH(80)
         BAL   R4,TESTDIB          CHECK STATUS
         EXEC DLI DLET USING PCB(1)                                    X
               SEGMENT(SEGB) SEGLENGTH(80) FROM(AREAB)
         BAL   R4,TESTDIB          CHECK STATUS
*******************************************************************
*  RETRIEVE SEGMENT BY QUALIFYING PARENT WITH CONCATENATED KEY,
*  OBJECT SEGMENT WITH WHERE OPTION USING A LITERAL,
*  AND THEN SET PARENTAGE
*
*  USE VARIABLES FOR PCB INDEX, SEGMENT NAME, AND SEGMENT LENGTH
*******************************************************************
         MVC   CONKEYB,=C'A200B230'
         MVC   SEGNAME,=CL8'SEGA'
         MVC   SEGLEN,=H'80'
         MVC   PCBNUM,=H'1'
         EXEC DLI GU USING PCB(PCBNUM)                                 X
               SEGMENT((SEGNAME))                                      X
               KEYS(CONKEYB) KEYLENGTH(8) SETPARENT                    X
               SEGMENT(SEGC) INTO(AREAC) SEGLENGTH(SEGLEN)             X
               WHERE(KEYC='C520')
         BAL   R4,TESTDIB          CHECK STATUS
*******************************************************************
*  RETRIEVE DATA BASE STATISTICS
*******************************************************************
         EXEC DLI STAT USING PCB(1) INTO(AREASTAT)                     X
               VSAM FORMATTED LENGTH(360)
         BAL   R4,TESTDIB          CHECK STATUS
*******************************************************************
*  RETRIEVE ROOT SEGMENT USING BOOLEAN OPERATORS
*******************************************************************
         MVC   SEGKEY1,=C'A050'
         MVC   SEGKEY2,=C'A150'
         EXEC DLI GU USING PCB(1) SEGMENT(SEGA) INTO(AREAA)            X
               SEGLENGTH(80) FIELDLENGTH(4,4,4,4)                      X
               WHERE(KEYA>SEGKEY1 AND KEYA<SEGKEY2 OR                  X
               KEYA>'A275' AND  KEYA<'A350')
         BAL   R4,TESTDIB          CHECK STATUS
*******************************************************************
*  TERMINATE PSB WHEN DLI PROCESSING IS COMPLETED
*******************************************************************
         EXEC DLI TERM
*******************************************************************
*  SEND OUTPUT MESSAGE
*******************************************************************
         EXEC CICS SEND MAP('SAMPMAP') MAPSET('MAPSET')
         EXEC CICS WAIT TERMINAL
*******************************************************************
*  COMPLETE TRANSACTION AND RETURN TO CICS
*******************************************************************
```

ASM example of EXEC DLI commands (continued)

```
         EXEC CICS RETURN
********************************************************************
*  CHECK STATUS IN DIB
********************************************************************
TESTDIB  EQU   *
         CLC   DIBSTAT,=C'  '      IS STATUS BLANK
         BER   R4                  YES - RETURN
*        HANDLE DLI STATUS CODES REPRESENTING EXCEPTIONAL CONDITIONS
*
         BR    R4                  RETURN
ERRORS   EQU   *
*        HANDLE ERROR CONDITIONS
*
ABENDS   EQU   *
*        HANDLE ABENDS INCLUDING DLI ERROR STATUS CODES
*
         END
```

COBOL example of EXEC DLI commands

```
CBL LIB,APOST,XOPTS(CICS,DLI)
      IDENTIFICATION DIVISION.
      PROGRAM-ID. SAMPLE.
     *            COBOL EXEC DLI ONLINE PROGRAM
      ENVIRONMENT DIVISION.
      CONFIGURATION SECTION.
      SOURCE-COMPUTER. IBM-370.
      OBJECT-COMPUTER. IBM-370.
      DATA DIVISION.
      WORKING-STORAGE SECTION.
      77  SEGKEYA                PIC X(4).
      77  SEGKEYB                PIC X(4).
      77  SEGKEYC                PIC X(4).
      77  SEGKEY1                PIC X(4).
      77  SEGKEY2                PIC X(4).
      77  SEGKEY3                PIC X(4).
      77  SEGKEY4                PIC X(4).
      77  CONKEYB                PIC X(8).
      77  SEGNAME                PIC X(8).
      77  SEGLEN                 COMP PIC S9(4).
      77  PCBNUM                 COMP PIC S9(4).
      01  AREAA                  PIC X(80).
     *    DEFINE SEGMENT I/O AREA
      01  AREAB                  PIC X(80).
      01  AREAC                  PIC X(80).
      01  AREAG                  PIC X(250).
      01  AREASTAT               PIC X(360).
          COPY MAPSET.
      PROCEDURE DIVISION.
     * ***************************************************************
     *  INITIALIZATION
     *  HANDLE ERROR CONDITIONS IN ERROR ROUTINE
     *  HANDLE ABENDS (DLI ERROR STATUS CODES) IN ABEND ROUTINE
     *  RECEIVE INPUT MESSAGE
     * ***************************************************************
          EXEC CICS HANDLE CONDITION ERROR(ERRORS) END-EXEC.
     *
          EXEC CICS HANDLE ABEND LABEL(ABENDS) END-EXEC.
     *
          EXEC CICS RECEIVE MAP ('SAMPMAP') MAPSET('MAPSET') END-EXEC.
     *    ANALYZE INPUT MESSAGE AND PERFORM NON-DLI PROCESSING
     * ***************************************************************
     *  SCHEDULE PSB NAMED 'SAMPLE1'
     * ***************************************************************
          EXEC DLI SCHD PSB(SAMPLE1) END-EXEC.
          PERFORM TEST-DIB THRU OK.
```

COBOL example of EXEC DLI commands (continued)

```
* ***************************************************************
*  RETRIEVE ROOT SEGMENT AND ALL ITS DEPENDANTS
* ***************************************************************
     MOVE 'A300' TO SEGKEYA.
     EXEC DLI GU USING PCB(1) SEGMENT(SEGA) INTO(AREAA)
          SEGLENGTH(80) WHERE(KEYA=SEGKEYA) FIELDLENGTH(4)
     END-EXEC.
     PERFORM TEST-DIB THRU OK.
 GNPLOOP.
     EXEC DLI GNP USING PCB(1) INTO(AREAG) SEGLENGTH(250)
     END-EXEC.
     IF DIBSTAT EQUAL TO 'GE' THEN GO TO LOOPDONE.
     PERFORM TEST-DIB THRU OK.
     GO TO GNPLOOP.
 LOOPDONE.
* ***************************************************************
*  INSERT NEW ROOT SEGMENT
* ***************************************************************
     MOVE 'DATA FOR NEW SEGMENT INCLUDING KEY' TO AREAA.
     EXEC DLI ISRT USING PCB(1) SEGMENT(SEGA) FROM(AREAA)
          SEGLENGTH(80) END-EXEC.
     PERFORM TEST-DIB THRU OK.
* ***************************************************************
* RETRIEVE 3 SEGMENTS IN PATH AND REPLACE THEM
* ***************************************************************
     MOVE 'A200' TO SEGKEYA.
     MOVE 'B240' TO SEGKEYB.
     MOVE 'C241' TO SEGKEYC.
     EXEC DLI GU USING PCB(1)
       SEGMENT(SEGA) WHERE(KEYA=SEGKEYA) FIELDLENGTH(4)
          INTO(AREAA)
          SEGLENGTH(80)
       SEGMENT(SEGB) WHERE(KEYB=SEGKEYB) FIELDLENGTH(4)
          INTO(AREAB)
          SEGLENGTH(80)
       SEGMENT(SEGC) WHERE(KEYC=SEGKEYC) FIELDLENGTH(4)
          INTO(AREAC)
          SEGLENGTH(80)
     END-EXEC.
     PERFORM TEST-DIB THRU OK.
*    UPDATE FIELDS IN THE 3 SEGMENTS
     EXEC DLI REPL USING PCB(1)
       SEGMENT(SEGA) FROM(AREAA) SEGLENGTH(80)
       SEGMENT(SEGB) FROM(AREAB) SEGLENGTH(80)
       SEGMENT(SEGC) FROM(AREAC) SEGLENGTH(80)
     END-EXEC.
     PERFORM TEST-DIB THRU OK.
* ***************************************************************
*  INSERT NEW SEGMENT USING CONCATENATED KEY TO QUALIFY PARENT
* ***************************************************************
     MOVE 'DATA FOR NEW SEGMENT INCLUDING KEY' TO AREAC.
     MOVE 'A200B240' TO CONKEYB.
     EXEC DLI ISRT USING PCB(1)
       SEGMENT(SEGB) KEYS(CONKEYB) KEYLENGTH(8)
       SEGMENT(SEGC) FROM(AREAC) SEGLENGTH(80)
     END-EXEC.
     PERFORM TEST-DIB THRU OK.
```

COBOL example of EXEC DLI commands (continued)

```
* ***************************************************************
*  RETRIEVE SEGMENT DIRECTLY USING CONCATENATED KEY
*  AND THEN DELETE IT AND ITS DEPENDANTS
* ***************************************************************
     MOVE 'A200B230' TO CONKEYB.
     EXEC DLI GU USING PCB(1)
       SEGMENT(SEGB)
          KEYS(CONKEYB) KEYLENGTH(8)
          INTO(AREAB) SEGLENGTH(80)
     END-EXEC.
     PERFORM TEST-DIB THRU OK.
     EXEC DLI DLET USING PCB(1)
       SEGMENT(SEGB) SEGLENGTH(80) FROM(AREAB) END-EXEC.
     PERFORM TEST-DIB THRU OK.
* ***************************************************************
*  RETRIEVE SEGMENT BY QUALIFYING PARENT WITH CONCATENATED KEY,
*  OBJECT SEGMENT WITH WHERE OPTION,
*  AND THEN SET PARENTAGE
*
*  USE VARIABLES FOR PCB INDEX, SEGMENT NAME, AND SEGMENT LENGTH
* ***************************************************************
     MOVE 'A200B230' TO CONKEYB.
     MOVE 'C520' TO SEGKEYC.
     MOVE 'SEGA' TO SEGNAME.
     MOVE 80 TO SEGLEN.
     MOVE 1 TO PCBNUM.
     EXEC DLI GU USING PCB(PCBNUM)
       SEGMENT((SEGNAME))
          KEYS(CONKEYB) KEYLENGTH(8) SETPARENT
       SEGMENT(SEGC) INTO(AREAC) SEGLENGTH(SEGLEN)
          WHERE(KEYC=SEGKEYC) FIELDLENGTH(4) END-EXEC.
     PERFORM TEST-DIB THRU OK.
* ***************************************************************
*  RETRIEVE DATA BASE STATISTICS
* ***************************************************************
     EXEC DLI STAT USING PCB(1) INTO(AREASTAT)
          VSAM FORMATTED LENGTH(360) END-EXEC.
     PERFORM TEST-DIB THRU OK.
* ***************************************************************
*  RETRIEVE ROOT SEGMENT USING BOOLEAN OPERATORS
* ***************************************************************
     MOVE 'A050' TO SEGKEY1.
     MOVE 'A150' TO SEGKEY2.
     MOVE 'A275' TO SEGKEY3.
     MOVE 'A350' TO SEGKEY4.
     EXEC DLI GU USING PCB(1) SEGMENT(SEGA) INTO(AREAA)
          SEGLENGTH(80) FIELDLENGTH(4,4,4,4)
          WHERE(KEYA>SEGKEY1 AND KEYA<SEGKEY2 OR
                KEYA>SEGKEY3 AND KEYA<SEGKEY4)
     END-EXEC.
     PERFORM TEST-DIB THRU OK.
* ***************************************************************
*  TERMINATE PSB WHEN DLI PROCESSING IS COMPLETED
* ***************************************************************
     EXEC DLI TERM END-EXEC.
* ***************************************************************
*  SEND OUTPUT MESSAGE
* ***************************************************************
```

COBOL example of EXEC DLI commands (continued)

```
     EXEC CICS SEND MAP('SAMPMAP') MAPSET('MAPSET') END-EXEC.
     EXEC CICS WAIT TERMINAL END-EXEC.
* ***************************************************************
*  COMPLETE TRANSACTION AND RETURN TO CICS
* ***************************************************************
     EXEC CICS RETURN END-EXEC.
* ***************************************************************
*  CHECK STATUS IN DIB
* ***************************************************************
 TEST-DIB.
     IF DIBSTAT EQUAL TO '  ' THEN GO TO OK.
*    HANDLE DLI STATUS CODES REPRESENTING EXCEPTIONAL CONDITIONS
 OK.
 ERRORS.
*    HANDLE ERROR CONDITIONS
 ABENDS.
*    HANDLE ABENDS INCLUDING DLI ERROR STATUS CODES
```

PL/I example of EXEC DLI commands

```
* PROCESS INCLUDE,GN,XOPTS(CICS,DLI);
 SAMPLE: PROCEDURE OPTIONS(MAIN);
     /* PROGRAM: PL/I EXEC DLI ONLINE PROGRAM                        */
 DCL      SEGKEYA                  CHAR (4);
 DCL      SEGKEYB                  CHAR (4);
 DCL      SEGKEYC                  CHAR (4);
 DCL      SEGKEY1                  CHAR (4);
 DCL      SEGKEY2                  CHAR (4);
 DCL      SEGKEY3                  CHAR (4);
 DCL      SEGKEY4                  CHAR (4);
 DCL      CONKEYB                  CHAR (8);
 DCL      SEGNAME                  CHAR (8);
 DCL      PCBNUM                   FIXED BIN (15);
 DCL 1    AREAA                    CHAR (80);
     /*   DEFINE SEGMENT I/O AREA                                    */
 DCL 1    AREAB                    CHAR (80);
 DCL 1    AREAC                    CHAR (80);
 DCL 1    AREAG                    CHAR (250);
 DCL 1    AREASTAT                 CHAR (360);
        %INCLUDE MAPSET
     /* ************************************************************ */
     /*  INITIALIZATION                                              */
     /*  HANDLE ERROR CONDITIONS IN ERROR ROUTINE                    */
     /*  HANDLE ABENDS (DLI ERROR STATUS CODES) IN ABEND PROGRAM     */
     /*  RECEIVE INPUT MESSAGE                                       */
     /* ************************************************************ */
     EXEC CICS HANDLE CONDITION ERROR(ERRORS);
     /*                                                              */
     EXEC CICS HANDLE ABEND PROGRAM('ABENDS');
     /*                                                              */
     EXEC CICS RECEIVE MAP ('SAMPMAP') MAPSET('MAPSET');
     /* ANALYZE INPUT MESSAGE AND PERFORM NON-DLI PROCESSING         */
     /* ************************************************************ */
     /*  SCHEDULE PSB NAMED 'SAMPLE1'                                */
     /* ************************************************************ */
     EXEC DLI SCHD PSB(SAMPLE1);
     CALL TEST_DIB;
     /* ************************************************************ */
     /*  RETRIEVE ROOT SEGMENT AND ALL ITS DEPENDANTS                */
     /* ************************************************************ */
     SEGKEYA = 'A300';
     EXEC DLI GU USING PCB(1) SEGMENT(SEGA) INTO(AREAA)
     WHERE(KEYA=SEGKEYA);
     CALL TEST_DIB;
 GNPLOOP:
     EXEC DLI GNP USING PCB(1) INTO(AREAG);
     IF DIBSTAT = 'GE' THEN GO TO LOOPDONE;
     CALL TEST_DIB;
     GO TO GNPLOOP;
 LOOPDONE:
     /* ************************************************************ */
     /*  INSERT NEW ROOT SEGMENT                                     */
     /* ************************************************************ */
```

PL/I example of EXEC DLI commands (continued)

```
AREAA = 'DATA FOR NEW SEGMENT INCLUDING KEY';
EXEC DLI ISRT USING PCB(1) SEGMENT(SEGA) FROM(AREAA);
CALL TEST_DIB;
/* ************************************************************ */
/*  RETRIEVE 3 SEGMENTS IN PATH AND REPLACE THEM                */
/* ************************************************************ */
SEGKEYA = 'A200';
SEGKEYB = 'B240';
SEGKEYC = 'C241';
EXEC DLI GU USING PCB(1)
  SEGMENT(SEGA) WHERE(KEYA=SEGKEYA)
     INTO(AREAA)
  SEGMENT(SEGB) WHERE(KEYB=SEGKEYB)
     INTO(AREAB)
  SEGMENT(SEGC) WHERE(KEYC=SEGKEYC)
     INTO(AREAC);
CALL TEST_DIB;
/* UPDATE FIELDS IN THE 3 SEGMENTS                              */
EXEC DLI REPL USING PCB(1)
  SEGMENT(SEGA) FROM(AREAA)
  SEGMENT(SEGB) FROM(AREAB)
  SEGMENT(SEGC) FROM(AREAC);
CALL TEST_DIB;
/* ************************************************************ */
/*  INSERT NEW SEGMENT USING CONCATENATED KEY TO QUALIFY PARENT */
/* ************************************************************ */
AREAC = 'DATA FOR NEW SEGMENT INCLUDING KEY';
CONKEYB = 'A200B240';
EXEC DLI ISRT USING PCB(1)
  SEGMENT(SEGB) KEYS(CONKEYB)
  SEGMENT(SEGC) FROM(AREAC);
CALL TEST_DIB;
/* ************************************************************ */
/*  RETRIEVE SEGMENT DIRECTLY USING CONCATENATED KEY            */
/*  AND THEN DELETE IT AND ITS DEPENDANTS                       */
/* ************************************************************ */
CONKEYB = 'A200B230';
EXEC DLI GU USING PCB(1)
  SEGMENT(SEGB)
     KEYS(CONKEYB)
     INTO(AREAB);
CALL TEST_DIB;
EXEC DLI DLET USING PCB(1)
  SEGMENT(SEGB) FROM(AREAB);
CALL TEST_DIB;
/* ************************************************************ */
/*  RETRIEVE SEGMENT BY QUALIFYING PARENT WITH CONCATENATED KEY,*/
/*  OBJECT SEGMENT WITH WHERE OPTION                            */
/*  AND THEN SET PARENTAGE                                      */
/*                                                              */
/*  USE VARIABLES FOR PCB INDEX, SEGMENT NAME                   */
/* ************************************************************ */
CONKEYB = 'A200B230';
```

PL/I example of EXEC DLI commands (continued)

```
    SEGNAME = 'SEGA';
    SEGKEYC = 'C520';
    PCBNUM = 1;
    EXEC DLI GU USING PCB(PCBNUM)
      SEGMENT((SEGNAME))
         KEYS(CONKEYB) SETPARENT
      SEGMENT(SEGC) INTO(AREAC)
         WHERE(KEYC=SEGKEYC);
    CALL TEST_DIB;
    /* ************************************************************ */
    /*  RETRIEVE DATA BASE STATISTICS                               */
    /* ************************************************************ */
    EXEC DLI STAT USING PCB(1) INTO(AREASTAT) VSAM FORMATTED;
    CALL TEST_DIB;
    /* ************************************************************ */
    /*  RETRIEVE ROOT SEGMENT USING BOOLEAN OPERATORS               */
    /* ************************************************************ */
    SEGKEY1 = 'A050';
    SEGKEY2 = 'A150';
    SEGKEY3 = 'A275';
    SEGKEY4 = 'A350';
    EXEC DLI GU USING PCB(1) SEGMENT(SEGA) INTO(AREAA)
         WHERE(KEYA>SEGKEY1 AND KEYA<SEGKEY2 OR
               KEYA>SEGKEY3 AND KEYA<SEGKEY4);
    CALL TEST_DIB;
    /* ************************************************************ */
    /*  TERMINATE PSB WHEN DLI PROCESSING IS COMPLETED              */
    /* ************************************************************ */
    EXEC DLI TERM;
    /* ************************************************************ */
    /*  SEND OUTPUT MESSAGE                                         */
    /* ************************************************************ */
    EXEC CICS SEND MAP('SAMPMAP') MAPSET('MAPSET');
    EXEC CICS WAIT TERMINAL;
    /* ************************************************************ */
    /*  COMPLETE TRANSACTION AND RETURN TO CICS                     */
    /* ************************************************************ */
    EXEC CICS RETURN;
    /* ************************************************************ */
    /*  CHECK STATUS IN DIB                                         */
    /* ************************************************************ */
TEST_DIB: PROCEDURE;
    IF DIBSTAT = '  ' THEN GO TO OK;
    /* HANDLE DLI STATUS CODES REPRESENTING EXCEPTIONAL CONDITIONS  */
OK:
END;
ERRORS:
    /* HANDLE ERROR CONDITIONS                                      */
END SAMPLE;
```

Chapter 2.6. DL/I services (DL/I CALL statement)

For assembler language, COBOL, and PL/I application programs using the EXEC CICS command level interface, DL/I CALL statements are similar to DL/I database CALL statements running in batch mode or under IMS/VS data communication. (For assembler-language application programs, the CALLDLI macro, rather than the CALL macro, should be used when running under CICS.)

However, the DL/I command level interface provides a simpler method (using the EXEC DLI command) of accessing DL/I databases.

This chapter describes the DL/I CALL statement method of accessing DL/I databases in an online environment only. The use of the EXEC DLI command is described in "Chapter 2.5. DL/I services (EXEC DLI command)" on page 101.

The two methods of accessing DL/I databases cannot both be used in the same task. However, it is possible for different tasks in the same system to use different methods.

The CICS application program can request DL/I services by means of a DL/I CALL statement. In response to such a request, control is passed to a CICS-DL/I routine that acts as an interface between the CICS application program and DL/I. This interface routine checks the validity of the CALL list, sets up DL/I to handle the request, and passes control and the CALL list to DL/I. When the interface routine regains control, it, in its turn, returns control to the calling program, unless a DL/I pseudoabend has occurred, in which case the CICS task is abnormally terminated.

| Application programs may reside above the 16-megabyte line but, for DL/I calls or EXEC DLI commands, the save area, parameter list, and all the parameters must be below the 16-megabyte line.

Under CICS, two or more tasks may require access to the same application program at the same time. Because CICS application programs must be quasi-reenterable, DL/I areas that may be modified under CICS, such as PCB pointers, segment search arguments, and I/O work areas, should be placed in dynamic storage. For assembler language this will be in the DFHEISTG DSECT, for COBOL in working storage, and for PL/I in AUTOMATIC storage.

The DL/I database access capabilities of a CICS application program are defined in a program specification block (PSB) which is created, by the system programmer, by means of a PSB generation utility program.

The PSB contains one or more program communication blocks (PCBs) that describe the database access requirements of each DL/I database to be accessed by the application program.

A CICS application program designed to access DL/I databases must schedule its access to DL/I. Scheduling involves, for example, ensuring that the PSB is valid, that the application is not already scheduled, that the referenced databases are open and enabled, and that there is no intent conflict between the PSB and already scheduled PSBs from other application programs. Negative responses to any of the above will prevent scheduling.

The scheduling call, if successful, returns a list of addresses of the PCBs within the scheduled PSB. The application program in a subsequent CALL statement can specify, from this list, the address of the PCB corresponding to the database to be accessed.

If the scheduling call is unsuccessful, an INVREQ (invalid request) indicator is returned in response to subsequent DL/I CALL statements in the application program.

A task may schedule only one PSB at a time. Any attempt to schedule a second PSB while one is still scheduled causes the INVREQ indicator to be returned.

A sync point request (see "Chapter 5.6. Recovery (sync points)" on page 331) by a task that is scheduled to use DL/I resources implies the release of those resources. This means that if, after issuing a sync point request, access to a DL/I database is required, the PSB must be rescheduled. The previous position of the database has been lost.

To access DL/I databases, the following steps are required.

1. Issue a DL/I call to schedule the PSB and obtain PCB addresses.
2. Issue a DL/I call to access the required database.
3. Check the status code and UIBRCODE immediately following each DL/I call.
4. Issue a DL/I call, when all DL/I access is complete, to terminate the connection by releasing the PSB. The DL/I call also causes a sync point to be taken. (Otherwise, the PSB is released automatically when the transaction is terminated.)

User interface block (UIB)

The CICS-DL/I routine that acts as the interface between the CICS application program and DL/I passes information to the application program in a User Interface Block (UIB). A definition of the UIB should only be included in the application program if the UIB is to be referenced. The UIB is acquired by the interface routine when an application program issues a schedule request specifying a pointer reference to be set with the address of the UIB. The UIB contains the address of the PCB address list (UIBPCBAL) from the schedule request and, for each DL/I request, the response (UIBRCODE) from the interface routine, as follows:

Fields of the UIB

Field	ASM	COBOL	PL/I
UIBPCBAL	DS A	PIC 9(8) COMP	POINTER
UIBRCODE	DS 0XL2	PIC XX	
UIBFCTR	DS X	PIC X	BIT(8)
UIBDLTR	DS X	PIC X	BIT(8)

The fields UIBFCTR and UIBDLTR are overlays for the first and second bytes respectively of the return code (see "Check the response to a DL/I CALL" on page 120).

ASM
The UIB definition is included by invoking the DLIUIB macro.

COBOL
The UIB definition is included by a COPY DLIUIB statement in the linkage section of the program.

PL/I
The UIB definition is included by a %INCLUDE DLIUIB statement.

Examples of these are given at the end of the chapter. A COBOL application program must not include both DFHTCADS and DLIUIB DSECTS, otherwise duplicate labels will be generated.

Schedule the PSB and obtain PCB addresses

The format of the CALL statement to request scheduling of the PSB and to obtain the associated PCB addresses is as follows:

ASM:

```
CALLDLI ASMTDLI,([parmcount,]
        function,psbname,ptr-ref)
```

COBOL:

```
CALL 'CBLTDLI' USING [parmcount,]
        function,psbname,ptr-ref
```

PL/I:

```
CALL PLITDLI(parmcount,
        function,psbname,ptr-ref)
```

where:

'parmcount'
is a binary fullword containing a count of the arguments that follow. (Required only for PL/I.)

'function'
is the name of the field containing the four-character function 'PCBb'.

'psbname'
is an 8-byte field containing the PSB generation name (1 through 8 characters) accessed by the application program. It is left justified and padded right with blanks as appropriate.

If the PSB name is specified as '*' padded right with blanks, a default name is supplied. This default is the name of the application program associated with this task in the CICS program control table (the PCT).

If the call is successful, field UIBPCBAL in the UIB will contain the address of the list of PCB addresses. The order of the addresses is the same as the PCBs within the PSB as specified when the PSB is generated.

If the call is unsuccessful, the reason for the failure will be indicated in field UIBRCODE in the UIB.

'ptr-ref'
is a pointer reference that will be set to the address of the UIB after the call has been processed. The UIB contains the address of the PCB address list and the response from the CICS-DL/I interface.

Segment search arguments

Segment search arguments (SSAs) are used to identify segments of a DL/I database. SSAs may be simple segment names or they may be qualified to include constraints made upon the values of fields within the named segment types.

All IMS/VS database command codes are supported, including the "Q" code (although corresponding dequeuing must be performed by a CICS sync point, or by a DL/I TERM call, because the DEQ call is not supported).

For information on how to build an SSA, see *IMS/VS Application Programming for CICS/OS/VS Users.*

Except for a read only operation, when it is unnecessary, SSAs used by a CICS application program must be in dynamic storage because of the requirement for the program to be quasi-reenterable.

- For assembler language programs, the SSAs should be placed in the dummy section called DFHEISTG.
- For COBOL programs, the SSAs should be in the working storage section.
- For PL/I programs, the SSAs should be in AUTOMATIC storage.

I/O work area for DL/I segments

An I/O work area is required by DL/I to hold the segment being retrieved or to hold the segment being written to the database. Like SSAs, this work area must be in dynamic storage. The address of the work area is specified as the address of the first byte of the data area.

Issue a DL/I database call

The format of the CALL statement to request DL/I services is as follows:

ASM:

```
CALLDLI ASMTDLI[,([parmcount,]function
        ,pcb,workarea[,ssa1,ssa2,...])]
```

COBOL:

```
CALL 'CBLTDLI' USING [parmcount,]
     function,pcb,
     workarea[,ssa1,ssa2,...]
```

PL/I:

```
CALL PLITDLI ([parmcount,]function
     ,pcb,workarea[,ssa1,ssa2,...])
```

where:

'parmcount'
is the name of a binary fullword containing a count of the arguments that follow.

'function'
is the 2 through 4 byte name of the function to be performed. Valid function names for a CICS application program are as follows:

'GU'
get a unique segment identified by SSAs.

'GN'
get the next segment in the database, optionally qualified by SSAs.

'GNP'
get the next segment within the scope of the current hierarchy in the database, optionally qualified by SSAs.

'GHU'
as for "GU", but in addition, hold the segment for subsequent update.

'GHN'
as for "GN", but in addition, hold the segment for subsequent update.

'GHNP'
as for "GNP", but in addition, hold the segment for subsequent update.

'ISRT'
insert a new segment at the current position; also used in the initial load of a database.

'REPL'
replace a segment at the current position.

'DLET'
delete the segment at the current position.

'pcb'
is a field containing the address of the PCB corresponding to the database specified in the call. This address is one of the addresses returned in the address list by the scheduling call.

'workarea'
specifies the work area that contains the segment being passed to DL/I or is to contain the segment being retrieved from DL/I.

'ssa1,ssa2,...'
are the names of the segment search arguments.

For details of these calls, see *IMS/VS Application Programming for CICS/OS/VS Users.*

Terminate a PSB in the CICS application program

When all DL/I operations have been completed, the PSB should be terminated (or released). This is done either by issuing an explicit termination call, or on termination of the task. The releasing application program can reuse the PSB or a different PSB as required.

The DL/I CALL statement is used to terminate a PSB. It causes all database records used by the application program, and all associated log records to be written out. It also causes a CICS sync point to be taken, which commits all activity performed by this task, both related to DL/I and to CICS recoverable resources.

Changes performed prior to the execution of the command will not be backed out either in the event of dynamic transaction backout for a single failing task, or in the event of an emergency restart following an abnormal termination of the system. A CICS sync point generates implicitly a DL/I termination call. (A sync point is specified by the SYNCPOINT command, as described in "Chapter 5.6. Recovery (sync points)" on page 331.) CALL statements and sync points should be specified only at points in the transaction where logically related processing ends.

The PSB must be rescheduled explicitly after it has been terminated (by a CALL or sync point) if further access to the database is required, because the position of the database has been lost by the release mechanism.

The format of the CALL statement to terminate a PSB is as follows:

ASM:

```
CALLDLI ASMTDLI,([parmcount,]function)
```

COBOL:

```
CALL 'CBLTDLI' USING [parmcount,]function
```

PL/I:

```
CALL PLITDLI (parmcount,function);
```

where:

'parmcount'
is the name of the binary fullword containing the parameter count value of one.

'function'
is the name of the field containing the four-character function 'TERM' or 'Tbbb'.

Check the response to a DL/I CALL

The response to a DL/I CALL statement should always be checked so that, if unsuccessful, alternative processing can be initiated.

Check the CICS-DL/I response codes in UIBRCODE

First use the response code in field UIBRCODE in the UIB for the task to check that the CICS-DL/I interface has been used correctly by the application program (for example, the required PSB not being found in the directory of PSBs would cause a response code to be returned).

Initially, one of three response codes appears in field UIBFCTR, as follows:

Contents of field UIBFCTR

Condition	ASM	COBOL	PL/I
NORESP	X'00'	12-0-1-8-9	00000000
INVREQ	X'08'	12-8-9	00001000
NOTOPEN	X'0C'	12-4-8-9	00001100

NORESP means "normal response". For the two responses INVREQ and NOTOPEN, further information appears in field UIBDLTR, as shown below. (UIBFCTR and UIBDLTR are known collectively as UIBRCODE.)

If the code for INVREQ appears in UIBFCTR, one of the following codes appears in UIBDLTR:

Field UIBDLTR for INVREQ

Condition	ASM	COBOL	PL/I
Invalid argument passed to DL/I	X'00'	12-0-1-8-9	00000000
PSBNF	X'01'	12-1-9	00000001
PSBSCH	X'03'	12-3-9	00000011
PSBFAIL	X'05'	12-5-9	00000101
TERMNS	X'07'	12-7-9	00000111
FUNCNS	X'08'	12-8-9	00001000
MPS batch	X'09'	12-9-9	00001001
DLINA	X'FF'	12-11-0-7-8-9	11111111

The above conditions have the following meanings:

PSBNF	PSB not found
PSBSCH	PSB scheduled
PSBFAIL	PSB initialization failed
TERMNS	PSB terminate failure
FUNCNS	function unscheduled
MPS batch	an MPS batch program attempted to issue a PCB call for a read-only PSB or for a nonexclusive PSB if program isolation is active
DLINA	DL/I not active

If the code for NOTOPEN appears in UIBFCTR, one of the following codes appears in UIBDLTR:

Field UIBDLTR for NOTOPEN

Condition	ASM	COBOL	PL/I
Intent scheduling conflict	X'02'	12-2-9	00000010

NOTOPEN will never be returned to the application. If a schedule request is made against a closed database, PSBFAIL will be returned. A database cannot be closed until all activity against it has been quiesced. While this is taking place, no further scheduling is allowed.

If fields UIBFCTR and UIBDLTR are normal, examine the DL/I status codes in the program control block (the PCB). These status codes are listed in the *IMS/VS Application Programming for CICS/OS/VS User's* manual.

Check the DL/I function

You can also check that the specified DL/I function has been performed correctly according to the rules of DL/I (for example, a segment that cannot be located from the specified SSA would cause an error indication). This type of error is detected internally by DL/I and is explained in the appropriate DL/I application programming reference manual.

DL/I may also issue a pseudoabend which causes the task to be terminated rather than control to be returned to the CICS application program. The task is terminated with an ABEND code of ADLA.

Example of DL/I request using call

The following example shows, in assembler language, COBOL, and PL/I, the use of DL/I CALL statements in a CICS application program to request DL/I services.

ASM example of DL/I call

```
DFHEISTG DSECT
UIBPTR   DS    F
IOAREA   DS    0CL40
AREA1    DS    CL3
AREA2    DS    CL37
         DLIUIB
         USING UIB,8
PCBPTRS  DSECT
*        PSB ADDRESS LIST
PCB1PTR  DS    F
PCB1     DSECT
         USING PCB1,6
DBPC1DBD DS    CL8
DBPC1LEV DS    CL2
DBPC1STC DS    CL2
DBPC1PRO DS    CL4
DBPC1RSV DS    F
DBPC1SFD DS    CL8
DBPC1MKL DS    F
DBPC1NSS DS    F
DBPC1KFD DS    0CL256
DBPC1NM  DS    0CL12
DBPC1NMA DS    0CL14
DBPC1NMP DS    CL17
ASMUIB   CSECT
         B     SKIP
PSBNAME  DC    CL8'ASMPSB'
PCBFUN   DC    CL4'PCB'
REPLFUN  DC    CL4'REPL'
TERMFUN  DC    CL4'TERM'
GHUFUN   DC    CL4'GHU'
SSA1     DC    CL9'AAAA4444'
GOODRC   DC    XL1'00'
GOODSC   DC    CL2'  '
SKIP     DS    0H
*              SCHEDULE PSB AND OBTAIN PCB ADDRESSES
         CALLDLI ASMTDLI,(PCBFUN,PSBNAME,UIBPTR)
         L     8,UIBPTR
         CLC   UIBFCTR,X'00'
         BNE   ERROR1
*              GET PSB ADDRESS LIST
         L     4,UIBPCBAL
         USING PCBPTRS,4
*              GET ADDRESS OF FIRST PCB IN LIST
         L     6,PCB1PTR
*              ISSUE DL/I CALL: GET A UNIQUE SEGMENT
         CALLDLI ASMTDLI,(GHUFUN,PCB1,IOAREA,SSA1)
         CLC   UIBFCTR,GOODRC
         BNE   ERROR2
         CLC   DBPC1STC,GOODSC
         BNE   ERROR3
*              PERFORM SEGMENT UPDATE ACTIVITY
         MVC   AREA1,.......
         MVC   AREA2,.......
```

ASM example of DL/I call (continued)

```
*        ISSUE DL/I CALL: REPLACE SEGMENT AT CURRENT POSITION
         CALLDLI ASMTDLI,(REPLFUN,PCB1,IOAREA,SSA1)
         CLC   UIBFCTR,GOODRC
         BNE   ERROR4
         CLC   DBPC1STC,GOODSC
         BNE   ERROR5
         B     TERM
ERROR1   DS    0H
*              INSERT ERROR DIAGNOSTIC CODE
         B     TERM
ERROR2   DS    0H
*              INSERT ERROR DIAGNOSTIC CODE
         B     TERM
ERROR3   DS    0H
*              INSERT ERROR DIAGNOSTIC CODE
         B     TERM
ERROR4   DS    0H
*              INSERT ERROR DIAGNOSTIC CODE
         B     TERM
ERROR5   DS    0H
*              INSERT ERROR DIAGNOSTIC CODE
TERM     DS    0H
*              RELEASE THE PSB
         CALLDLI ASMTDLI,(TERMFUN)
         EXEC CICS RETURN
         END   ASMUIB
```

COBOL example of DL/I call

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. 'CBLUIB'.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77  PSB-NAME PIC X(8) VALUE 'CBLPSB  '.
       77  PCB-FUNCTION PIC X(4) VALUE 'PCB '.
       77  TERM-FUNCTION PIC X(4) VALUE 'TERM'.
       77  GHU-FUNCTION PIC X(4) VALUE 'GHU '.
       77  REPL-FUNCTION PIC X(4) VALUE 'REPL'.
       77  SSA1 PIC X(9) VALUE 'AAAA4444 '.
       77  SUCCESS-MESSAGE PIC X(40).
       77  GOOD-STATUS-CODE PIC XX VALUE '  '.
       77  GOOD-RETURN-CODE PIC X VALUE LOW-VALUE.
       01  MESSAGE.
           02 MESSAGE1 PIC X(38).
           02 MESSAGE2 PIC XX.
       01  DLI-IO-AREA.
           02 AREA1 PIC X(3).
           02 AREA2 PIC X(37).
       LINKAGE SECTION.
(1)    01  BLLCELLS.
           02 FILLER PIC S9(8) COMP.
           02 UIB-PTR PIC S9(8) COMP .
           02 B-PCB-PTRS PIC S9(8) COMP.
           02 PCB1-PTR PIC S9(8) COMP.
(2)        COPY DLIUIB.
(3)    01  PCB-PTRS.
           02 B-PCB1-PTR PIC 9(8) COMP.
(4)    01  PCB1.
           02 PCB1-DBD-NAME PIC X(8).
           02 PCB1-SEG-LEVEL PIC XX.
           02 PCB1-STATUS-CODE PIC XX.
           02 PCB1-PROC-OPT PIC XXXX.
           02 FILLER PIC S9(5) COMP.
           02 PCB1-SEG-NAME PIC X(8).
           02 PCB1-LEN-KFB PIC S9(5) COMP.
           02 PCB1-NU-SENSEG PIC S9(5) COMP.
           02 PCB1-KEY-FB PIC X(256).
       PROCEDURE DIVISION.
      *       SCHEDULE PSB AND OBTAIN PCB ADDRESSES
            CALL 'CBLTDLI' USING PCB-FUNCTION, PSB-NAME, UIB-PTR.
            IF UIBFCTR IS NOT EQUAL LOW-VALUES THEN
      *       INSERT ERROR DIAGNOSTIC CODE
              EXEC CICS RETURN END-EXEC.
            MOVE UIBPCBAL TO B-PCB-PTRS.
            MOVE B-PCB1-PTR TO PCB1-PTR.
      *       ISSUE DL/I CALL: GET A UNIQUE SEGMENT
            CALL 'CBLTDLI' USING GHU-FUNCTION, PCB1, DLI-IO-AREA, SSA1.
            SERVICE RELOAD UIB-PTR
            IF UIBFCTR IS NOT EQUAL GOOD-RETURN-CODE THEN
      *       INSERT ERROR DIAGNOSTIC CODE
              EXEC CICS RETURN END-EXEC.
            IF PCB1-STATUS-CODE IS NOT EQUAL GOOD-STATUS-CODE THEN
      *       INSERT ERROR DIAGNOSTIC CODE
              EXEC CICS RETURN END-EXEC.
      *       PERFORM SEGMENT UPDATE ACTIVITY
```

COBOL example of DL/I call (continued)

```
            MOVE ....... TO AREA1.
            MOVE ....... TO AREA2.
      *       ISSUE DL/I CALL: REPLACE SEGMENT AT CURRENT POSITION
            CALL 'CBLTDLI' USING REPL-FUNCTION, PCB1, DLI-IO-AREA, SSA1.
            IF UIBFCTR IS NOT EQUAL GOOD-RETURN-CODE THEN
      *       INSERT ERROR DIAGNOSTIC CODE
              EXEC CICS RETURN END-EXEC.
            IF PCB1-STATUS-CODE IS NOT EQUAL GOOD-STATUS-CODE THEN
      *       INSERT ERROR DIAGNOSTIC CODE
              EXEC CICS RETURN END-EXEC.
      *       RELEASE THE PSB
            CALL 'CBLTDLI' USING TERM-FUNCTION.
             EXEC CICS RETURN END-EXEC.
```

Notes:

1. The linkage section must start with a definition of this type to provide addressability to a parameter list which will contain the addresses of storage outside the working storage of the application program. The first 02 level definition is used by CICS to provide addressability to the other fields in the list. There is a one to one correspondence between the other 02 level names and the 01 level data definitions in the linkage section.
2. This will be expanded as shown in "User interface block (UIB)" on page 117.
3. The UIB will return the address of an area containing the PCB addresses. This definition is required to obtain the actual PCB addresses.
4. The PCBs are defined in the linkage section.

PL/I example of DL/I call

```
    PLIUIB: PROC OPTIONS(MAIN);
     DCL PSB_NAME CHAR(8) STATIC INIT('PLIPSB  ');
     DCL PCB_FUNCTION CHAR(4) STATIC INIT('PCB ');
     DCL TERM_FUNCTION CHAR(4) STATIC INIT('TERM');
     DCL GHU_FUNCTION CHAR(4) STATIC INIT('GHU ');
     DCL REPL_FUNCTION CHAR(4) STATIC INIT('REPL');
     DCL SSA1 CHAR(9) AUTOMATIC INIT('AAAA4444 ');
     DCL PARM_CT_1 FIXED BIN(31) STATIC INIT(1);
     DCL PARM_CT_3 FIXED BIN(31) STATIC INIT(3);
     DCL PARM_CT_4 FIXED BIN(31) STATIC INIT(4);
     DCL GOOD_RETURN_CODE BIT(8) STATIC INIT('0'B);
     DCL GOOD_STATUS_CODE CHAR(2) STATIC INIT('  ');
(1)  %INCLUDE DLIUIB;
(2)  DCL 1 PCB_POINTERS BASED(UIBPCBAL),
           2 PCB1_PTR POINTER;
     DCL 1 DLI_IO_AREA,
           2 AREA1 CHAR(3),
           2 AREA2 CHAR(37);
(3)  DCL 1 PCB1 BASED(PCB1_PTR),
           2 PCB1_DBD_NAME CHAR(8),
           2 PCB1_SEG_LEVEL CHAR(2),
           2 PCB1_STATUS_CODE CHAR(2),
           2 PCB1_PROC_OPTIONS CHAR(4),
           2 PCB1_RESERVE_DLI FIXED BIN (31,0),
           2 PCB1_SEGNAME_FB CHAR(8),
           2 PCB1_LENGTH_FB_KEY FIXED BIN(31,0),
           2 PCB1_NUMB_SENS_SEGS FIXED BIN(31,0),
           2 PCB1_KEY_FB_AREA CHAR(17);
         /* SCHEDULE PSB AND OBTAIN PCB ADDRESSES */
       CALL PLITDLI(PARM_CT_3,PCB_FUNCTION,PSB_NAME,UIBPTR);
       IF UIBFCTR¬=GOOD_RETURN_CODE THEN DO;
         /* ISSUE DL/I CALL: GET A UNIQUE SEGMENT */
       END;
       CALL PLITDLI(PARM_CT_4,GHU_FUNCTION,PCB1,DLI_IO_AREA,SSA1);
       IF UIBFCTR¬=GOOD_RETURN_CODE THEN
          IF PCB1_STATUS_CODE=GOOD STATUS CODE THEN
       DO;
         /* PERFORM SEGMENT UPDATE ACTIVITY */
         /*  INSERT ERROR DIAGNOSTIC CODE  */
       END;
       IF PCB1_STATUS_CODE¬=GOOD_STATUS_CODE THEN DO;
         /*  INSERT ERROR DIAGNOSTIC CODE  */
       AREA1=.......;
       AREA2=.......;
         /* ISSUE DL/I: REPLACE SEGMENT AT CURRENT POSITION */
       CALL PLITDLI(PARM_CT_4,REPL_FUNCTION,PCB1,DLI_IO_AREA,SSA1);
       END;
    END;
       IF UIBFCTR¬=GOOD_RETURN_CODE THEN DO;
         /* ANALYZE UIB PROBLEM */
         .
         .
         /* ISSUE DIAGNOSTIC MESSAGE */
       END;
```

PL/I example of DL/I call (continued)

```
ELSE IF PCB1_STATUS_CODE¬=GOOD_STATUS_CODE THEN DO;
     /* EXAMINE PCB1_STATUS_CODE */
     .
     .
     /* ISSUE DIAGNOSTIC MESSAGE */
   END;
     /* RELEASE THE PSB */
   CALL PLITDLI(PARM_CT_1,TERM_FUNCTION);
   EXEC CICS RETURN;
   END PLIUIB;
```

Notes:

1. This will be expanded as shown in “User interface block (UIB)” on page 117.
2. The UIB will return the address of an area containing the PCB addresses. This definition is required to obtain the actual PCB addresses.
3. The PCBs are defined based on the addresses passed in the UIB.

Chapter 2.7. DL/I batch programs (shared DB)

The IMS batch application programmer must be aware of certain restrictions that exist when DL/I batch application programs run in a shared database environment under CICS. These restrictions, which apply to batch application programs and utilities, are as follows.

Two types of DL/I requests may be issued by a batch shared database application program:

- All database access calls and EXEC DLI commands (GN, GU, GNP, ISRT, REPL, and DLET)
- System service calls (CHKP, LOG, and ROLL) or the EXEC DLI CHKP command.

  The ROLL call results in message DFH3731 being issued. The system action for this message states that any updates to DL/I databases since the last checkpoint, or since the start of the job step, are backed out (assuming that the dynamic transaction backout facility is active in the CICS system). The batch job step is abended with a user abend code of 3731.

If the application program issues a CHKP and the CICS shared database session is in quiesce, that is, if the master terminal has issued

```
CEMT PERFORM SHUTDOWN
```

or

```
CEMT SET IRC CLOSE
```

the application program will terminate immediately after the CHKP with a user abend code of 3707 or 3708.

The first byte of a log record used in a LOG call must be equal to or greater than X'A0', as in IMS DB. An additional restriction when using shared database is that the second byte of the record must be X'00'. If these restrictions are not observed, a PCB status code of GL is returned, and the record is not logged.

IMS application programs that use GSAM program specification blocks (PCBs) or PCBs with PROCOPT=L or LS (that is, those used for loading a database) are not supported in the batch shared database environment.

In all other respects, IMS batch application programs run satisfactorily in a shared database session without being compiled or link edited again. IMS/360 application programs, however, are not supported. The IMS batch application program may reside above the 16-megabyte line but, for DL/I calls and EXEC DLI commands, the CALL parameter list, the CALL parameters, and the save area must reside below the line.

The IMS batch application programmer should be aware that resources used by batch programs must be released as soon as possible (by means of CHKP) so that online programs are not delayed by waiting for these resources.

Application programs that are used in a shared database session may issue SPIE and STAE macros. When a DL/I request is made by the application program, the batch region controller modules will issue their own SPIE macro for the duration of the request, and will then restore the user's SPIE, if any.

There are certain abnormal terminations from which recovery cannot be attempted. Indeed, in these situations, the batch region controller will have broken the link between the batch regions and CICS. Therefore, the user application program should not use a STAE (or ESTAE) exit unless the exit continues the abend.

If the user application program or STAE routine completes by returning to the batch region controller, the controller will assume that the application program has completed successfully and may indicate to CICS that any DL/I database updates should be committed rather than backed out.

If the application program wishes to indicate that the updates should be backed out, it should issue an operating system ABEND macro or a DL/I ROLL call. If a program check occurs and the user has no SPIE exit, an abend will be forced.

Use of PL/I SPIE and STAE options has the effect of raising the ERROR condition on program check and system abends respectively.

If the user requires dynamic transaction backout, the ERROR condition should be allowed to continue if an ON ERROR unit is present (the ON ERROR unit should not attempt to issue any DL/I calls).

If a PL/I application program completes with a return code of 2000 or more, it will be assumed that the application program has failed. An abend (user code 3734) will be issued and, if dynamic transaction backout is active, any DL/I updates will be backed out to the previous checkpoint (or to the start of the program if no previous checkpoint exists).

PL/I gives a return code of 2000 or more if the ERROR condition is raised and execution is allowed to continue.

Part 3. Data communication operations

Chapter 3.1. Introduction to data communication operations 133

Chapter 3.2-1. Introduction to basic mapping support 135

Chapter 3.2-2. Minimum function BMS 139

Chapter 3.2-3. Standard function BMS 159

Chapter 3.2-4. Full function BMS 173

Chapter 3.2-5. BMS macro and command reference summary 193

Chapter 3.3. Terminal control 221

Chapter 3.4. Batch data interchange 265

Chapter 3.1. Introduction to data communication operations

Three methods are available to the CICS application programmer for communicating with the terminals and logical units in the subsystems of the network that forms part of the CICS system. The methods dealt with are:

- Basic mapping support (BMS)
- Terminal control
- Batch data interchange.

See the *CICS/OS/VS IBM 3270 Data Stream Device Guide* for full details of programming for the IBM 3270 Information Display System and similar terminals, such as the IBM 8775 Display Terminal.

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. Also, in some cases, the overhead incurred to achieve data stream independence may outweigh the advantages. BMS is described in Chapters 3.2-1 through 3.2-5.

Terminal control is the basic method for communicating with devices, whereas both BMS and batch data interchange extend the facilities of terminal control to simplify further the handling of data streams. Both BMS and batch data interchange use 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 3.3. Terminal control" on page 221.

Batch data interchange provides commands and options that may be used, possibly in conjunction with BMS commands, to communicate with the 6670 logical unit and with the batch logical units of the 3770 and 3790 subsystems. Batch data interchange is described in "Chapter 3.4. Batch data interchange" on page 265.

Chapter 3.2-1. Introduction to basic mapping support

Basic mapping support (BMS) is an interface between CICS and its application programs. The three versions of BMS are:

- Minimum function BMS
- Standard function BMS
- Full function BMS.

Each version formats input and output display data in response to BMS commands in the application programs. To do this, it uses device information from CICS system tables and formatting information from maps you have prepared for your application program.

BMS commands have a simple, generalized form, because formatting information is stored separately, in what are called maps. This makes it easier to write your application programs and makes them less susceptible to changes to the system or its devices. Such changes can be made independently of your application programs simply by changing the maps.

A single BMS command in an application program can address more than one kind of device. This is because BMS gets information about the terminal from a system table. It interprets commands differently for different device types.

How BMS affects programming

A CICS application program does not use ordinary programming language commands to perform input and output. Instead, it uses BMS commands, or terminal control commands, or both. BMS provides most of the input and output facilities required by application programs, and is easier to use than terminal control. Nevertheless, you might have special requirements that favor the use of terminal control.

BMS allows you to separate the tasks of display design and CICS application programming. It interprets generalized application program output commands, and generates data streams for specific output devices. (Such data streams are said to be **device dependent.**) Conversely, it transforms incoming data streams to a form acceptable to application programs. It obtains information 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. The same BMS input or output commands in an application program can be used with different kinds of device.

You can design several versions of a display map, each exploiting the advantages of a different device. By defining data as having **field** format, you can use application program commands to address predefined fields in a display by name, without knowing the positions of those fields. The same fields must appear in all versions of a display, but can be arranged differently in different versions. This is most useful when an installation uses devices with a variety of screen sizes. A suffixing mechanism enables BMS to associate a display version with the kind of device to which it applies.

The process of changing field data to and from its displayable form is called **mapping**.

As an alternative to field data format, you can display data in **text** data format. Text format presents data as a series of lines on a display or printer. To format text data, BMS breaks the data into strings that are, as nearly as possible, the same width as the display device. Rather than breaking a word or character string that cannot fit at the end of a line, BMS places it whole at the beginning of the next line.

BMS maps

Maps tell BMS how to format (map) field data for display. They are not needed for text data output. Every BMS field data mapping command names a map that contains formatting (mapping) instructions. Each map has two forms, **physical** and **symbolic**.

BMS formats a display for a given device by embedding control characters in the data stream. A **physical map** tells it how to do this.

A **symbolic description map** is a source language data structure that the assembler or compiler uses to resolve source program references to fields in the map. Symbolic description maps are described in more detail in "Chapter 3.2-2. Minimum function BMS" on page 139.

The physical map and the symbolic description map are generated separately. However, a complete map definition contains enough data to produce both types. You only need to change the TYPE operand in the definition to produce one type rather than the other.

You will be told later how to produce physical maps and symbolic description maps.

Every map must be part of a map set, even a single map. You usually group related maps together into one of these map sets. You define a map set by coding a series of CICS macro instructions. The first of these macros defines the map set itself, the second defines the first or only map; and others define fields within those maps.

When a CICS task uses a map, CICS loads the entire map set that includes that map, into storage. The map set remains in storage until the task either ends or requests a map from a different map set. Obviously, if several maps are used by the same application or transaction, it makes sense to define them in the same map set, thus ensuring that CICS loads them all at once.

You could define a single map set to suit every terminal attached to a CICS system. However, it may be necessary or desirable to format the same data differently for different devices. For example, the same transaction can be initiated from displays of various screen sizes. Alternatively, a program might communicate with a device that has a special feature, such as screen partitioning. The map set definition macro enables you to associate a special version of a map set with a terminal type or model.

BMS map definition

You define map sets, maps, and fields within maps by means of the following macros:

```
DFHMSD - defines a map set
DFHMDI - defines a map
DFHMDF - defines a field
```

The macros define the size, shape, position, potential content, and characteristics of the various elements that make up a display. It is best to design the layout and content of a display before attempting to code the macros.

You can also use the IBM licensed program Screen Definition Facility/CICS (SDF/CICS) to define, edit, and generate BMS maps interactively. By using SDF/CICS, you avoid having to code map definition macros. SDF/CICS runs as a CICS application program. However, this description of BMS assumes that you are going to use the macros, and describes the procedure.

A map definition always starts with the DFHMSD macro. You use the TYPE=MAP operand to generate a physical map; you use the TYPE=DSECT operand to generate a symbolic description map, for example:

```
DFHMSD TYPE=DSECT
```

The DFHMSD macro starts the definition of the map set. It is always followed by a DFHMDI macro for the first (or only) map in the map set, for example:

```
DFHMSD TYPE=DSECT
DFHMDI ...
```

After the DFHMDI macro, there follow, optionally, one or more DFHMDF macros defining the individual fields within the map:

```
DFHMSD TYPE=DSECT
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. The sequence of macros would look like this:

```
DFHMSD TYPE=DSECT
DFHMDI ...
DFHMDF ...
DFHMDF ...
DFHMDI ...
DFHMDF ...
DFHMDF ...
DFHMDF ...
DFHMSD TYPE=FINAL
```

You specify attributes of map sets, maps, and fields by using operands in the appropriate macros. You can specify the same operand in more than one of the macros defining a single map, selecting a different value each time. For example, you can specify a value for COLOR in a field definition that is different from that in the corresponding map definition. Both can differ from the value in the map set definition.

An operand in a DFHMDF macro overrides, for that field, the same operand in a DFHMDI macro. Similarly, an operand of DFHMDI overrides, for a map, the same operand of DFHMSD. If an operand is omitted from DFHMDF, the macro will adopt the same operand value from DFHMDI. If it is omitted from both DFHMDF and DFHMDI, the operand in DFHMSD is adopted. If omitted altogether, an operand will adopt the default for DFHMDF.

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.

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. You assemble and link edit physical maps, storing them in the CICS program library. You also assemble symbolic description maps, but you store the assembler output from these, which is a source language data structure, in the source library. You copy the structure into any application program, that refers to the map set, before assembling or compiling that program. The TYPE operand of the DFHMSD macro governs whether the assembler produces a physical map or a symbolic description map.

A physical map must have an entry in the processing program table (the PPT) before it can be loaded by CICS. The simplest way of creating such an entry is by using the CEDA transaction. This transaction is described in the *CICS/MVS Resource Definition (Online)* manual. Alternatively, you can use the DFHPPT system macro, as described in the *CICS/MVS Resource Definition (Macro)* manual.

BMS commands

Input and output operations are performed by BMS in response to commands in your application program. These commands have a similar form to other CICS commands. A command requesting BMS services names the map containing the mapping information. Non-CICS application program statements, that is, normal assembler language, COBOL, or PL/I statements can refer to fields in a map by name. Using BMS commands, not only can you read and change the contents of fields, you can also determine or modify their attributes (for example, length or color).

Facilities provided by BMS

As stated earlier, BMS exists in three pregenerated versions: **minimum, standard,** and **full**. The version available on your system will have been decided before CICS system generation. Each version provides a different level of function, and therefore requires a different amount of virtual storage. The minimum version uses considerably less storage than the other two. You can only use minimum BMS at the command level. Both standard and full can be used at command and macro level.

If you use either the full version or the standard version, you can benefit from the size of the minimum version. This is because the minimum version is a discrete component of the other two, provides their most commonly used functions, and can be paged into real storage independently of their other component modules. This is likely to reduce the size of your CICS working set if most of your BMS requests can be satisfied by the minimum version.

The support provided by each version is as follows:

Minimum BMS

Function provided:

- SEND MAP command
- RECEIVE MAP command
- SEND CONTROL command
- Default and alternate screens
- Extended attributes
- Map set suffixes
- Screen coordination with null map
- Field and block data

Devices supported:

All 3270 displays and printers except SNA character string printers, which are defined as such in the TCT.

Standard BMS

Function provided:

- All function of minimum PLUS
- SEND TEXT command
- Outboard formats
- Partitions
- Control of an MSR
- NLEOM mode for 3270 printers
- Subsystem LDC controls

Devices supported:

All devices supported by BMS

Full BMS

Function provided:

- All function of standard PLUS
- Terminal operator paging
- Cumulative mapping
- Page overflow
- Cumulative text processing
- Message routing
- Message switching
- Returning BMS-generated data stream to program before output
- Report controller

Devices supported:

All devices supported by BMS

Sample programs

Appendixes D, E, and F contain sample programs that illustrate, among other things, various aspects of programming for BMS.

Read these programs as you study new topics. Each sample program illustrates different aspects of BMS. Some of the samples do not apply to minimum BMS. Others do not apply to standard BMS either.

Chapter 3.2-2. Minimum function BMS

Minimum function BMS supports the IBM 3270 and IBM 3270-like range of displays and printers (but not SCS printers). For convenience, 'minimum function BMS' will be shortened to 'minimum BMS'. This chapter introduces:

- The IBM 3270 Information Display System
- The principles of display layout design
- The way in which you specify display layouts to CICS
- The commands and options provided by minimum BMS for communicating with a display that has a predefined layout.

The information in this chapter applies equally well to the IBM 3270-like displays, for example the IBM 8775 Display Terminal.

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. Using BMS commands, you do not have to understand the format of the data stream. Nevertheless, you need to know the range of things the data stream allows you to do. This section describes the features of 3270 terminals, and discusses how you can use them. See the *CICS/OS/VS IBM 3270 Data Stream Device Guide* for more information on the 3270 data stream, and the features available on 3270 and 3270-like terminals.

Input operations

The operations you perform at a 3270 terminal need not always result in data being sent to the host processor. For example, you 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 have not provided any data.

Apart from the alphanumeric keys, the keys that you can press without sending data are:

- 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 and special features of individual models make it easier for you to enter data. The features are described in the *3270 Information Display System Operator's Guide*, GC27-2742 and the *IBM 8775 Display Terminal User's Guide*.

When you have typed data onto a display, you will probably want to send it to the host processor. You do this by one of the following:

- Pressing the ENTER key
- Pressing a program function (PF) key
- Using an identification card reader
- Using a magnetic slot reader and hand scanner
- Detecting a light-pen attention field.

Although CICS will send modified data when you press PF keys, the keys are not normally used for this. Generally, you assign a specified meaning to the key itself.

If you want to get the attention of the host processor without sending data, you can:

- Press the CLEAR key
- Press a program access (PA) key.

If you want to send data without having to enter it explicitly, you can:

- Use a light pen
- Press the cursor select key.

An attention identifier (AID) character is always sent to the host processor whenever a 3270 input operation is performed. This indicates the cause of the input operation.

CICS ensures that an application program receives 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 upon the design of the application program.

Output operations

A terminal can receive data from an application program, as well as send data to it. Some of the data can be displayed, the rest consists of device controls. By building data streams containing device controls, you can, for example:

- Sound the audible alarm (if the terminal has one)
- Unlock the keyboard for input
- Reset the modified data tag (the MDT) of each field
- Print the contents of a 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

An application program can divide a screen into more than one field. The fields combine to produce a complete screenful of data.

A field starts with an attribute character, continues with data characters, and ends at the next attribute character. A field can contain only a single character or it can span several lines, as the last character on a line is logically followed by the first character on the next line.

BMS does not allow a field to "wrap around" from the end of one line to the start of the next. Nor does it allow a field to "wrap around" from the bottom of the screen to the top.

Normally, an image is divided into several fields by the program but it is possible to have an image with no fields (no attribute characters). This occurs when you press the CLEAR key; such unformatted images are not supported by BMS. An application program can use the HANDLE AID command to detect the use of the CLEAR key. This command is described in "HANDLE AID command" on page 154. An attempt to read from a cleared screen raises the MAPFAIL condition.

Attribute character: The attribute character is always the first character of a field. It occupies a character position on the screen but appears as a blank. An extended data stream is used to communicate with a device that supports extended color, highlighting, programmed symbols, or validation. The single blank attribute character on a display produced by such a data stream can represent several attribute bytes.

Attribute bytes can convey the following field attributes.

- **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. Stopper fields following variable-length data fields are normally defined with protected attribute characters. If the operator attempts to enter more characters than the variable-length data field can contain, the stopper field following it will cause the keyboard to be locked.

- **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 is often used to highlight keywords, errors, or operator messages.

- **Base color**

  The IBM 3279 Model 2A or 3A display device 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.

- **Extended color**

  The IBM 3279 Model 2B or 3B uses **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.

  An IBM 3279 Model 2B or 3B will act as a Model 2A or 3A until it detects an extended color attribute byte in the data stream. It will display the image in default color or base color, according to the setting of the switch on the control panel.

  As soon as 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 will change 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 UNDERLINE.

- **Nondisplay**

  A nondisplay field does not display the data on the screen for operator viewing and does not print the field data. This might be used to enter security data when the screen is visible to others. This attribute characteristic should be used with care, as 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 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.

- **Programmed symbols**

  As well as the standard display symbol sets, the 3278, the 8775, the 3279 Model 2B or 3B, and the 3290 can have optional additional symbol store, enabling them to display up to six 191-character symbol sets, whose fonts and codes are defined by the user. Characters in different display fields can be selected from different symbol sets. This feature uses the extended data stream.

  The definition of the programmed symbols must be sent to the terminal before programmed symbols can be used. This is discussed further in the *CICS/OS/VS IBM 3270 Data Stream Device Guide*.

- **Light pen detectable**

  A light pen detectable field is sensitive to the light pen (a special feature) and the cursor select key. Two types of detectable fields are possible: a delayed detectable field and an immediately detectable field.

  If a delayed detectable field is selected by the operator using the light pen, the modified-data tag (MDT) is turned on. If an immediately detectable field is selected, the modified-data tag is turned on and transmission occurs. See *An Introduction to the IBM 3270 Information Display System*.

- **Validation**

  The extended data stream can be used to define an input field in an 8775 display as one of the following:

  Mandatory fill
  Input field must be filled before pressing ENTER.

  Mandatory enter
  The operator must key data into the input field before pressing ENTER.

  Trigger
  Every time the cursor leaves a data field that has the trigger attribute and that has been modified by the operator, the terminal transmits the contents of the field to CICS.

  Note: Although mandatory fill and mandatory enter force you to enter data into a field, your program can provide an "escape" mechanism for an operator who does not know what data to enter. Terminals that have the validation feature usually have ERROR keys to help you do this. Such keys generate the character X'3F'. Your program can test for this character in input fields whenever a validation operation is performed. The supplied DFHERROR constant makes it unnecessary for you to remember the character value. "Attribute constants" on page 150 shows how this and other constants can help you modify data structures.

- **Modified data tag (MDT)**

  The modified data tag is turned on when 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 light pen are transmitted to the processor. The program may send fields to the 3270 with the modified-data tag already on to guarantee that the field will be returned with the next transmission.

- **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. 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 is specified, the cursor is placed at position zero (row 1, column 1) on the screen.

- **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.

- **Field outlining**

  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.

- **SO/SI creation**

  Indicates 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 provided they are correctly paired. The terminal user can transmit them inbound if they are already present in the field, but he may only add them to an EBCDIC field if the field has the SOSI attribute.

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.

The following features of some 3270 displays make it easier for the layout designer to fulfil the requirements:

- Color
- Field highlighting
- Programmed symbols
- Easy correction
- Numeric shift for numeric data
- Validation
- Field delimiters or stoppers (to control the length of data entered).

The first step in designing 3270 screen layouts is to divide the screen into functional areas such as a title area, an application data area, and a message area.

Title area: The title area of a screen should identify the program that displayed the data. Data fields from the same file can appear in the same screen locations for different applications, permitting the operator to become familiar with fields by their screen location. A title can be used to help the operator recognize the application. The title area is normally the top one or two lines of the screen and may contain a page number (if more than one page is needed), field headings, and other data besides the title.

Application data area: The application data area comprises the main portion of the screen. Data from one or more records in the same file or multiple files is entered or displayed, depending on the application requirements.

Three kinds of field are usually found in this area: keyword, data, and stopper.

Keyword fields contain constant data sent by the program to identify the contents of a data field. For example, a keyword field containing 'ACCOUNT BALANCE:' might precede and identify a data field containing '$129.54'. A keyword field might also be used in a data entry application to identify the data being entered. For example,

```
QUANTITY:
```

means enter the quantity.

Data fields contain file data that the application program retrieves from files and displays. The data may appear exactly as stored in the file, or it may be edited by the program. Data fields may also be left blank for the operator to enter data. The application program can use the entered data to make changes to a record or to alter the processing of the program. In some cases, it may be appropriate for the program to display characters in an entry data field to guide the operator in entering the data. For example,

```
DATE: MMDDYY
```

means enter month, day, year, each having two characters.

Stopper fields (see "Attribute character" on page 140) on data entry screens restrict the length of the data fields. Stopper fields containing no data are used to define the space between data fields and to stop the operator from entering too many characters in a field.

For example, a field containing a street address may be 20 characters long, but for screen layout reasons an entire line of 40 characters is provided for this field. To prevent the operator from keying more than 20 characters on this line, the program should define a stopper field starting in the twenty-first position of the line. The stopper field should be protected from data entry to restrict the operator to the 20-character field.

The BMS map definition macros do not allow you to define a zero length field. Thus a stopper field occupies two screen positions, one for the attribute byte, and one for a blank data character.

Message area: The message area of a screen is used to send instruction messages to assist the operator in processing a transaction. This area should be separate from the application data area to allow communication with the operator, without disturbing the application data. The message area is normally the bottom one or two lines of the screen.

Screen sizes: The 3270 Information Display System is available in several screen sizes. Some 3270 devices are available with two screen sizes: the DEFAULT (small) size and the ALTERNATE (large) size. The system programmer specifies the screen sizes in the terminal control table (TCT), and specifies, in the program control table (PCT) for each transaction, which of the two possible sizes that transaction will use.

If ERASE is not specified on a terminal control or BMS output command, the screen will be unchanged from its previous screen size setting, that is, the previous transaction selection, or the default if the operator has just switched on or has cleared the screen.

In normal practice, this means that an application program should specify ERASE with its first output request. On receipt of a CLEAR key indication, CICS will preserve the selected screen size, so that an ERASE is not needed for output requests following the first.

Defining BMS maps

This section describes the three macros DFHMSD, DFHMDI, and DFHMDF, that 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 catalog this map set for use by application programs.

As you read about the macros, you will probably find it helps to study the sample map definitions in Appendixes D, E, and F.

You must code the macros according to assembler language source coding rules. 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.

Defining a map set

You use the DFHMSD macro to define a set of maps, that is, a 'map set'. The macro consists of operands that define characteristics of the map, or maps, comprising the map set. Some operands specified in DFHMSD can be overridden, for individual maps or fields that make up the map set, by operands in the map (DFHMDI) and field (DFHMDF) definition macros. The full syntax of the map set definition macro, and a description of its operands, is given in "Chapter 3.2-5. BMS macro and command reference summary" on page 193. However, using minimum BMS you will not use all of the operands.

You must produce at least two versions of any map set that you define. One must contain the operand TYPE=DSECT, the other TYPE=MAP. The map set source files can be stored before assembly. Alternatively, the value of the TYPE operand can be supplied by the SYSPARM mechanism before assembly. This avoids the need for two similar map sets, which differ only in their TYPE operand. The SYSPARM mechanism is described in the *CICS/MVS Installation Guide*.

Operands of DFHMSD: You use DFHMSD to specify the following:

- The name of your map set (by labeling the DFHMSD macro).
- Whether it is a physical map or a symbolic description map (TYPE operand).
- Whether it is to be used for input, output, or both (MODE operand).
- Which programming language will be used to code the application programs that will use the map or maps in the map set (LANG operand). If programs written in different languages are to use the same map set, you must prepare a separate version of the symbolic description map for each language.
- Whether maps within the map set will occupy the same area in storage, each overlaying the last as it is loaded (STORAGE and BASE operands). Assuming that you do not want them to overlay each other, you code STORAGE=AUTO. See "Getting storage for a data structure" on page 151.
- Whether 3270 terminal control commands (such as sounding the alarm) are to be activated during transmission (CTRL operand).
- Whether maps in the map set have attributes that use the extended data stream; for example, color, highlighting, programmed symbols, or validation (MAPATTS and DSATTS operands for maps and DSECTs respectively).

  EXTATT is accepted for compatibility with previous releases. KEXTATT is accepted for compatibility with the IBM Japan 5550 support feature.
- The default values of the extended attributes, where applicable (COLOR, HILIGHT, OUTLINE, PS, SOSI, TRANSP, and VALIDN operands).
- Whether the map set name is to be suffixed (TERM or SUFFIX operand). See "Assembling and cataloging BMS maps" on page 146.

The syntax of the DFHMSD macro is as follows:

```
mapset DFHMSD
    TYPE={DSECT|MAP}
   ,TIOAPFX=YES
   ,{STORAGE=AUTO|BASE=name}
  [,MODE={IN|OUT|INOUT}]
  [,LANG={ASM|COBOL|PLI|RPG}]
  [,CTRL=([PRINT]
     [,FREEKB][,ALARM][,FRSET])]

  [,MAPATTS=(attr1,attr2,...)]
  [,DSATTS=(attr1,attr2,...)]
  [,COLOR={DEFAULT|color}]
  [,VALIDN={[MUSTFILL][,MUSTENTER]
     [,TRIGGER]}]
  [,HILIGHT={OFF|BLINK|REVERSE|
  UNDERLINE}]
  [,PS={BASE|psid}]
  [,OUTLINE={BOX|([LEFT][,RIGHT]
    [,OVER][,UNDER])}]
  [,SOSI={NO|YES}]
  [,TRANSP={YES|NO}]

  [,SUFFIX=suffix|
   ,TERM={3270-1|3270-2}]
```

Most of the operands are self explanatory. For more information, see the description of the operand in "Map definition macro operand summary" on page 195.

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. It allows you to override some of the options inherited from DFHMSD, and to specify some new ones. The full syntax of the macro is shown in "Chapter 3.2-5. BMS macro and command reference summary" on page 193.

You use DFHMDI to specify the following:

- The name of a map (by labeling the DFHMDI macro).
- Its size; that is, the depth in number of lines and the width in number of columns. (SIZE operand.)
- The position (line and column) of its top left-hand corner. (LINE and COLUMN operands.)
- Whether 3270 terminal control commands (such as sounding the alarm) are to be activated during transmission (CTRL operand.)
- Whether maps in the map set have attributes that use the extended data stream; for example, color, highlighting, programmed symbols, or validation (MAPATTS and DSATTS operands for maps and DSECTs respectively).

  EXTATT is accepted for compatibility with previous releases. KEXTATT is accepted for compatibility with the the IBM Japan 5550 support feature.
- The default values of the extended attributes, where applicable (COLOR, HILIGHT, OUTLINE, PS, SOSI, TRANSP, and VALIDN operands.)
- That the map contains no fields; that is, it is a **null map**. You would find this useful if you wanted to reserve part of the screen for a program other than BMS (FIELDS operand).

The syntax of the DFHMDI macro is as follows:

```
map DFHMDI
  [,SIZE=(line,column)]
  [,LINE=number]
  [,COLUMN=number]
  [,JUSTIFY=BOTTOM]
  [,CTRL=([PRINT]
      [,FREEKB][,ALARM][,FRSET])]

  [,EXTATT={NO|MAPONLY|YES}]
  [,COLOR={DEFAULT|color}]
  [,VALIDN={[MUSTFILL][,MUSTENTER]
     [,TRIGGER]}]
  [,HILIGHT={OFF|BLINK|REVERSE|
         UNDERLINE}]
  [,PS={BASE|psid}]

  [,FIELDS={YES|NO}]

  [,SUFFIX=suffix|
   ,TERM={3270-1|3270-2}]

  [,MAPATTS=(attr1,attr2,...)]
  [,DSATTS=(attr1,attr2,...)]
  [,OUTLINE={BOX|([LEFT][,RIGHT]
    [,OVER][,UNDER])}]
  [,SOSI={NO|YES}]
  [,TRANSP={YES|NO}]
```

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. Ensure that you use unique names for maps within a map set, or within multiple map sets that are copied into one 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 screen. Therefore, 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. In this way, you can erase parts of the contents of the screen without affecting the rest.

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 either 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, BMS has no affect on data that appears in the reserved part of the screen. You would use null maps in this way if you wanted to build a composite display containing both BMS text data and graphics data. If the graphics data is produced by GDDM, ensure that a GDDM PSRSRV call is included to prevent programmed symbol sets that are being used by BMS from being corrupted by GDDM.

There are other considerations when coordinating use of a screen between BMS and other programs, see "Accessing data outside the program" on page 153.

Defining fields within a BMS map

The DFHMDF macro is used to specify initial attributes to be given to fields within a map.

You use DFHMDI to specify the following:

- The one-to-seven character name of the field. You only have to name fields if your application program refers to them. Only named fields appear in the symbolic description map.
- The position of the start of the field relative to the map origin. This is the position of the attribute byte for the field (POS operand.)
- The length of the field excluding its attribute byte (LENGTH operand). Specifying the length of a field does not cause BMS to delimit it with "stopper" fields; you must do that yourself either by making successive fields contiguous, or by inserting fields with ATTRB=PROT.

  A field cannot extend beyond the right-hand edge of the map, that is, it cannot "wrap" around the display.
- Whether data placed in the field is to be left-or right-justified. (JUSTIFY operand.)
- What character must be used to pad a justified field. (JUSTIFY=BLANK or JUSTIFY=ZERO.)
- The initial contents of the field. (INITIAL or XINIT operand.)
- Attributes of the field, for example, skip, protect, nondisplay. (ATTRB operand.)
- Extended data stream attributes of the field. (COLOR, HILIGHT, OUTLINE, PS, SOSI, TRANSP, and VALIDN operands.)
- A picture to be used to edit input. (PICIN and PICOUT operands.)

These functions are a subset of those provided by the standard and full BMS systems. The syntax of the full DFHMDF macro, and a definition of its operands can be found in "Chapter 3.2-5. BMS macro and command reference summary" on page 193.

The syntax of the DFHMDF macro is as follows:

```
fld  DFHMDF
 [,POS={number|(line,column)}]
 [,LENGTH=number]
 [,JUSTIFY=
   ([{LEFT|RIGHT}][,{BLANK|ZERO}])]
 [,INITIAL='character data'|
         XINIT='hexadecimal data']
 [,ATTRB=([{ASKIP|PROT|UNPROT
        [,NUM]}][,{BRT|NORM|DRK}]
        [,DET][,IC][,FSET])]
 [,CASE=MIXED]

 [,GRPNAME=group-name]
 [,OCCURS=number]
 [,PICIN='value']
 [,PICOUT='value']

 [,COLOR={DEFAULT|color}]
 [,VALIDN={[MUSTFILL][,MUSTENTER]
    [,TRIGGER]}]
 [,PS={BASE|psid}]
 [,HILIGHT={OFF|BLINK|REVERSE|
     UNDERLINE}]
 [,OUTLINE={BOX|([LEFT][,RIGHT]
   [,OVER][,UNDER])}]
 [,SOSI={NO|YES}]
 [,TRANSP={YES|NO}]
```

Field groups: Although most of the operands of DFHMDF explain themselves, the operands OCCURS, PICIN, PICOUT, and GRPNAME need more explanation. OCCURS, PICIN, and PICOUT are described in detail in "Map definition macro operand summary" on page 195. The GRPNAME operand will be described here.

Very often, an output data display field has to contain several subfields, all sharing the same display attributes, each of which 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, comprising a 'day' subfield, a 'month' subfield, and a 'year' subfield. The contents of the year subfield remain constant over a relatively long period. Its value can safely be taken from a map. However, the day value and month value must be updated regularly. 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 do not have to be attributes. If the subfields are positioned contiguously with no intervening blanks, the POS of the second and succeeding subfields must be the last characters of the previous subfield.

Terminating a map set definition

The macro DFHMSD TYPE=FINAL terminates a map set definition. It is coded as follows:

```
mapset DFHMSD TYPE=FINAL
```

The name of the map set, if specified, must match that specified by the DFHMSD TYPE=INITIAL macro.

Example of map set definitions

Appendixes D, E, and F contain sample programs in assembler language, COBOL, and PL/I respectively. The programs show various aspects of CICS application programming, including map definition. You will probably find it useful to study the sample map definitions now.

Assembling and cataloging BMS maps

You assemble a BMS map definition to generate either a symbolic description map or a physical map. The *CICS/MVS Installation Guide* describes how to assemble and catalog the maps.

Symbolic description map: A symbolic description map set definition (DFHMSD TYPE=DSECT) is assembled, and cataloged in the source statement library. The member name is usually the same as the map set name, but it need not be. Alternatively, the symbolic description map can be copied or inserted directly into the application program.

Physical map: A physical map set definition (DFHMSD TYPE=MAP) is assembled, link edited, and cataloged in the CICS load library.

When you catalog the physical map, consider whether to add a suffix to its name (specified with the NAME statement). The reason for suffixing a map is that you might want to produce alternative versions of it 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 wish to organize output data according to the terminal in use, making best use of its features, you ought to consider suffixing map sets.

For example, 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." You add a different suffix to each version of the same map.

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.

To understand why you might suffix a map set name, you must understand how CICS uses suffixes. Figure 21 on page 147 summarizes the map set selection logic used by BMS.

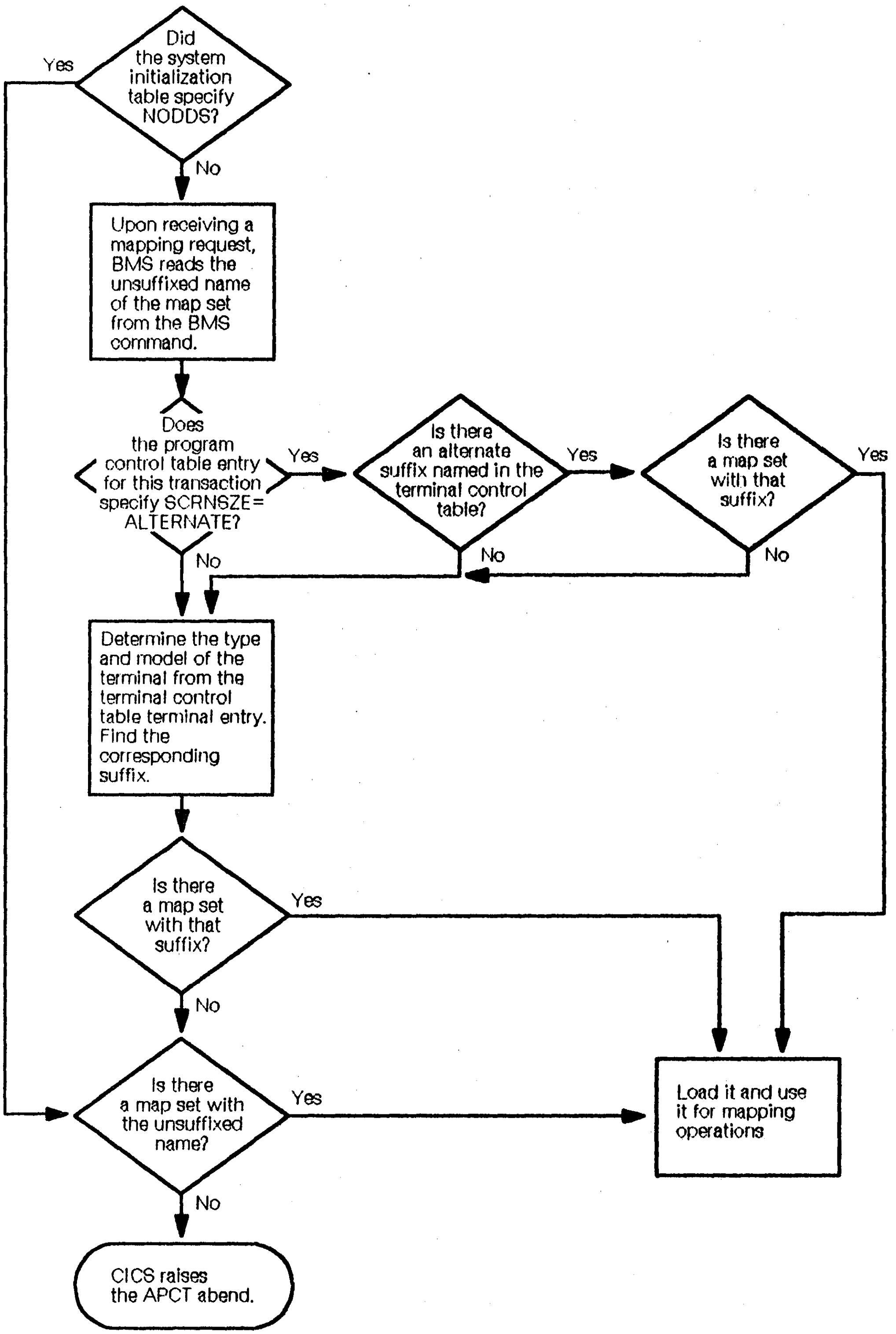

Did the system initialization table specify NODDS?
Yes
No
Upon receiving a mapping request, BMS reads the unsuffixed name of the map set from the BMS command.
Does the program control table entry for this transaction specify SCRNSZE= ALTERNATE?
Yes
No
Is there an alternate suffix named in the terminal control table?
Yes
No
Is there a map set with that suffix?
Yes
No
Determine the type and model of the terminal from the terminal control table terminal entry. Find the corresponding suffix.
Is there a map set with that suffix?
Yes
No
Is there a map set with the unsuffixed name?
Yes
No
Load it and use it for mapping operations
CICS raises the APCT abend.

Figure 21. BMS map set suffixing logic

Minimum BMS supports only 3270 terminals. There are only two TERM values that have meaning under minimum BMS: TERM=3270-1 and TERM=3270-2. Their maps should be suffixed L and M respectively.

By default, terminal type 3270-1 is a 40-column display; 3270-2 is an 80-column display. If your terminal has different characteristics, you must use a suffix of your own choice, using the SUFFIX operand. For example, a 3278 Model 5 display device has a screen that is 132 characters wide, and 27 characters deep. Maps that use the full display must have special suffixes. 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.

Finally, ask your system programmer to ensure that your physical maps are cataloged with the correct suffixes. In particular, note the following points about suffixing:

1. If you specify TERM or SUFFIX on your DFHMSD macro, ensure that the physical map set is cataloged using the correctly suffixed name.
2. You can code SUFFIX, instead of TERM, on DFHMSD if you need to create a special version of a map. By specifying SCRNSZE=ALTERNATE when you define the CEDA profile 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 ALTSFX operand of the TCT entry for the terminal. (If you do not use CEDA, you must code SCRNSZE=ALTERNATE on the PCT entry for your transaction.)

Table entries such as those described above are usually defined by a system programmer.

If all your map sets are unsuffixed, you get better performance if NODDS is specified for the BMS operand of the DFHSIT system macro. 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 21 on page 147 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. This section describes how this can be done.

Application programs use BMS SEND and RECEIVE commands to send and receive display data. This section 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.

BMS commands are not accepted by the assembler or compilers. They must be translated, as are other CICS commands, before being assembled or compiled.

Copying symbolic description maps

Earlier in this chapter, under "Defining BMS maps" on page 143, we describe how to define, assemble, and catalog the symbolic version of a map set. The cataloged version of a map set (the symbolic storage definition) is an application data structure, which must be copied into any application program that refers to fields in its maps.

Appendixes D, E, and F show examples of application program data structures for assembler language, COBOL, and PL/I respectively. It might help if you pause to study these before proceeding further.

The following examples show you how to copy these structures for each programming language. In these examples, mapsetname1, mapsetname2, and mapsetname3 are the names of members in the source library that contain the assembly of a BMS symbolic map definition. These member names are the same as the names used to catalog the symbolic description map set, as described in "Assembling and cataloging BMS maps" on page 146. The following examples assume that the source library has been assigned to SYSLIB by suitable job control language as described in the *CICS/MVS Installation Guide*.

1. An assembler language program must contain COPY instructions for each symbolic storage definition. You can specify that all definitions must occupy the same area (that is, by overlays). If this is what you want, the second and subsequent COPY instructions must be preceded by ORG instructions to reposition the assembler to the start of the data area.

```
COPY mapsetname1
COPY mapsetname2
COPY mapsetname3
```

2. A COBOL program must contain a COBOL COPY statement for each symbolic map definition. Generally, you should code the COPY statements in the working storage section of a program. This saves you from having to acquire storage for them.

```
WORKING STORAGE SECTION.
COPY mapsetname1.
COPY mapsetname2.
COPY mapsetname3.
⋮
```

3. A PL/I program must contain a %INCLUDE statement for each symbolic storage definition.

```
%INCLUDE mapsetname1;
%INCLUDE mapsetname2;
%INCLUDE mapsetname3;
⋮
```

Alternatively, the assembled symbolic description map set can be inserted directly into your application program, as described in "Assembling and cataloging BMS maps" on page 146.

Processing data structures under BMS

An application program can read or modify the attributes or initial data of any named field in an application data structure. The form of an input map data structure differs from that of an output map structure.

When designing a map, you assign names to fields that contain variable data. The symbolic map data structure contains extended versions of these fields, each one consisting of subfields. Each subfield can be referred to by its name, which is the name assigned in the symbolic map definition, plus a single letter suffix. Each kind of subfield has a different suffix.

Furthermore, the whole input data structure, 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'. For assembler maps only, the start and end of the data structure are labeled automatically with the map name extended by the suffixes 'S' and 'E' respectively. The suffix 'L' is added to the mapname to produce a label denoting the actual length of the map. You need to determine the length of the DSECT, for example, if you wish to initialize its contents to X'00'.

Input map data structures: The suffixes used to address subfields, and the contents of those subfields, in input maps are:

D (ASM only) The first byte of the first occurrence of a field defined by the OCCURS operand of the DFHMDF macro.

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 input process is inhibited by a light pen, the first byte of the input data subfield will be set to X'FF', the rest of the field being set to nulls (X'00'). The use of the light pen is discussed under "Handling light pen AIDs" on page 155.

L a halfword binary length value. This defines the number of characters that are typed into the data field before it is read by BMS.

N (ASM only) The first byte of the next occurrence of a field defined by the OCCURS operand.

If MODE=INOUT is specified, the 'fieldname.A' subfield will be defined in the input map data structure. (Compiler errors will occur if a MOVE statement modifying an attribute byte is qualified to refer to the output map.)

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. Provided one of these 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 whole map enables you to manipulate the whole data structure. For example, you can write simple commands to copy the whole 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. This will be ignored if the terminal does not support the extended data stream.

D (ASM only) The first byte of the first occurrence of a field defined by the OCCURS operand of the DFHMDF macro.

H an attribute byte defining the highlighting to be used within a field in a display. This will be ignored if the terminal does not support the extended data stream.

M an attribute byte defining that SO/SI creation is to be used.

N (ASM only) The first byte of the next occurrence of a field defined by the OCCURS operand.

O output data to be sent to the display. The program will usually store data in such a field before sending the map. If the contents of the field begin with a null (X'00') character, the whole field will be ignored, the contents of the display field being 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 will override the contents of the physical map.

P an attribute byte defining the programmed symbol set to be used within a field in a display. This will be ignored if the terminal does not support the extended data stream.

If you want to use programmed symbols, you must ensure that a suitable symbol set has been sent to the device. The *CICS/OS/VS IBM 3270 Data Stream Device Guide* describes how to do this.

T an attribute byte defining that background transparency is to be used.

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. This will be ignored if the terminal does not support the extended data stream.

Subfields with suffixes H, P, V, C, U, M, and T are only generated if the corresponding attribute types are included in the DSATTS operand of the DFHMDI or DFHMSD macros. If EXTATT=YES is specified, subfields with suffixes H, P, V, and C are generated. If KEXTATT=YES is specified in addition to EXTATT=YES, subfields with suffixes U and M are also generated.

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, say 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 will occupy subfield BALANCEO.

You can also manipulate the whole output data structure using its suffixed name. For example, you could copy data into it from another area. More importantly, you can write commands to set the whole 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. The following code shows how you might do this in assembler language, COBOL, and PL/I respectively.

```
ASM     XC MAPO(MAPE-MAPO),MAPO[1]

COBOL   MOVE LOW-VALUES TO MAPO

PL/I    DCL STR BASED CHAR(32767);
                  .
                  .
                  .
        SUBSTR(ADDR(MAPO)->STR,1,
        STG(MAPO))=LOW(STG(MAPO))
```

[1] Data structure must be less than or equal to 256 bytes. You must use another method if the structure is larger than this.

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 (a copy book supplied by IBM and stored in the system source library at installation) into your program, using the name DFHBMSCA. The constants and their meanings are shown under "BMS related constants" on page 205.

Using attribute constants and subfield suffixing, a program can modify field attributes using simple commands. The following examples illustrate 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.

```
ASM    MVC ACCOUNTO,CUSTNO........ (1)
       MVC ACCOUNTC,DFHBLUE....... (2)
       MVC ACCOUNTH,DFHBLINK...... (3)

COBOL  MOVE CUSTNO TO ACCOUNTO.... (1)
       MOVE DFHBLUE TO ACCOUNTC... (2)
       MOVE DFHBLINK TO ACCOUNTH.. (3)

PL/I   ACCOUNTO=CUSTNO;........... (1)
       ACCOUNTC=DFHBLUE;.......... (2)
       ACCOUNTH=DFHBLINK;......... (3)
```

Additional installation defined named attribute constants can be created and cataloged in DFHBMSCA in the source library.

The value of an attribute constant can be determined by referring to the publication *An Introduction to the IBM 3270 Information Display System.*

Invalid data: BMS does not check the validity of attribute and data values in the symbolic data structure. Invalid data may be transmitted to the terminal. Some terminals will detect this invalid data and send error information to CICS. This error information is handled by CICS code and usually results in an abnormal termination of the transaction with an ATNI abend code. This abend can be intercepted by user-written terminal or node error programs (TEPs or NEPs) as described in the *CICS/MVS Customization Guide.*

Sending data to a display

You use the SEND MAP command to send mapped data to a display. You can send three kinds of data, depending on what options you specify, as follows:

- **constant display data** (with attributes) such as headings, footings, prompt fields, and comments
- **variable display data** (with attributes) such as user data or warning messages
- **device control data** such as instructions to clear the screen, or sound an alarm, before displaying data.

The syntax of the command is:

```
SEND MAP(name)
[MAPSET(name)]
[FROM(data-area) LENGTH(data-value)|
  DATAONLY|MAPONLY]

[devcntrl...]
```

The MAP option names the map that is used to format the data, and the MAPSET option names the map set to which the map belongs. If the map set has the same name as the map, you do not need to specify MAPSET.

In its simplest form, the SEND MAP command is used as follows:

1. The application program assigns values to variables named in the symbolic description map.
2. The program issues a 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 a map set called DISPLAY contains an output map of the same name, the map can be displayed using the command:

```
SEND MAP('DISPLAY')
```

Another map, called ERROR, in the same map set can be displayed by:

```
SEND MAP('ERROR') MAPSET('DISPLAY')
```

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 a SEND MAP command.

Composite displays: If your program sends a succession of maps to a display, the final form of the display depends on both the design of the maps, and the form of the SEND MAP command. For example, if the final map fills the screen, or the SEND MAP command includes the ERASE option (see "Device control options" on page 152) 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 144), you can combine them to produce the final display.

Refreshing and modifying displays: You use the **MAPONLY** option of the 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, as 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 that has already been created by a SEND MAP command. BMS transmits variable data but no physical map data.

You cannot issue a SEND MAP DATAONLY command if the screen is unformatted (that is, if there has been no preceding SEND MAP).

No data is sent for fields that you have cleared to nulls (X'00'). You can use a SEND MAP DATAONLY to ensure that only changed fields are sent.

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 GETMAIN commands. You can usually avoid having to code 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 for each language, and how it affects application programs.

However you acquire storage, you must clear its contents (to X'00') before issuing a SEND MAP command. If you do not do this, existing data in storage can modify the output display unpredictably. If you use GETMAIN to acquire storage, you can clear the storage by coding the INITIMG option.

Here are the rules for assembler language (ASM), COBOL, and PL/I:

ASM

STORAGE=AUTO If you code this operand, each map will need its own storage. To acquire the storage automatically (without coding GETMAIN commands), code your COPY statements for map sets immediately after the DFHEISTG statement. If you do this, the DFHEIENT macro acquires storage for you. This is the best way of getting the storage. It is demonstrated in the "Low balance report sample program (ASM)" on page 375.

If you do not code the COPY statements after DFHEISTG, you must code GETMAIN commands before you use maps.

BASE=name You cannot code this for assembler language maps.

nothing specified the assembler generates ORG statements that cause maps in the set to overlay each other. Code your COPY commands for map sets immediately after the DFHEISTG statement (as for STORAGE=AUTO). CICS acquires storage automatically, but only enough to hold the largest map in the set.

If you do not code the COPY statements immediately after DFHEISTG, you must perform your own GETMAIN, specifying the size of the largest map in the set.

Note: There is more information on using the BASE operand under "Map definition macro operand summary" on page 195.

COBOL

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 a GETMAIN command.

BASE=name the map set must be copied into the linkage section. You must code a 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 do not have to code a 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 a GETMAIN command to get storage for it.

Note: When you use GETMAIN to get main storage for a COBOL map, you must ensure that you establish addressability for the map. For more information, see "Base locator for linkage (BLL)" on page 21 and the coding example in the description of the BASE option under "Map definition macro operand summary" on page 195.

PL/I

STORAGE=AUTO CICS acquires storage automatically for every map; you do not have to code a GETMAIN command.

BASE=name you must code a GETMAIN command that gets at least enough main storage to contain the largest symbolic map in the map sets sharing this database.

nothing specified you must code a GETMAIN command that sets the pointer BMSMAPBR to the address of the acquired data area. The GETMAIN must get at least enough storage to contain the largest map in the sets.

Alternative data structures: The examples so far have shown SEND MAP commands that contain literal map names. If the map name referenced by your program is to be a variable, you need to code additional options, FROM and LENGTH, on the SEND MAP command.

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. It need not be coded unless the data to be mapped occupies less than the whole data area.

| **Note:** LENGTH must be specified if the data to be mapped
| is less than the data area receiving the map.

Device control options: As well as transmitting application data to a display, BMS can relay device control commands. An application program uses options of the SEND command to specify which controls are to be activated. Alternatively, it can use the BMS SEND CONTROL command, which transmits device control commands without also sending application data. In the syntax display for SEND MAP, these options are indicated by 'devcntrl...'. For example, the command

```
SEND MAP('ERROR') MAPSET('DISPLAY') ERASE
```

erases the screen before data is displayed.

You can code one or more of the following device control options in a 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 153.

ERASE erase screen and place cursor in top left-hand corner of screen before output.

The first 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 operand of the DFHPCT TYPE=ENTRY system macro.

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).

Sending device controls without display data

You use the BMS SEND CONTROL command to transmit device control orders without also sending data.

The BMS SEND CONTROL command allows you to send any of the device orders listed in "Device control options." The command has the following syntax:

```
SEND CONTROL
[ERASEAUP|ERASE]
[ALARM]
[FREEKB]
[FRSET]
[CURSOR[(data-value)]]
[PRINT]
[FORMFEED]
```

Cursor positioning

You can control the positioning of the display cursor in two different ways, as described below.

Normal cursor positioning: You can specify a two-byte cursor position on the BMS SEND commands. This enables you to specify the absolute value of the cursor position on the screen after the SEND has been performed. The first location on the display screen is address 0.

You specify the address in parentheses after the CURSOR keyword, as follows:

```
CURSOR(44)
```

If you omit the CURSOR option, BMS will search 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 0.

If you omit the CURSOR option from the SEND CONTROL command, the 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 SEND MAP command. To position the cursor symbolically you must mark a field in the symbolic map data structure with a special symbol. If CICS finds this symbol in a data field, it places the cursor under the first data byte in the field on the output screen.

To use symbolic cursor positioning, you must:

1. Specify MODE=INOUT in the DFHMSD macro.
2. Set the length of the of the field (to which the cursor is to be positioned) to −1.
3. Execute the SEND command, specifying CURSOR without an argument.

Accessing data outside the program

Sometimes your program needs access to information held by CICS. The ASSIGN command allows it such access.

Some ASSIGN options apply exclusively to BMS; there is a full list of these under "BMS related ASSIGN options" on page 212. However, the only ASSIGN options you can use under minimum BMS are those concerned with null maps. (Null maps have already been described under "Maps without fields" on page 145.) The ASSIGN options you can use are:

MAPLINE requests the number of the line, on a display, that contains the map's origin.

MAPCOLUMN requests the number of the column, on a display, that contains the map's origin.

MAPWIDTH returns the width of the map.

MAPHEIGHT returns the height of the map.

For example, you can write a mixed GDDM/BMS application program using a BMS SEND MAP statement to position the 'graphic hole'; using a CICS ASSIGN command to determine the size and position of this graphic hole; and using the returned size and position in a GDDM create graphic field (GSFLD) call.

Receiving data from a display

You use the RECEIVE MAP command to receive data from a display. The data from the display is mapped into a data area in an application program.

The syntax of the command is:

```
RECEIVE MAP(name)
[MAPSET(name)]
[INTO(data-area)|SET(ptr-ref)]
[FROM(data-area) LENGTH(data-value)|
  TERMINAL|[ASIS]]
```

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 to which the map belongs. If the map set has the same name as the map, you do not need to specify MAPSET.

For example, in its simplest form, the RECEIVE MAP command is coded as:

```
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:

```
RECEIVE MAP('MENU') MAPSET('DISPLAY')
```

This command tells BMS to map the input data into a symbolic map data structure called MENU.

After a RECEIVE MAP command, your program can determine the inbound cursor position by inspecting the halfword binary value stored in EIBCPOSN. Similarly, it can determine the type of attention identifier (AID) by inspecting EIBAID.

You cannot issue RECEIVE MAP or RECEIVE PARTN commands in a nonterminal task, because these tasks do not have a TIOA or a TCTTE. The INVREQ condition will be raised if you attempt this.

Receiving data into an alternative data structure: The sample 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, in which case you must use one of the options INTO or SET.

If you code INTO, display data will be mapped into the named data area rather than into the data structure for the symbolic description.

If you code SET, 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. This data area includes the 12-byte TIOA prefix.

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 151.

BMS sets the receiving area to nulls (X'00') before performing the RECEIVE. So you should save any data in this area before performing a RECEIVE. Furthermore, if you depend on BMS to set a data area to nulls for you during a RECEIVE, you should be aware of the MAPFAIL condition. If this arises, 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 upon what you specify with the JUSTIFY operand of the DFHMDF macro. For more information on JUSTIFY, see page 199.

The MAPFAIL condition can arise unexpectedly after a RECEIVE MAP command. For example, it arises if you press a PA or PF key when CICS is waiting to perform a RECEIVE command. Therefore, always consider coding a 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 (FEATURE=UCTRAN). You can override this for any individual RECEIVE command by specifying ASIS. However, ASIS has no effect for a RECEIVE MAP command if that command maps the data that initiated the transaction.

Mapping data from another data area: Sometimes, however, you need to perform an input mapping operation in two stages; accepting and storing the input data in one stage, mapping it in the second. For example, your program might receive (but not map) data using a terminal control RECEIVE command. It would then have to map the data from CICS storage.

You use the FROM and LENGTH options of the RECEIVE MAP command to specify that data is to be mapped from a data area instead of from a terminal. FROM names the data area; LENGTH indicates the number of bytes of data to be mapped. If the data is produced by a terminal control RECEIVE command or by a BMS RECEIVE PARTN command, the LENGTH value of the RECEIVE MAP command must match that specified in the original RECEIVE command.

The terminal control RECEIVE command is described in "Chapter 3.3. Terminal control" on page 221. The RECEIVE PARTN command is described on page 168.

Note: The data obtained from a RECEIVE BUFFER command cannot be mapped because the data will not contain set buffer address (SBA) orders, and a MAPFAIL condition will be raised.

Responding to terminal input

As we have seen at the beginning of this chapter, certain operator actions cause an AID to be transmitted to CICS. Each such action generates a different AID. The AID is a one-byte character, and can be tested by an application program. This can be used as a mechanism for controlling program flow. The HANDLE AID command controls conditional branching caused by AIDs.

HANDLE AID command

```
HANDLE AID
option[(label)]...
```

You use the 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 a HANDLE AID for an AID, control returns to the application program at the point immediately following the input request.

You can suspend the HANDLE AID command by means of the PUSH and POP commands as described in "Chapter 1.5. Exceptional conditions" on page 43.

A HANDLE AID command takes precedence over a HANDLE CONDITION command. If an AID is received during an input operation, for which a HANDLE AID is active, control passes to the label specified in that request regardless of any exceptional conditions, for example MAPFAIL, that occur (provided they do not stop receipt of the AID).

The HANDLE AID options that can be specified under minimum BMS are:

ANYKEY any PA key, any PF key, or the CLEAR key, but not ENTER

CLEAR for the key of that name

CLRPARTN for the key of that name

ENTER for the key of that name

LIGHTPEN for a light pen attention

OPERID for the operator identification card reader, the magnetic slot reader (MSR), or the extended MSR

PA1, PA2, or PA3 any of the program access keys

PF1 through PF24 any of the program function keys

TRIGGER which means that the display cursor has left a field described as a trigger field which has been modified by the terminal operator.

A HANDLE AID command for a specified AID remains active until the task is terminated or until another HANDLE AID is issued for that AID. (If no label is specified in the new request, the existing HANDLE AID command is suspended.)

A HANDLE AID command is valid only for the program in which it is issued. 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 HANDLE AID commands that were active in the higher-level program before control was transferred from it are reactivated, and any HANDLE AID commands activated in the lower-level program are deactivated.

If CICS detects an OPERID AID, the code that handles the AID can inspect the EXEC interface block (the EIB) to find out which of the magnetic stripe readers (MSR OR MSRE) has been used. MSR generates an AID of X'E6', MSRE generates an AID of X'E7'.

If an AID covered by the general option ANYKEY is received and there is no active HANDLE AID command for the specified AID but there is an active HANDLE AID ANYKEY command, control will pass to the label specified in this command. A HANDLE AID command for an AID overrides the HANDLE AID ANYKEY command as far as that AID is concerned.

The following example shows a 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
```

You cannot code more than 16 options in a single HANDLE AID command.

Rather than using HANDLE AID, a program can examine the value of the EIBAID field in the EIB to find out which attention key has been pressed. The 3270 terminal transmits an AID character, which is stored in field EIBAID. The program can compare the contents of EIBAID with the constants supplied in the CICS copy book DFHAID. The contents of DFHAID relevant to minimum BMS are shown below. For the full list of contents, see "Attention identifier constants (DFHAID)" on page 208.

Constant	Meaning
DFHENTER	ENTER key
DFHCLEAR	CLEAR key
DFHPA1-DFHPA3	PA1-PA3 keys
DFHPF1-DFHPF24	PF1-PF24 keys
DFHOPID	OPERID or MSR
DFHMSRE	Extended(standard) MSR
DFHTRIG	Trigger field
DFHPEN	SELECTOR PEN or CURSOR SELECT key

Handling light pen AIDs: There are two different kinds of light pen detectable fields: immediate and deferred. You indicate an immediate field by storing a blank (X'40') in its first byte. You show that it is a deferred field by storing a question mark (?) in its first byte. In either case, you must ensure that the attribute byte for the field indicates that it is light pen detectable.

When an operator selects an immediate field, the display device transmits an AID plus the contents of the field. It sets the first byte of the returned field to X'FF' to indicate that the field is a light pen field.

When an operator selects a deferred light pen detectable field, the display device changes the field's question mark to a 'greater than' (>) sign. This indicates that the field has been selected. No data is transmitted at this point. By reselecting the > sign, the operator can cancel the selection, resetting the question mark. Data is eventually transmitted when another operation, for example pressing the ENTER key, or selecting an immediate detectable field, generates an AID. Before the data field is returned to BMS, its first byte is set to X'FF' to indicate that the field is a light pen field.

Deferred fields are useful when an operator has to select a series of items in a display. Deferring the input operation until all the selections have been made improves efficiency.

The CLEAR, PA1, PA2, and PA3 keys do not transmit data to CICS. They normally signal a special operator request that does not require data, such as print, page forward, page backward, or exit from a repeating transaction. In practice, the PF keys are also often used for this purpose.

Exceptional conditions

CICS exceptional conditions have already been introduced under "Chapter 1.5. Exceptional conditions" on page 43. BMS commands in your program can raise a number of exceptional conditions. These, and the default system action they invoke, are listed under "BMS exceptional conditions" on page 217.

You are only likely to encounter two exceptional conditions when using minimum BMS, as follows:

MAPFAIL occurs, on input only, if the data to be mapped has a length of zero or does not contain a set buffer address (SBA) order. This is what happens if you press a PA key, the CLEAR key, or either ENTER or a PF key without data.

Default action: terminate the task abnormally.

ERROR See under "ERROR exceptional condition" on page 44 for an explanation of the ERROR condition.

An exceptional condition is not necessarily an error condition. Sometimes you might even wish to treat an exceptional condition as part of the normal course of events.

You use the HANDLE CONDITION command to respond to exceptional conditions.

Printed output

Very often you will find that you want printed output (hard copy) as well as, or instead of, the screen images produced by a transaction. You have a choice of methods of producing such output. Which one you choose depends on your requirements. This section describes the methods available; briefly, they are:

- Hardware print key feature
- CICS local copy key
- The ISSUE PRINT command
- Asynchronous page build transaction.

Hardware print key feature: Some display terminals models have a hardware print key. Pressing one of these keys 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.

CICS local copy key: In the absence of a hardware print key, you may want your program's end-user operators to be able to print copies of the screen contents using a PA (program access) key. If that is so, you must ensure that your CICS system has been designed to allow it. The local copy key causes CICS to print the contents of a screen on the first eligible and available 3270 printer.

To define a terminal to use a local copy key, see the *CICS/MVS Resource Definition (Macro)* manual or the *CICS/MVS Resource Definition (Online)* manual.

For more information, see the *CICS/OS/VS IBM 3270 Data Stream Device Guide*.

ISSUE PRINT command: The format of the command is:

```
ISSUE PRINT
```

The ISSUE PRINT command prints the contents of a screen on the first eligible and available 3270 printer.

For more information on defining print keys, see the *CICS/OS/VS IBM 3270 Data Stream Device Guide*.

Asynchronous page build transaction: The methods of printing described so far have the advantage of being simple to implement. However, they do not provide the flexibility often needed in commercial applications. In particular, they do not allow your program to combine mapped output data to produce an entire printed page. There are two ways of achieving this under minimum BMS. Both apply only to non-SCS 3270 printers. For additional printer support you must use standard function BMS (see "Printer support" on page 160). Before looking at the two methods, let us first consider how a program uses a 3270 printer.

A 3270 printer contains a page buffer. BMS moves data into this page buffer when instructed to do so by SEND MAP or SEND CONTROL commands. The page buffer is printed only when BMS receives a SEND MAP or SEND CONTROL command containing the PRINT option. Likewise, it is erased only if BMS receives a SEND MAP or SEND CONTROL command that specifies the ERASE option. These properties of the printer make it possible for a program to build a single page of printed output from a series of maps.

Two ways of printing a page built from multiple maps are:

1. Using the interval control START command

   You use the START command (see "Start a task (START)" on page 277) to initiate a secondary CICS task. This will be 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 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 SEND 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 WRITEQ command. CICS can be made to initiate a print transaction when a certain 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 at which it is started, are defined in the destination control table (DCT).

   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. Alternatively, each instance of your transaction can obtain exclusive control of the transient data queue by ENQ and DEQ commands.

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: BMS builds device dependent data streams for 3270 printers by computing 'set buffer address' (SBA) orders based on the page width specified by the PGESIZE or ALTPGE options of the DFHTCT TYPE=TERMINAL system macro, or by the PAGESIZE or ALTPAGE attributes of the TYPETERM definition.

The 3270 printer, however, prints the 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, unless it is overridden by the L40, L64, or L80 options of the SEND MAP or SEND CONTROL command that specifies the PRINT option.

Unexpected results will occur if the TCT page width is not 40, 64, or 80 columns, or is not the printer platen width. Unexpected results will also occur if the WCC page width is set by the application program to be different from the TCT page width. Normally, your program should not specify one of these options in a BMS SEND command. For this reason, the command syntax panels in this chapter do not show the options, even though you can code them.

Form feed characters: You can code an option, called FORMFEED, on the SEND MAP and SEND CONTROL commands. This generates a form feed (X'0C') character at the start of the data stream. If you code this option for a terminal that does not support form feed, CICS simply ignores the request. To indicate that a terminal supports form feed, you must code "FF=YES" on its DFHTCT TYPE=TERMINAL system macro, or use the FORMFEED attribute of the TYPETERM definition.

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, in which case form feed will not occur.

The FORMFEED option on 3270 displays is particularly useful if the screen is to be printed using the hardware local copy key or the CICS PA1 print key facility. Its use ensures that the screen image is printed on a fresh page.

Be careful when using the FORMFEED option on a SEND CONTROL command. The SEND CONTROL command will always generate a complete blank page. Thus a SEND CONTROL FORMFEED will skip to a new page and also send this as a blank page. However, as described earlier, 3270 printers sometimes suppress null lines so that a blank page will be printed as a single line.

Chapter 3.2-3. Standard function BMS

This chapter describes the features provided by standard function BMS that are additional to those provided by minimum function BMS as described in "Chapter 3.2-2. Minimum function BMS" on page 139. For convenience, 'standard function BMS' will be shortened to 'standard BMS'.

Standard BMS supports all terminals and printers in the 3270 family of devices (for example 3180, 3270-PC, 3279, 3286, 3290, LUTYPE2, LUTYPE3, and SCSPRT). It also supports the terminal types and associated terminal features listed below. This list does not include all terminals supported by CICS; some devices can only be used through the terminal control interface.

```
CRLP or TRMTYPE=TCAM terminals
Magnetic Tape
Sequential Disk
TWX Model 33/35
1050
2740-1,-2 (no buffer receive)
2741
2740-2 (buffer receive)
2770
2780
3780
3767/70 Interactive LU
2980 Models 1 and 2
2980 Model 4
3600 (3601) LU
3650 Host Convers (3653) LU
3650 Interpreter LU
3650 Host Convers (3270) LU
3770 Batch LU
LUTYPE4
```

Standard BMS provides the following function in addition to that provided by minimum BMS:

- Text processing. The SEND TEXT command allows your application program to send data to a terminal. BMS splits the text into lines that fit the target terminal, ensuring that words do not split across line boundaries.
- Printer support. The NLEOM option of the SEND MAP and SEND TEXT commands tells BMS to build a device dependent data stream for 3270 printers. Such a data stream allows greater flexibility in page width settings for 3270 printers, and avoids the suppression of null lines in printed output.
- Partition support. Some terminals allow your application program to divide the display area into a number of independent "logical terminals" or partitions that it can address individually.
- Logical device components. Some terminals (for example, the 3601, LUTYPE4, and 3790) have more than one component, such as a printer and a console. BMS can treat each of these components as a separate terminal.
- 10/63 Magnetic slot reader control. Support for application program control of the 10/63 magnetic slot reader attached to an 8775 or a 3643 terminal.
- Trigger fields. Support for the TRIGGER validation attribute. This allows a program to start processing input data without the terminal operator having to press the ENTER key. This can make data entry more efficient.
- Outboard formatting. This is a process that helps to reduce line traffic.
- Block data format. (See "Block data format" on page 172.) This is an alternative to field data format, but you are recommended not to use it.

Text processing

Use the SEND TEXT command to send text to a terminal. Its syntax under standard BMS is:

```
SEND TEXT
FROM(data-area)
LENGTH(data-value)
[CURSOR(data-value)]
[FORMFEED]
[ERASE]
[PRINT]
[FREEKB]
[ALARM]
[NLEOM]
[LDC(name)|[ACTPARTN(name)]
[OUTPARTN(name)]]
[MSR(data-value)]

Conditions: INVLDC, INVPARTN,
INVREQ, RETPAGE, WRBRK
```

Use FROM to specify the data area containing the text to be sent, and LENGTH to specify the length of this area.

When formatting the text, BMS flows it from one line to the next and left-justifies it (just like the text you are reading here). BMS splits the text into a series of lines, starting each line with a single character. On a 3270 display screen, this is the attribute byte; it is unprotected, normal intensity, so it looks like a blank character on the screen. On a non-display device, this *is* a single blank character. (So your maximum line length will always be one character less than the physical terminal page width.[2]) BMS also pads the ends of lines with blanks to avoid splitting words.

If the FROM data area contains more text than will fit on a single page, BMS creates more than one page. These overwrite each other on a display terminal, unless you code the PAGING option (which is available only under full BMS).

New line characters (X'15') and character attribute controls embedded in the text are honored. So are blanks (X'40'), with one exception:

If a line of text ends with a non-blank character, and the next character is a blank, BMS discards the blank before starting the new line. It puts the next character of the text into the first text column (that is, column two) of the new line. This is the one case where a blank is not honored. (If a line of text ends with a blank and the next character is also a blank, BMS honors the blanks, processing the data without discarding what would otherwise be one or more leading blanks on the new line.)

The other options of the SEND TEXT command have the same effect as the corresponding options of the SEND MAP command, as described on page 209.

Character attribute control

When data is destined for a device that supports the 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. 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 SEND TEXT command.

As described in "Attribute constants" on page 150, copy book DFHBMSCA contains a selection of predefined constants that you can use in your programs. Here is a simple PL/I statement that will color a single word blue:

```
TEXTSTR='data '||DFHSA||DFHCOLOR||
        DFHBLUE||'blueword '||
        DFHSA||DFHCOLOR||
        DFHDFCOL||'rest of data';

SEND TEXT FROM(TEXTSTR) LENGTH(100);
```

Printer support

This section discusses the differences between 3270 printers, 3270 printers using the NLEOM option, and other printers. It also contains guidelines on writing application programs that are independent of printer type.

3270 printers without the NLEOM option

IBM 3270 printers (whose DFHTCT TYPE=TERMINAL macro does not specify TRMTYPE=SCSPRT, or whose DEVICE attribute on their TYPETERM definition is not SYSPRINT) differ from other types of printer, because they contain a page buffer. Data is moved into this page buffer by SEND MAP, SEND TEXT, and SEND CONTROL commands. The page buffer is printed only if those commands specify the PRINT option. The page buffer is erased only if those commands specify the ERASE option. This has the following consequences:

1. A printer page can be composed of several small maps. The first SEND MAP specifies the ERASE option, and the last SEND MAP specifies the PRINT option.
2. If successive printer pages are required with a single map on each page, each SEND MAP command must specify both the ERASE and PRINT options.

Other printers do not have page buffers. Instead, for such printers, each SEND MAP, SEND TEXT, or SEND CONTROL command prints a complete page. A single page cannot be composed of several small maps, unless the ACCUM option (full BMS only) is used.

For 3270 printers, null lines are suppressed; that is, they are not printed. Every line in a map for a 3270 printer should contain a field; a blank field will suffice.

If you are using 3270 printers without the NLEOM option, ensure that your TCT terminal entry PGESIZE and ALTPGE settings, or the PAGESIZE or ALTPAGE attributes of the TYPETERM definition, are 40, 64, 80, or the printer platen width, otherwise unpredictable results will occur. This is discussed further in "Setting the printer page width" on page 157.

[2] Specify the terminal page width by the PGESIZE or ALTPGE operands of the DFHTCT TYPE=TERMINAL system macro, or by the PAGESIZE or ALTPAGE attributes of the TYPETERM definition.

3270 printers with the NLEOM option

If you omit the NLEOM option from SEND MAP or SEND TEXT commands, BMS builds a device dependent data stream that includes SBA (set buffer address) orders. These orders position fields on the printer page.

If you include the NLEOM option on SEND MAP or SEND TEXT commands, BMS builds a device dependent data stream using new line and blank characters to position fields on the printer page. The data stream is terminated by an EOM (end of message) order, which stops printing.

If you use the NLEOM option, note the following:

1. The printer buffer is printed only if the PRINT option is specified in a SEND MAP, SEND TEXT, or SEND CONTROL command.
2. The printer buffer is cleared only if the ERASE option is specified in a SEND MAP, SEND TEXT, or SEND CONTROL command. Successive BMS output commands specifying NLEOM, but not ERASE, build successive pieces of that data stream in the printer page buffer. However, as the first piece of data stream is terminated by an EOM order, this is the only data that can be printed.

   A printer page cannot be composed of several small maps if NLEOM is used, unless ACCUM is also specified. The ACCUM option is supported only by full BMS.
3. The printer buffer can never contain null lines, even if the map contains blank lines. Blank lines in the map are represented by new line characters in the printer page buffer.
4. Use of NLEOM may allow larger pages to be printed. This is because NLEOM uses the buffer more efficiently. It eliminates spacing with null characters in the print buffer.
5. Use of NLEOM allows you to specify a page width other than 40, 64, 80, or the printer platen width in your TCT.

SCS and other non-3270 printers

A 3270 printer whose DFHTCT TYPE=TERMINAL macro specifies TRMTYPE=SCSPRT, or whose TYPETERM definition specifies DEVICE(SCSPRINT), or a non-3270 printer, does not have a page buffer. As a result the NLEOM, PRINT, and ERASE options on SEND MAP, SEND TEXT, and SEND CONTROL commands are ignored. Each BMS output command causes a complete page to be printed.

BMS generates a data stream for these terminals using blanks and new line characters to position data. It also uses horizontal and vertical tab characters to position data in the following circumstances:

1. The DFHTCT TYPE=TERMINAL macro specifies either HF=YES or VF=YES, or both. The equivalent attributes for the TYPETERM definition are HORIZFORM and VERTICALFORM respectively.
2. The DFHMSD map set definition macro defines a tab map using the HTAB and VTAB operands.

The use of tab characters may result in a shorter data stream.

The horizontal and vertical tabs generated by BMS assume that the application program has previously set up the tabs on the printer using a terminal control SEND command. The *CICS/OS/VS IBM 3270 Data Stream Device Guide* describes how to do this.

FORMFEED option

The FORMFEED option has been described for minimum BMS. However, 3270 printers and non-3270 printers respond differently to the option on the SEND CONTROL command. Use the FORMFEED option on SEND CONTROL with care.

A SEND CONTROL FORMFEED command directed to an SCS printer with form feed support will start a new page, and then print a blank page (all BMS output commands directed to an SCS printer transmit an entire page). A SEND CONTROL FORMFEED PRINT command directed to a 3270 printer with form feed support will similarly start a new page, and then transmit an entire page of null lines (if NLEOM is not specified). However, the 3270 printer will suppress these null lines, replacing them all with a single new line.

Printers and BMS text

Note the following if your application program uses the BMS SEND TEXT command to communicate with a printer.

Text output to SCS printers and, if NLEOM is specified, to other 3270 printers can be formatted in two ways, depending on the specification of PRINTERCOMP on the profile definition if you are using resource definition online (RDO) or PTRCOMP/NPTRCOMP if you are using the PCT macro.

PRINTERCOMP(NO), which is the default, produces printed output consistent with the format that would be produced on a 3270 display, that is, the first character of each line, and of each separate SEND TEXT request if this continues a line, is a blank, corresponding to the 3270 attribute byte on the display.

BMS starts each line of output with a single blank. The maximum available line width is one character less than the printer page width.

If you use cumulative text processing, discussed in "Cumulative text formatting" on page 181, bear in mind that BMS precedes each block of text (that is, the text specified in a single SEND TEXT command) by a single blank attribute byte. Do not split single words across multiple text blocks.

PRINTERCOMP(YES) allows use of the full width of the page for printed data, and suppresses the initial blank on each line.

PRINTERCOMP(NO) is recommended for new applications, unless the full width of the page is required. This option ensures consistent results if the application is run on different printer types.

PRINTERCOMP(YES) should be used if the full width of the page is required and the application will always run on SCS printers or the SEND TEXT requests include the NLEOM option. PRINTERCOMP(YES) provides printed output compatible with that produced in CICS Version 1.5 and before.

In both cases, new line characters (X'15') embedded in the text data are always honored. If, for example, you want to print lines of 120 characters, you must embed new line characters in your text data and ensure that your TCT page width setting is at least 121 characters (subject to the description on printer line widths in "3270 printers without the NLEOM option" on page 160).

Printers and device independence

Use the following guidelines if your application program is to function, under standard BMS, on both 3270 and SCS printers:

1. Each printer page should contain a single map or block of text. Both the ERASE and PRINT options should be specified on the SEND MAP and SEND TEXT commands. Alternatively, the ACCUM option should be used; this requires full BMS.
2. If the NLEOM option is not used for 3270 printers, a field (perhaps a blank field) should be defined on each line of the map.
3. The 3270 printer page width should be 40, 64, or 80, or the printer platen width, or the NLEOM option should be used.

Partition support

Application programs can use simple BMS commands to manage a partitioned display.

An IBM device that supports partitions can divide its screen to produce up to eight working areas, called partitions. When you write a CICS transaction for a partitioned display, you can treat each partition as a different display. You can send data from different programs, or different program steps, in the transaction to different partitions (though different transactions cannot communicate with different partitions at the same time). This enables you to design very complex interactive transactions without making operator procedures too complicated to be practical.

The design of the terminal makes it easy for operators to use a properly designed transaction using partitions. Only one partition can be "active" at a time. This active partition contains the cursor.

The normal display controls (such as the cursor control key, the enter key, and cursor wraparound) apply to the active partition only, not to the whole display. For example, pressing ENTER after typing data into an input display menu transmits data from the active partition only. An operator can concentrate on the contents of a single partition, using the terminal keyboard to communicate with that partition alone. A special key, the PARTITION JUMP key, allows you to change the active partition at will.

The CLEAR key erases the entire screen, as for any other display device. However, for keyboards that contain a CLEAR PARTITION key, the contents of the active partition can be erased without affecting other partitions.

A CICS program can determine from which partition the data it receives has been sent. It can also control which partition is active.

When you define a partition, you allocate it an area of the display screen, called a 'viewport', and a portion of the device's buffer storage, called its 'presentation space'. If the presentation space is larger than the viewport, the partition can be scrolled (vertically, but not horizontally).

Screen partitioning is a relatively new concept, and you are unlikely to have experience of using it. The following examples may help you to appreciate the range of possible applications.

Applications of partitions under CICS

You can modify tables and maps to execute existing transactions from a partitioned screen. However, the full benefit of partitioning can best be obtained by designing new transactions.

How existing programs can use partitions

Sending a different map to each partition: You can execute existing CICS BMS applications on partitioned screens simply by modifying maps. Each map can be sent to a different partition.

Cursor movement on a partitioned screen is restricted to the viewport of the active partition (for example, the tab key does not move the cursor out of the viewport). You can use this fact to improve the usability of displays built from several maps.

Defining an error partition: Terminal operators can find transactions easier to use if you define an error message partition. CICS sends error messages to an error partition whenever possible, thereby keeping the contents of the transaction's display intact.

How new application programs can use partitions

Overlapping operator keystrokes: One source of operator frustration is the delay sometimes experienced when performing a repetitive data entry task. After filling a screen with data and pressing ENTER, the operator has to wait for the application program to respond before further typing.

A data entry transaction using a terminal with two partitions can display two copies of the same data entry panel. After filling the first panel, the operator presses ENTER to transmit the data, then presses the PARTITION JUMP key. While CICS is processing the input from the first, the operator can type data into the second partition. Further input to the first partition is inhibited until the application program responds to it. Error messages associated with the first panel are sent to the first partition, and do not affect the data being entered in the second partition. If you define an error partition, the error message need not affect either data entry partition.

Sample programs are provided in COBOL and PL/I which illustrate overlapped keystroking into two BMS partitions. The source code for these programs is in the CICS/MVS library CICS212.SAMPLIB and is named DFH$CPKO(COBOL) and DFH$PPKO(PL1).

Scrolling: Scrolling can remove the need for BMS terminal paging, when handling larger amounts of data than will fit onto a partition's viewport. (Terminal paging is a function of full BMS.) As well as simplifying operator procedures, scrolling reduces line traffic. It can also improve response time, because after data has been transmitted to the terminal, scrolling can be performed without involving the host processor. If the quantity of data to be sent to the screen exceeds the size of the presentation space, BMS paging must also be used.

You can use scrolling even when there is only one partition. This can be especially useful, because you can use all available buffer storage in the terminal to store a single message.

Look aside querying: An operator performing order entry in one partition can activate another partition to make look aside queries. These can include checks on stock levels, prices, or customer credit levels. Without partitions this could only be achieved by complicated programming and a high level of data transmission. Using partitions, a partially completed order need not be transmitted to the host processor before releasing the screen for an inquiry. The order can be entered in one partition, the inquiry in another.

Sample programs are provided in COBOL and PL/I which illustrate overlapped keystroking into one BMS partition while look aside queries can be made using another BMS partition. The source code for these programs is in the CICS/MVS 2.1 library CICS170.SAMPLIB and is named DFH$CPLA(COBOL) and DFH$PPLA(PL1).

Data comparison: An operator can compare two or more sets of data by displaying them simultaneously in different scrollable partitions. To do this without partitions, an application program would probably use BMS page chaining. However, page chaining is only available under full BMS.

The data to be compared could be taken from two different parts of the same document, or from two separate documents.

Tutorial information: One of the partitions on a screen can be reserved for use as a HELP panel. This can be useful as a training aid for new operators, or as memory aid when you are using a transaction that is rarely needed.

How CICS manages partitions

CICS partitions a display according to definitions stored in a **partition set**. You create a partition set by coding and assembling a series of macro instructions. Every different partition configuration requires its own set of macros.

To partition a display, CICS loads a partition set into the internal buffer of the display device. You can use the CEDA transaction (or the DFHPCT system macro) to name a partition set to be loaded for a particular transaction. CICS loads this partition set when the transaction first sends data to the display. If you do not define such a **transaction partition set**, CICS sets a display device to its base (unpartitioned) state before initiating the transaction. The terminal does not have to remain in this state throughout the transaction: the BMS command SEND PARTNSET, allows you to load a partition set, dynamically, from your application program.

You use BMS SEND and RECEIVE commands to communicate with a display that has been partitioned. The commands allow you to name the partition with which you wish to communicate.

Map suffixing and partitions: BMS map set suffixes relate different versions of a map set to different terminal models. This allows you to format the same data differently on different screen types, in response to the same programming request, as shown in Figure 21 on page 147.

Suffixing can also relate versions of maps to particular partitions. Each partition on a display can be treated as a separate screen. The same transaction may be initiated from partitions of various sizes, and thus might need a different version of each map for each partition. When selecting a map set to perform an input or output operation, BMS loads the version with a suffix specified in the definition for that partition. If such a version does not exist, BMS uses the unsuffixed version.

Partitioning concepts

CICS makes it easy to write programs to use partitioned displays. Nevertheless, there are several concepts that you must understand before you can design such programs. This section outlines the concepts, and introduces some new terms.

Partitions: A partition is an addressable subset of the internal resources of a display device. It consists of a fixed part of its screen, and a fixed part of its internal storage. The part of the screen allocated to a partition is called its **viewport**. The internal storage containing data to be displayed is called its **presentation space**. A partition's presentation space contains only display data that CICS sent to that partition. If it is larger than the viewport, only part of the data it contains can be displayed at one time. The part being displayed is called the partition's **window**.

Partition set: Although your programs can address individual partitions, CICS can deal only with partition sets. A partition set is a group of partitions designed to share the same screen. CICS must load the whole partition set onto a terminal before it can communicate with any of the partitions. This does not mean that you have to have more than one partition in a set; you can have a set that contains only one partition. By defining a single partition, you can use all of the available display buffer as the presentation space for that partition. This enables you to scroll large amounts of data.

Application partition set: CICS does not load a partition set into a display's buffer until your application program issues an output request. The partition set it loads becomes the **application partition set**. By default, this is the partition set that is named in the PCT when your transaction is added to the CICS system. Alternatively, it is the partition set named by the most recent SEND PARTNSET command that your program issued.

Output partition: CICS directs output data to the partition named in the OUTPARTN option of the SEND MAP, SEND TEXT, or SEND CONTROL command. If OUTPARTN is not coded on a SEND command, BMS sends the data to the first partition in the partition set. If the display has not been partitioned, or cannot be partitioned, BMS ignores the OUTPARTN option.

Active partition: The **active partition** is the partition that contains the cursor. It can be scrolled vertically. While a partition is active, the cursor 'wraps around' at the viewport boundaries, and the ENTER key (or any input key) transmits data from that partition only. The active partition can be changed either by coding the ACTPARTN operand on a SEND MAP, SEND TEXT, or SEND CONTROL command, or by using the PARTITION JUMP key.

Input partition: Although a program can activate a particular partition, the terminal operator can use the partition jump key to activate a different one instead. If the program logic requires input from a particular partition, you must code the INPARTN option on the RECEIVE MAP command. This option tells BMS from which partition it must receive data. If it receives data from the wrong partition, BMS moves the display cursor into the correct input partition. It does not change the contents of other partitions, except to display a message in the error partition (if there is one).

Although the RECEIVE MAP command can be used to receive data from any partition on a screen, it must specify the correct map for the partition that supplied the data. A new command, RECEIVE PARTN, can be used to read data from an unspecified partition into a data area (and to discover which partition it comes from). This command is fully discussed in "Determining the actual input partition" on page 168. The data can then be mapped, according to the partition it has been read from, using a RECEIVE MAP FROM command. For example, a PL/I application program might contain the following:

```
RECEIVE PARTN(PNAME) INTO(A);
IF PNAME='P1' THEN
        RECEIVE MAP(MAP1) FROM(A);
IF PNAME='P2' THEN
        RECEIVE MAP(MAP2) FROM(A);
```

Unpartitioned or "Base" state: CICS can only create an application partition set on the instructions of either your application program or its PCT entry. If it receives no such instructions, it sets the terminal to **base state** before sending data to it. In this state, the terminal behaves as an ordinary (unpartitionable) display device.

Symbol sets and character size: The IBM 3290 can contain up to six of your own symbol sets, in which you can specify the size of the characters. Any of your symbol sets can be loaded or replaced under program control. The 3290 also contains two standard character sets.

During customer set up (CSU), you must ensure that the 3290 divides the screen in an appropriate fashion, by specifying a character cell size, which becomes the default. You can override this default by coding CHARSZE when you define a partition. Character cell definition is discussed further in "Character cells in partitions" on page 168.

Summary of implementation and use of partitions

The rest of this section describes what you must do if you want to use partitions fully. The following summarizes the steps involved.

1. Ensure that the version of BMS loaded during CICS initialization supports partitions. This is part of the **Installation** task.
2. Define partition sets (Application Programming).
3. Write application programs that use the partition sets (Application Programming).
4. Define BMS maps for partitions (Application Programming).
5. Assemble partition sets and store in the CICS program library. This is part of the **Resource Definition** task.
6. Assemble programs and maps (Application Programming).
7. Create TCT entries for the terminals. This is part of the **Resource Definition** task.

   The IBM 3290 can be configured as more than one logical unit, in which case it should have more than one definition in the TCT.
8. Use CEDA (or DFHPCT and DFHPPT) to define and install resource groups containing entries for related programs, map sets, partition sets, transactions, and profiles. This is part of the **Resource Definition** task.
9. Document new operator procedures for users of the new devices. This is a prerequisite for the **Operation** task.

Application programming

CICS partition support is based on the concept of the partition set. Partition sets are analogous to map sets. They are defined using partition definition macros.

As map set definition allows a programmer to design a program and its maps separately, so partition set definition can be performed apart from application programming. This means that predesigned partition sets can be used to control transmission of data requested using simple CICS commands. The commands you code to communicate with a display that supports partitions are described later. The next section describes how to define partition sets and prepare them for use.

Defining partition sets

Partitions are defined by coding the macros DFHPSD (partition set definition) and DFHPDI (partition definition). Each partition definition must be part of a partition set definition.

Partition set definition macro (DFHPSD): This section describes the partition set definition macro, DFHPSD. Each DFHPSD macro is followed by one or more DFHPDI macros, and is ended with a DFHPSD TYPE=FINAL macro.

The format of the partition set definition macro is:

```
partnset DFHPSD
        [SUFFIX=user-suffix]
        [,ALTSCRN=(lines,columns)]
        [,CHARSZE=(vpels,hpels)]
```

The operands have the following meanings:

partnset is a 1-to 6-character partition set name.

SUFFIX=user-suffix is a 1-character user suffix for this version of the partition set. It allows different versions of a partition set to be associated with different terminals. Partition sets are selected according to the same rules as map sets. For more information, see Figure 21 on page 147.

ALTSCRN(lines,columns) specifies the size, in characters, of the usable area of the target terminal. This is normally the same as the ALTSCRN operand of the DFHTCT TYPE=TERMINAL entry for the terminal. You use ALTSCRN to ensure that the viewports of partitions within a partition set fit into the usable area of the screen.

CHARSZE(vpels,hpels) specifies the size of the character cell, on a display, to be allocated to each character displayed in partitions of the partition set. You specify the size as 'vpels' (the number of vertical picture elements), and as 'hpels' (the number of horizontal picture elements). For guidance on the choice of values, see the description of CHARSZE in "Character cells in partitions" on page 168. The values specified in this operand become the defaults for all partitions in the partition set. You can override this default for individual partitions by coding CHARSZE in the DFHPDI macro.

Partition definition macro (DFHPDI): A partition set contains one or more partitions. Each partition is defined by coding a partition definition macro.

The format of the partition definition macro is:

```
[partn] DFHPDI
         VIEWPOS=(lines,columns)
        ,VIEWSZE=(lines,columns)
       [,BUFSZE=(lines,columns)]
       [,CHARSZE=(vpels,hpels)]
       [,MAPSFX=mapset-suffix]
       [,ATTRB=ERROR]
```

The operands have the following meanings:

partn is a 1 or 2 character partition name. It allows you to refer to the partition in your application programs.

Every partition in a partition set must have a different name. Only the error partition can be unnamed (see ATTRB = ERROR operand).

VIEWPOS = (lines,columns) specifies the position of the top left hand corner of this partition's viewport. You specify the position in numbers of lines and numbers of columns.

The DFHPDI macro checks that viewports do not overlap. If the ALTSCRN operand of the DFHPSD macro has been coded, DFHPDI also checks that all viewports fit within the usable area of the terminal screen.

VIEWSZE = (lines,columns) specifies the size, in lines and columns, of the partition viewport. The DFHPDI macro checks that viewports do not overlap. If you code the ALTSCRN operand of the DFHPSD macro, DFHPDI will check that the partitions all fit within the usable area of the terminal screen.

BUFSZE = (lines,columns) specifies the size of the partition's presentation space. Device limitations mean that the 'columns' value must be equal to the 'columns' value specified by the VIEWSZE operand. The 'lines' value can be greater than or, by default, equal to the value specified by the VIEWSZE operand. A greater lines value implies that the target terminal supports vertical scrolling. The default value of 'lines' is the same as the value specified by the VIEWSZE operand.

CHARSZE(vpels,hpels) specifies the size of the character cell, on an IBM 3290 (or similar) display, to be allocated to each character displayed in the partition. You specify the size as 'vpels' (the number of vertical picture elements), and as 'hpels' (the number of horizontal picture elements). If you code the CHARSZE operand on the DFHPDI macro, you must also code it on the DFHPSD macro, specifying the default cell size.

MAPSFX = mapset-suffix is the partition's 1-character map set suffix. BMS uses the suffix to select map set versions in the same way as the ALTSFX operand of the DFHTCT TYPE = TERMINAL macro. For more information on map set suffixing, see Figure 21 on page 147. If the MAPSFX operand is omitted, a suffix L is assumed if the 'columns' value of the BUFSZE operand is less than or equal to 40; otherwise M is assumed.

ATTRB = ERROR specifies that error messages are to be directed to this partition whenever possible. The partition is cleared before an error message is displayed. Attributes specified on the ERRATT operand of the DFHTCT TYPE = TERMINAL macro will be honored, but the LASTLINE operand will be ignored.

An error message partition can be used directly by a BMS application program, but CICS error messages may be written to this partition, destroying any user data it contains.

If you do not define an error message partition, CICS will send error messages to a cleared unpartitioned screen, obeying any ERRATT operand specified on the DFHTCT TYPE = TERMINAL macro. You are recommended to define an error message partition whenever possible.

Note: The information given here on positioning viewports is necessarily brief. For more information, consult the component description for the device you are using.

End of partition set (DFHPSD TYPE = FINAL macro): This macro ends a partition set definition. Its format is:

```
[partnset] DFHPSD TYPE=FINAL
```

If you code a label on this macro, it should match the label on the DFHPSD macro that started the partition definition.

Assembling and cataloging a partition set

When you have defined a partition set, you must assemble it and store it in the program library. Its name must be given an appropriate suffix, if necessary. You use the dynamic addition transaction (CEDA) to create a PPT entry for it. Otherwise you must create a PPT entry using the DFHPPT system macro.

Specifying a partition within a map set

You use the PARTN operand of the map definition macros (DFHMSD and DFHMDI) to associate an output partition or input partition with a map or map set. The format of the operand is:

```
PARTN=(name[,ACTIVATE])
```

If you code the operand in DFHMSD, it sets the default partition name for all maps in the map set. You can override the default, for individual maps, by the same operand on DFHMDI. The partition becomes the active partition if you specify ACTIVATE. The PARTN operand allows some existing BMS transactions to exploit multiple partitions by map and table changes only.

Any partition operands associated with a map are overridden by the corresponding options on the BMS SEND or RECEIVE commands.

How you code programs to manage partitions

This section describes the commands or keywords you must code in your application programs if you wish to use BMS partition management services. It describes commands that enable you to:

- Handle exceptional conditions caused by your partitioning commands
- Load the application partition set
- Name the partition in which BMS must display data
- Change the active partition
- Name a partition from which BMS must receive input data
- Read data from a partition into a data area before mapping it
- Obtain information about the state of a device's partitions.

The rest of this section explains how CICS support of partitions affects CICS outboard formatting, terminal sharing, and shared use of a screen by BMS and another display manager.

Handling conditions raised by partition operations: A new HANDLE AID keyword (CLRPARTN) allows an application to intercept CLEAR PARTITION requests from the keyboard.

The HANDLE CONDITION command can recognize new error conditions raised by partition management. These conditions are described in "Exceptional conditions" on page 169.

Loading the application partition set: The SEND PARTNSET command loads the correctly suffixed version of a partition set into CICS storage from the CICS program library. The partition set becomes the application partition set, and is loaded onto the terminal when the next BMS output command is executed.

```
SEND PARTNSET[(name)]

Conditions: INVPARTNSET, INVREQ
```

PARTNSET(name) specifies the 1 to 6 character name of the partition set to be loaded. If 'name' is omitted, the terminal is set to base (unpartitioned) state.

Note: A SEND PARTNSET command must not be followed immediately by a RECEIVE command. The two commands must be separated by a SEND MAP, SEND TEXT, or SEND CONTROL command, so that the partition set is sent to the terminal.

Setting the current output partition: The OUTPARTN(name) option of the BMS SEND MAP, SEND TEXT, and SEND CONTROL commands names the partition to which data is to be sent. The partition name can be one or two characters long. This option is ignored if the terminal does not support partitions, or if no application partition set has been specified.

An OUTPARTN option in a SEND MAP command overrides an OUTPARTN operand coded in a BMS map definition macro.

The OUTPARTN option and the LDC option cannot both be specified in the same command (see later in the chapter).

If OUTPARTN is omitted, and a partition set has been loaded, the data is sent to the first partition defined in the partition set.

Conditions: INVPARTN, INVREQ

Setting the active partition: The ACTPARTN(name) option of the BMS SEND MAP, SEND TEXT, and SEND CONTROL commands names the partition to be activated. The cursor is moved into the active partition, and the keyboard is reset for that partition.

The ACTPARTN option is ignored if the terminal does not support partitions, or if there is no application partition set. If the operand is not coded, no partition is activated, and the cursor does not move.

The ACTPARTN option and the LDC option cannot both be specified in the same command (see later in the chapter).

Conditions: INVREQ, INVPARTN

Setting the expected input partition: The INPARTN(name) option of the BMS RECEIVE MAP command names the 'expected input partition' for an input operation. The option is ignored if the terminal does not support partitions, or if there is no application partition set. If INPARTN is omitted, input data is accepted from any partition. If INPARTN is coded, and input is received from a partition other than the expected input partition, CICS takes the following action:

1. It activates the expected input partition (moving the cursor and unlocking the keyboard).
2. It sends the following message to the error message partition named in the application partition set:

   ```
   DFH4190 INPUT DATA ENTERED
   FROM THE WRONG PARTITION.
   RE-ENTER IN PARTITION
   CONTAINING THE CURSOR.
   ```

   No message is issued if no error message partition has been defined.

   Data sent to CICS from the wrong partition is not destroyed. It can be sent when the expected input has been transmitted.
3. It reissues the BMS RECEIVE command.

Steps 1 to 3 are performed three times. If data is still entered from the wrong partition, BMS raises the PARTNFAIL condition.

INPARTN and FROM cannot both be specified in the same RECEIVE MAP command.

Conditions: INVPARTN, PARTNFAIL

Determining the actual input partition: Sometimes you want a CICS application program to accept input from any partition, mapping the data differently according to its origin. You can make it do this by using the RECEIVE PARTN command. The data is received and mapped in separate steps. The RECEIVE PARTN command is followed by code to discover the name of the input partition. Then, mapping is performed using a map designed to compensate for the characteristics of the partition. You code a RECEIVE PARTN command to read data into a data area, and a RECEIVE MAP FROM command to map the data.

```
RECEIVE PARTN(data-area)
{INTO(data-area)|SET(ptr-ref)}
LENGTH(data-area)
[ASIS]

Conditions: INVPARTN, LENGERR
```

The options have the following meanings:

PARTN names the two byte data area into which CICS must put the partition name of the actual input partition.

INTO or SET specify the data area into which CICS must put the inbound data stream. The real terminal AID will be returned in EIBAID, and the inbound cursor position in EIBCPOSN.

LENGTH specifies the name of a data area into which CICS must put the length of the received data string. If INTO is used, LENGTH must be initialized to the length of the INTO area before the RECEIVE command is executed.

ASIS specifies that data must not be translated to uppercase, even if the terminal has been defined with FEATURE=UCTRAN.

ASSIGN command: You use the ASSIGN command to ask CICS for information about your application program environment. There are four options that produce information about partitions, as follows:

PARTNSET specifies that the value required is the one through six character name of the application partition set. A blank value is returned if there is no application partition set.

INPARTN specifies that the value required is the one or two character name of the most recent input partition. A blank value is returned if partitions are not in use.

PARTNPAGE specifies that the value required is the one or two character name of the partition that most recently caused page overflow. A blank value is returned if partitions are not in use.

PARTNS specifies that the value required is a one byte indicator showing that the terminal supports partitions (X'FF') or does not (X'00').

Character cells in partitions: You use the CHARSZE operand of the DFHPSD and DFHPDI macros to define the size (in pels) of the character cells used in an individual partition.

You establish a default cell size for a partition set when you code the CHARSZE operand on the DFHPSD macro.

If you do not specify CHARSZE, the 3290 adopts the cell size specified at CSU **for all partitions**.

In all cases, it will then select the optimum character size for that cell size. If CHARSZE is specified on any DFHPDI macro, it must be specified on all DFHPDI macros within the same partition set, or on the DFHPSD macro.

A cell size set by the DFHPSD macro can be overridden for individual partitions using the CHARSZE operand on the DFHPDI macro. However, if you code CHARSZE on the DFHPDI macro, you must also code CHARSZE on the DFHPSD macro; do not let it default to the size chosen during CSU. The assembler will not accept a partition set that has CHARSZE specified in DFHPSD but not in DFHPDI.

A cell size is specified as the number of vertical picture elements (vpels), and the number of horizontal picture elements (hpels).

If you decide to define partitions that have different values of CHARSZE, you have to work in units of pels when calculating the sizes and positions of partitions. That is, the partition height is the product of the number of rows in the partition and the vertical CHARSZE dimension (vpels); the partition width is the product of the number of columns and the horizontal CHARSZE component (hpels).

The screen has a physical limitation of 750 by 960 pels. This means that you fit the maximum amount of data onto the screen by using a CHARSZE of 6 by 12 with a display area measuring 160 characters by 62 characters. If you use the 160 by 62 screen size but have characters larger than 6 by 12, your partitions will overflow the screen at the bottom and right. During assembly, the partition set definition macros warn you that overflow will occur.

When data is sent to a partition, the 3290 selects the appropriate character set and places characters at the top left hand corners of their cells. If the cell size defined by CHARSZE exceeds the character sizes defined in the character set used, characters will appear widely separated. Conversely, if the cell size is smaller than the defined character size, the characters will be truncated at the bottom and right.

Partitions and outboard formats: 3270 outboard formatting (see "Outboard formatting" on page 172) works with multiple partitions.

Partitions and terminal sharing: You can use partitions with terminal sharing. However, systems involved in terminal sharing should always use the same partition set name to refer to the same partition set.

Partitions and GDDM: CICS output and GDDM output cannot be displayed on the same screen if partitions are being used.

Exceptional conditions: A program using BMS partition management can encounter the following additional exceptional conditions. You can code HANDLE CONDITION commands to override the default action taken by CICS. Unless otherwise stated, the default action is to terminate the task abnormally.

INVPARTNSET partition set is invalid; that is, it is not a partition set.

INVREQ the last request was invalid. This condition indicates a different error for each BMS request that caused it. For more information, see the full description in "Chapter 3.2-5. BMS macro and command reference summary" on page 193.

INVPARTN the partition named by an INPARTN, OUTPARTN, or ACTPARTN operand, or returned by a RECEIVE PARTN command, is not a member of the application partition set.

LENGERR the INTO area named by the RECEIVE PARTN command is too small for the inbound data. This condition does not terminate the task abnormally; instead, CICS truncates data to the size of the INTO area.

PARTNFAIL data has been entered from the wrong partition three times in succession.

Logical device components

The 3601, 3770 batch, 3770, 3790 batch and LUTYPE4 terminals may be configured with a number of subcomponents, such as a printer and a console. Each of these subcomponents is a **logical device component** , and is handled by BMS output commands as if it were a separate terminal.

Defining logical device components

Logical device components are similar to partitions, but unlike partitions, are statically defined by the DFHTCT system macros. See the *CICS/MVS Resource Definition (Macro)* manual for more information on DFHTCT.

The following is defined for each logical device component:

- A 2-character component name.
- A 1-character code, indicating the device type (for example line printer, card punch). This code is assigned by CICS when the logical device component is defined. Possible codes are listed in the *CICS/MVS Resource Definition (Macro)* manual.
- A BMS page size. BMS positions map and text data within this area.
- A BMS page status (AUTOPAGE or NOAUTOPAGE). This is discussed further in "Chapter 3.2-4. Full function BMS" on page 173.

Sending data to a logical device component

A BMS generated data stream can be directed to a particular logical device component by specifying the LDC option on the SEND MAP, SEND TEXT, or SEND CONTROL command. This option specifies a 2 character mnemonic, corresponding to an LDC definition in the TCT.

If BMS output is directed to a logical device component, but the LDC option is omitted from the SEND MAP command, the output is sent to any logical device component associated with the map. The target component is identified by the LDC operand of DFHMSD. You can override the choice of component to which data is to be sent by coding the LDC option on a BMS SEND command. If there is no component associated with the map, or the LDC option is omitted from a SEND TEXT or SEND CONTROL command, the data is sent to the "default logical device component". This default is specified by the DFHTCT macros which define the logical device components for the terminal, and depends on the terminal type. This is discussed further in "Chapter 3.2-5. BMS macro and command reference summary" on page 193.

10/63 magnetic slot reader control

Some IBM display terminals support a magnetic slot reader (an MSR), a device that reads data from small magnetic cards, as an optional feature. Some control the reader themselves, but others (such as the IBM 8775 and the IBM 3643) let you control functions of the reader from your programs.

An MSR has colored indicator lights and an audible alarm to prompt operator actions. You can control these components of an MSR from an application program.

Application programming

You can code application program commands to control an MSR attached to a terminal such as the 8775 or 3643. You use the MSR option of the BMS SEND MAP, SEND TEXT, and SEND CONTROL commands to identify a four-byte field containing device control data. The format of the option is:

```
MSR(data-value)
```

where 'data-value' is the name of the four-byte device control data field. Named constants supplied with CICS provide the most useful combinations of device control commands. A list of the supplied constants appears below. Users can create other constants if they are needed.

Checking that a device supports MSR control: The MSRCONTROL option of the ASSIGN command allows an application program to determine whether a target terminal supports MSR control. The format of the command is:

```
ASSIGN MSRCONTROL(data-area)
```

where 'data-area' is the name of a one-byte data area into which CICS places its response to the ASSIGN command. The response is hexadecimal X'FF' if the terminal supports MSR control, X'00' if it does not.

The ASSIGN command is described in "Chapter 1.6. Access to system information" on page 51.

Supplied constants: A selection of MSR control bit patterns has been created for CICS and stored in the copy book DFHMSRCA. Table 2 on page 171 shows the meaning of each bit. You can use this information to create new constants to add to DFHMSRCA. The patterns are stored as named constants that can be loaded by application program commands. Provision of such constants saves you from having to build a commonly used bit pattern whenever it is required. The constants supplied in DFHMSRCA are as follows:

Constant	Meaning
DFHMSRST	MSR reset. All lights and buzzers off. MSR available for input.
DFHMSCON	Transaction ready for more input. Green and yellow on; emit short buzz; IN PROCESS (user) mode set.
DFHMSFIN	Input complete. Green on; emit short buzz; IN PROCESS mode reset.
DFHMSALR	Operator alert. Green, yellow, and red on; emit long buzz; IN PROCESS mode reset.
DFHMSALS	Operator alert. Green, yellow, and red on; emit long buzz; IN PROCESS mode set.
DFHMSIPY	IN PROCESS state set. Yellow on.
DFHMSIPN	IN PROCESS state reset.
DFHMSLKY	MSR operation inhibited. Yellow on.
DFHMSLKN	MSR input allowed. Green on. Yellow on.
DFHMSAEY	MSR auto enter on. Yellow on.
DFHMSAEN	MSR auto enter off. Yellow on.
DFHMSLBN	Long buzzer suppressed. Yellow on.
DFHMSLBY	Long buzzer permitted. Yellow on.
DFHMSSBN	Short buzzer suppressed. Yellow on.
DFHMSSBY	Short buzzer permitted. Yellow on.
DFHMSNOP	Leave all MSR settings unchanged.

The SEND CONTROL, SEND MAP, or SEND TEXT commands that include the MSR option have no effect until the next RECEIVE command is executed, or until the task terminates. For example, SEND CONTROL MSR commands must be interspersed with RECEIVE commands.

Table 2. MSR control byte values

Byte (purpose)	Bit	Value	Meaning
1 (STATE MASK) If a bit is on in the STATE MASK, the state it represents is adopted by the device if the corresponding bit is also on in the STATE VALUE byte.	0	USER	User mode. Turn the yellow light on if the same bit is on in STATE VALUE.
	1	LOCK	Locked/Unlocked. If locked, MSR input is inhibited.
	2	AUTO	Autoenter on/off. If set on, any card read by the MSR will cause an ENTER operation. If off, only a secure card causes an ENTER.
	3	Ai1S	Suppress audible alarm 1
	4	Ai2S	Suppress audible alarm 2
2 (STATE VALUE) Modifies state to on or off if the corresponding bit is set on in STATE MASK.			
3 (INDICATOR MASK) Performs a similar function to STATE MASK, but for indicators.	0		Light 1 (Green)
	1		Light 2 (Yellow)
	2		Light 3 (Red)
	3		Audible Alarm 1 (long buzz)
	4		Audible Alarm 2 (short buzz)
4 (INDICATOR VALUE) Performs similar function to STATE VALUE.			

Trigger fields

You can use display trigger fields to initiate input to an application program. A trigger field is one that is transmitted to the host processor as soon as the terminal operator has modified the field and then tries to move the cursor out of it. The trigger attribute is ignored if the operator has not modified the trigger field.

Application programming

CICS application programs can receive data from trigger fields. Any keystroke (for example, the tab key) that makes the cursor leave a trigger field containing modified data causes the terminal to transmit an attention identifier (AID), plus the contents of the field, to CICS. The value of the AID is X'7F'.

You define trigger fields during map definition. Application programs that use trigger fields should contain the HANDLE AID TRIGGER command, passing control to a label that processes trigger input. The code at this label should process the input rapidly, as a long delay can inhibit input. This is because, after sending trigger data, the terminal stores further keystrokes, but cannot process them until the host processor acknowledges the trigger. The operator can perform up to 30 keystrokes during the wait.

Following receipt of a trigger field, the application program must validate the trigger field data, and respond to the 8775 terminal as follows:

- If validation is successful, the application program should issue a SEND MAP, SEND TEXT, or SEND CONTROL command specifying the FREEKB option and omitting the ERASE and ERASEAUP options. The command must address the partition containing the trigger field; it allows the 8775 terminal to process stored keystrokes.
- If validation is unsuccessful, the application program can instruct the 8775 to discard stored keystrokes, as follows:
  - By issuing a SEND MAP, SEND TEXT, or SEND CONTROL command that does not specify the FREEKB option, and/or is not directed to the partition containing the trigger field. Typically, this SEND command would issue an error message indicating why the trigger field input was rejected.
  - By issuing a SEND MAP, SEND TEXT, or SEND CONTROL command specifying the ERASE, ERASEAUP, or ACTPARTN options.
  - By issuing a RECEIVE MAP, RECEIVE PARTN, or a terminal control RECEIVE command.
  - By terminating the transaction.

Defining maps to provide trigger validation: Parts of a display can be assigned the trigger validation attribute by coding the VALIDN=TRIGGER operand in one or more of the map definition macros, DFHMSD, DFHMDI, and DFHMDF. The form of the operand is:

```
[,VALIDN=([MUSTFILL][,MUSTENTER]
       [,TRIGGER])]
```

As for other map set definition macro operands, values established in the DFHMSD macro apply, by default, to maps within the set, and fields within those maps.

CICS-supplied trigger constants: As well as being able to define maps with trigger fields, you can write programs that set the trigger attribute of a field dynamically in an application program. Copybook DFHBMSCA contains constants that provide the bit settings for the attribute byte. The constants are listed in "Chapter 3.2-5. BMS macro and command reference summary" on page 193.

Handling the trigger AID: The TRIGGER option of the HANDLE AID command allows a program to pass control to a handling routine upon receiving input from a trigger field. The form of the command is:

```
HANDLE AID TRIGGER(label)
```

where 'label' is the name of a program statement to which control is passed when the program receives a TRIGGER AID.

Outboard formatting

Outboard formatting is a technique for reducing the amount of line traffic between the host processor and an attached subsystem. The reduction is achieved by sending only variable data across the network. This data is combined with constant data by a program within the subsystem. The formatted data can then be displayed.

You can use outboard formatting with either a 3650 Host Communication Logical Unit, or an 8100 Series processor with DPPX and DPS Version 2. Maps used by the 3650 must be redefined using the 3650 transformation definition language before they can be used. For more information, see the section describing BMS in the *CICS/OS/VS IBM 3650/3680 Guide*. Maps to be used with the 8100 must be generated on the 8100 using either a utility of SDF/CICS or the interactive map definition component of the licensed program DPS Version 2. For more information on either of these methods, see the *DPPX/DPS Version 2 System Programming Guide*.

If a program in the host processor sends a lot of mapped data to subsystems, you can reduce line traffic by telling BMS to transmit only the variable data in maps. The subsystem must then perform the mapping operation when it receives the data. BMS prefixes the variable data with information that identifies the subsystem map to be used to format the data.

Terminals that support outboard formatting will have BMSFEAT=OBFMT specified in their TCT entries. When a program issues a SEND MAP command for such a terminal, and the specified map definition contains OBFMT=YES, BMS assumes that the subsystem is going to format the data. It therefore generates an appropriate data stream.

If you send a map that has OBFMT=YES to a terminal that does not support outboard formatting, BMS will ignore the OBFMT operand.

Users of full BMS may be interested to know that floating maps, which are discussed in "Chapter 3.2-4. Full function BMS" on page 173, can be sent to an 8100 processor for outboard formatting.

Block data format

A symbolic description map can be generated using block data format by specifying DATA=BLOCK on the DFHMSD and DFHMDI map definition macros.

The block data format of the symbolic map is an image of the source map. This image contains one character for each character position in the source map. If the source map is 80 columns wide by 5 lines deep, the symbolic map data structure will contain 412 characters, including a 12 byte TIOA prefix. The fields in the symbolic map are positioned as in the source map. A field positioned at column 10 line 2 in the above source map would have an attribute field 101 characters from the start of the symbolic map (allowing a 12 byte TIOA prefix), and a data field 102 bytes from the start of the symbolic map. As there is only room for one attribute field in the block data format, block data cannot have extended attributes.

The block data symbolic map format can be useful if an application program has built a printer page image and wishes to display it on a screen. Most applications, however, will find the normal field data symbolic map format more useful.

Chapter 3.2-4. Full function BMS

This chapter describes the additional facilities provided by full function BMS. For convenience, 'full function BMS' will be shortened to 'full BMS'. Full BMS supports the same range of devices as standard BMS, but provides extra function, as follows:

- Logical message handling. This enables you to request:
  - Terminal operator paging using the PAGING option
  - Cumulative output data formatting using the ACCUM option
  - Combined use of ACCUM and PAGING.
- Message routing. The ROUTE command allows an application program to build device dependent data streams for several terminals simultaneously.
- Message switching. The message switching transaction (CMSG) allows a terminal operator to send a message to a list of terminals, or a list of terminal operators.
- Facilities for intercepting formatted device-dependent data and relaying it to a terminal later. This facility enables you to develop output routines to modify data streams before output. This is not recommended, as the format of the device-dependent data stream cannot be guaranteed.

  You use the SET option to intercept output data, and then relay it using the SEND TEXT MAPPED command.

Logical message handling

A logical message is a collection of formatted output data produced by chaining several smaller items of data. You build a logical message by coding a series of BMS SEND commands, each having either the ACCUM option or PAGING option, or both. When you build a page of message data, CICS does not send the data until you issue a SEND PAGE command. However, if you produce more than a pageful of data, it will send the data in installments (one every time page overflow occurs).

You complete a logical message by issuing a SEND PAGE command. Alternatively, you can cancel the message by issuing the PURGE MESSAGE command.

If you issue a SYNCPOINT command, or terminate your transaction, before issuing SEND PAGE, CICS usually assumes that you meant to issue SEND PAGE. Consequently, it terminates the message, and sends the data. However, if your logical message does not contain a single complete page, the logical message will be lost. You must therefore always explicitly code a SEND PAGE command before the SYNCPOINT command or before the transaction is terminated, and not rely on any implied SEND PAGE command.

The PAGING option tells BMS to send the output data to temporary storage, from where an operator can retrieve it using the terminal operator paging transaction. With this transaction, the operator can view the pages in any order, and as often as necessary. When the message is no longer needed, the operator can delete it and continue normal transaction processing.

The ACCUM option tells BMS to accumulate pages of output data, and to send each page when it is complete. BMS sends the pages directly to the termina' unless PAGING or SET is specified as well as ACCUM. If PAGING is specified, BMS accumulates them on temporary storage. If SET is specified, BMS returns completed pages to the application program, as described below. This is an economical way of building pages of display data, because CICS fits as much data as possible on each page before sending it.

It is usual to build logical messages using both the ACCUM and PAGING options, in combination with floating maps or cumulative text (which are discussed later). If you do this, BMS optimizes the use of the available display area on the target terminal, building several pages each containing a pageful of data. It then writes each page to CICS temporary storage, and initiates the terminal operator paging transaction when the SEND PAGE command is encountered, see "Example of how to use paging" on page 181.

The following rules apply while a BMS logical message is active:

1. Only one BMS logical message can be active at a time.
2. All SEND MAP, SEND TEXT and SEND CONTROL commands in a BMS logical message must specify the same combination of ACCUM, PAGING, SET, TERMINAL, and REQID options as the BMS command that started the logical message.
3. A ROUTE or SEND PARTNSET command cannot be issued.
4. SEND CONTROL commands can be used with either SEND MAP or SEND TEXT commands. However, SEND MAP and SEND TEXT commands can be mixed in a BMS logical message only if the text data and mapped data are sent to different partitions or LDCs.

With only one exception, the INVREQ condition is raised if any of these rules are violated. The exception is that the IGREQID condition is raised if the REQID for a SEND command differs from the REQID for the whole message.

The ACCUM option and the PAGING option are discussed later in this section; the SYNCPOINT command is described in "Chapter 5.6. Recovery (sync points)" on page 331. The SEND PAGE and PURGE MESSAGE commands, which apply only to logical messages, are described below.

BMS message recovery/restart: Following a warm or emergency restart, you can retrieve logical messages under certain circumstances.

BMS provides message recovery for routed and nonrouted logical messages. Recoverable messages must satisfy the following requirements:

- The PAGING option must have been specified in the BMS SEND commands that built the logical message.
- The BMS default REQID ('**') or the specified REQID for the logical message must have been identified to the temporary storage program (via the DFHTST macro) as recoverable.
- The task that built the message must have reached its logical end of task.
- The temporary storage program and the interval control program must also support recovery.
- Should any of the pages have been retrieved, the paging session will be effectively restarted. For a warm restart where the paging session has been started from a terminal that had an operator signed on to it, the message will only be retrievable by the same operator on the same terminal.
- The message has not been purged.

Terminal operator paging and display data accumulation are described later. First, consider the SEND PAGE and PURGE MESSAGE commands.

SEND PAGE command: The syntax of the SEND PAGE command is as follows. This command is only available on full function BMS.

```
SEND PAGE
[RELEASE[TRANSID(name)]|RETAIN]
[TRAILER(data-area)]
[SET(ptr-ref)]
[AUTOPAGE[CURRENT|ALL]|NOAUTOPAGE]
[OPERPURGE]
[FMHPARM]
[LAST]

Conditions: IGREQCD, INVREQ,
RETPAGE, TSIOERR, WRBRK
```

The SEND PAGE command causes BMS to generate a device dependent data stream for the last (perhaps the only) page of data. Typically this last page is only partially full.

Options can be included to specify how much control the terminal operator should have over the disposition of the logical message (AUTOPAGE, NOAUTOPAGE, and OPERPURGE), to determine whether control should return to the application program after transmission of the logical message (RELEASE and RETAIN), to add trailer data to a text logical message (TRAILER), and to return the device dependent data stream for the last page of a logical message to the application program (SET).

The TRAILER option is only relevant to text messages, and is discussed in "Cumulative text formatting" on page 181.

All options except SET and TRAILER apply only to paging logical messages. Their use is described under "Logical messages for terminal operator paging" on page 175.

PURGE MESSAGE command: The PURGE MESSAGE command simply deletes the current logical message, including any pages of device dependent data stream already written to CICS temporary storage. The application program may then build a new logical message.

The syntax of the PURGE MESSAGE command is as follows. This command is only available with full BMS.

```
PURGE MESSAGE

Condition: TSIOERR
```

Logical messages for direct terminal output

If the TERMINAL option (the default) is used for this logical message, the SEND PAGE command immediately sends the last page of the device dependent data stream to the terminal. The RETAIN and RELEASE options are then ignored.

You can use the ACCUM and TERMINAL options together to build a single page out of several maps or blocks of text. Data stream generation and transmission for this single page is then deferred until the logical message is terminated by a SEND PAGE command. This gives better performance on a display device than a series of separate BMS SEND commands.

This form of processing is essential if a printer page for a non-3270 printer or a 3270 printer whose DFHTCT TYPE=TERMINAL macro specifies TRMTYPE=SCSPRT is composed of several maps or text blocks. Otherwise, each BMS SEND MAP, SEND TEXT, or SEND CONTROL command will send a whole page, using a form feed or an appropriate number of blank lines. This is discussed in "Chapter 3.2-3. Standard function BMS" on page 159.

Logical messages with the SET option

If the SET option is used for this logical message, the SEND PAGE command returns the last page of a device dependent data stream to the application program. This is further discussed in "Returning mapped data to a program before output" on page 189. The RETAIN and RELEASE options are ignored if they are specified with SET.

Logical messages for terminal operator paging

If the PAGING option is specified on BMS SEND MAP, SEND TEXT, and SEND CONTROL commands, the device dependent data stream built by BMS is sent to CICS temporary storage for subsequent retrieval by an operator using the terminal operator paging transaction. A separate temporary storage queue is used for each BMS logical message. The queue name is determined by CICS, using the value of any REQID operand of the SEND MAP, SEND TEXT, and SEND CONTROL commands as the first 2 characters of the queue name. If the REQID option is omitted, the first 2 characters of the queue name are '**'. The REQID option allows application programs to send some BMS logical messages to recoverable temporary storage, and some to nonrecoverable temporary storage, as discussed in "BMS message recovery/restart" on page 174.

The temporary storage queue for a BMS logical message is deleted by a PURGE MESSAGE command, or (ignoring routing) by the terminal operator purging the message at the end of a terminal operator paging session.

The application program terminates the logical message, and initiates the terminal operator paging transaction by the SEND PAGE command, whose syntax is shown in "SEND PAGE command" on page 174.

If the PAGING option is used for this logical message, the SEND PAGE command writes the last page of device dependent data stream to CICS temporary storage. The SEND PAGE command then initiates the terminal operator paging transaction. This initiation is controlled by the RETAIN and RELEASE options as follows:

- If RETAIN is specified, the terminal operator paging transaction is initiated immediately. The first page is sent to the terminal (or the first page in each partition), and the terminal operator can use the terminal operator paging commands described in "Terminal operator paging commands" on page 176. The terminal operator paging transaction operates conversationally in this mode. When the terminal operator terminates the paging session by purging the message, control is returned to the application program following the SEND PAGE RETAIN command.

  The application program retains control after a paging session.

  The OPERPURGE option specifies that CICS is to delete the BMS logical message only when the terminal operator requests deletion by an explicit page purge command. If the option is omitted, CICS deletes the message if the operator enters data that is not a paging command.

  If OPERPURGE is not specified on the SEND PAGE command, the terminal operator can terminate the paging session by entering data which is not a valid terminal operator paging command. This data can be accessed by the application program issuing a RECEIVE MAP or a terminal control RECEIVE following the SEND PAGE RETAIN command.

- If RELEASE is specified, the terminal operator paging transaction immediately displays the first page of data (or the first page in each partition). The application program transaction is then terminated, and the paging session continues pseudoconversationally. When the terminal operator terminates the paging session by purging the message, control is returned to CICS. CICS will then initiate any transaction specified by the TRANSID option of the SEND PAGE RELEASE command. The application program releases control after the SEND PAGE command.

  A SEND PAGE RELEASE must be the last executed command in a transaction. SEND PAGE RELEASE can be thought of as a combination of SEND PAGE RETAIN and a CICS RETURN command.

- If neither RETAIN nor RELEASE is specified, the terminal operator paging transaction is initiated for execution when the current application transaction terminates. The terminal operator paging transaction is queued for execution on the target terminal, following any other transactions queued for that terminal.

The AUTOPAGE and NOAUTOPAGE options overwrite any paging status specified by the PGESTAT option of the DFHTCT TYPE=TERMINAL macro. The PGESTAT value is honored if neither AUTOPAGE nor NOAUTOPAGE is specified on the SEND PAGE command. AUTOPAGE specifies that each page of the logical message is to be sent to the terminal without terminal operator paging commands. This is frequently used for printers.

The options CURRENT and ALL apply to 2741 terminals only, and are discussed in "Map definition macro operand summary" on page 195.

If an error occurs during the processing of a SEND PAGE command, control is returned to the application program, and the RETAIN or RELEASE specification is ignored. The logical message is not considered complete. The application program should either retry the SEND PAGE operation or delete the logical message.

Terminal operator paging commands: The terminal operator commands provided by the terminal operator paging transaction are discussed in the *CICS/MVS CICS-Supplied Transactions* manual. They are summarized here for completeness.

The terminal operator paging transaction is usually initiated automatically by a BMS application program issuing a SEND PAGE command for a logical message that was built using the PAGING option. However, the terminal operator has to start the paging transaction explicitly (by entering CSPG) if automatic transaction initiation (ATI) is not available with the terminal (see the TRMSTAT operand of the DFHTCT TYPE=TERMINAL system macro), and terminal operator paging was initiated with a SEND PAGE specifying neither RELEASE nor RETAIN.

The terminal operator paging transaction provides the following facilities:

Page retrieval: The terminal operator can enter a command (or use a PF key if these have been defined via the SKRxxx options of the DFHSIT macro) to retrieve the first, last, nth, next, or previous page. If partitions are used, the page retrieval command relates to the partition in which it was entered. If LDCs are used, the LDC name is entered as part of the page retrieval command.

Page query: The terminal operator may obtain a list of logical messages queued for his terminal. This list includes the BMS assigned message identifier, and any TITLE option specified by the ROUTE command of the application program that built the message. The terminal operator may use the message identifiers to retrieve pages for a specified message.

Set to autopage: The terminal operator of a printer keyboard terminal may request that all remaining pages (for all partitions or LDCs) be sent without further operator intervention.

Page copy: The terminal operator may copy the current page to another terminal (generally a printer). BMS does any reformatting that may be needed, if the target terminal for the copy has a smaller page size than the source terminal.

Page chaining: The terminal operator may invoke another transaction, which communicates with the terminal in the normal way. This invoked transaction may in turn build pages, which if the SEND PAGE command in the invoked transaction specified RETAIN or RELEASE are 'chained' to the pages built by the original transaction. The terminal operator may then retrieve pages for either transaction, possibly for comparison purposes.

The normal BMS application is unlikely to use page chaining.

Page purge: When the terminal operator has finished reviewing the pages of a logical message it can be purged by an explicit page purge command, or implicitly by entering data which is not a paging command. However, this implicit purging is only possible if the OPERPURGE option was not specified on the SEND PAGE command which initiated the paging session.

The process is more complex if page chaining is used. The terminal operator can purge various levels of chained pages.

The PAGE PURGE command purges all pages in all partitions or LDCs for the appropriate logical message.

Cumulative output processing (ACCUM option)

If the ACCUM option is used in a BMS logical message, a device dependent data stream is generated on 'page overflow'. BMS disposes of each page of a device dependent data stream as follows:

1. If the TERMINAL option (the default) is used for this logical message, BMS immediately sends the page to the terminal. This may be useful for printers, but successive pages will overwrite each other on a screen.
2. If the SET option is used for this logical message, BMS returns the page to the application program.
3. If the PAGING option is used for this logical message, BMS writes the page of device dependent data stream to CICS temporary storage.

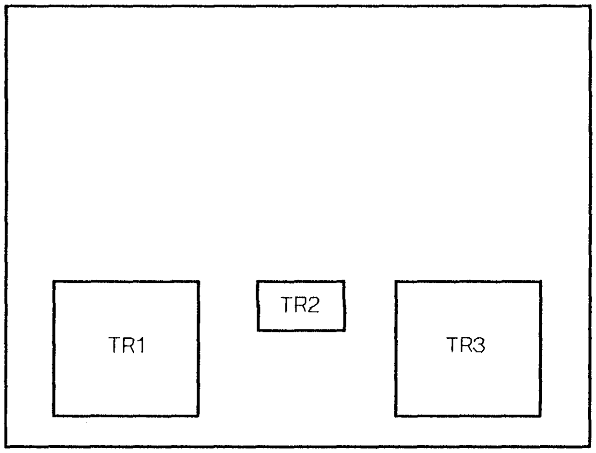
The page overflow condition is fully discussed in "Handling page overflow" on page 180. Briefly, page overflow occurs when the next BMS map or block of text will no longer fit on the current page of the target terminal.

Floating maps, header and trailer maps: In minimum and standard BMS, all maps were positioned absolutely, as specified by the LINE=number and COLUMN=number operands of the DFHMDI map definition macro. In full BMS, maps can "float". That is, they can be positioned relative to the previous map. This is done by coding LINE=SAME|NEXT or COLUMN=SAME|NEXT on the DFHMDI map definition macro. Floating maps can be sent to a terminal by successive SEND MAP ACCUM commands, until no more will fit on the current page. Page overflow then occurs, and can be handled as described in "Handling page overflow" on page 180.

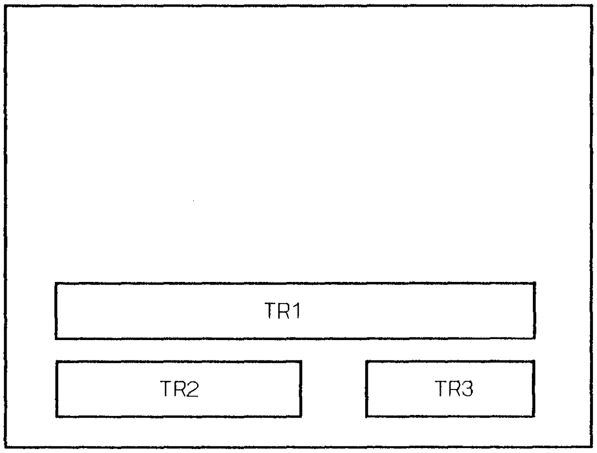
Avoid using floating maps, header maps, or trailer maps in a RECEIVE MAP command. This is because the floating map will be positioned on an empty page and the meanings of the LINE, COLUMN, and JUSTIFY operands of the DFHMDI macro are modified as explained in "Map definition macro operand summary" on page 195.

A map can be defined as a TRAILER map by specifying TRAILER=YES on the DFHMDI map definition macro. JUSTIFY=LAST is usually also specified to position the map at the bottom of the page. BMS allows for trailer maps in determining on each BMS SEND MAP ACCUM command whether the map referenced by this command will fit on the current page. BMS does this by leaving space for the largest trailer map in the map set referenced by the SEND MAP ACCUM command. If several map sets are used to compose a page, each map set which contains floating maps should also contain a trailer map (a dummy map which is not otherwise used will suffice) to allocate space for the actual trailer map(s) transmitted by the page overflow process.

A dummy trailer map may also be needed to allocate the overall trailer area if the application program sends several trailer maps. Its depth must be at least equal to the combined depths of the trailer maps. This is illustrated in the following diagrams:



No dummy trailer required



Dummy trailer required

An attempt to place more lines of trailer data on the page than are available, causes the trailer data to be placed on a separate page by itself.

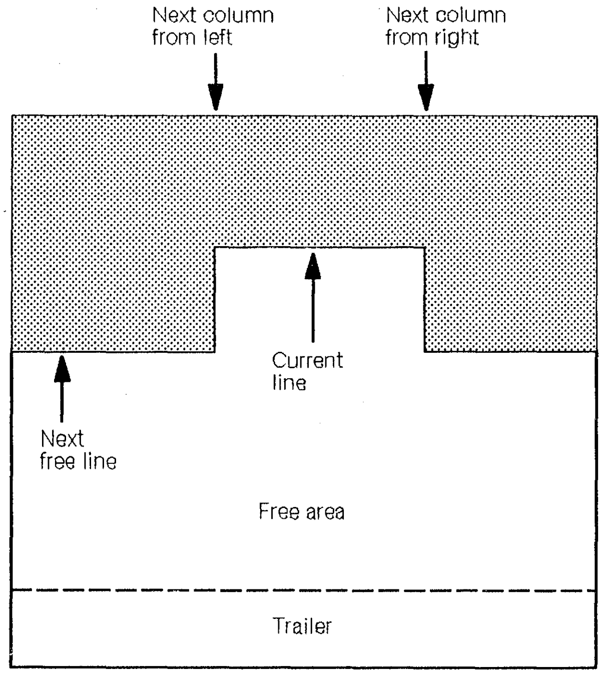
A map can be defined as a HEADER map by specifying HEADER=YES on the DFHMDI map definition macro. JUSTIFY=FIRST is usually also specified to complete processing of the previous page, and to begin a new page. An attempt to place more header data on the page than the page can contain causes multiple pages to be created.

If a header map is not used, JUSTIFY=FIRST must be specified for the first map used to start a new page. Failure to specify this will result in a further page being sent for each SEND MAP command (as each map will be placed at the bottom of a page, causing page overflow).

Map positioning: This section explains the full capability of the BMS map positioning algorithm. In practice, however, it is unlikely that this full capability is needed. If JUSTIFY=RIGHT is avoided on the DFHMDI map definition macro, BMS will fill the page from top left to bottom right. Any unused areas to the top and left of the current map are thus unavailable for maps on this page.

The position of a map on a screen is determined by two major factors: the current contents of the screen, and the values coded for the LINE, COLUMN, and JUSTIFY operands of the DFHMDI macro. Positioning is also affected if the DFHMDI macro specifies HEADER=YES or TRAILER=YES, and by the depth of the deepest trailer map in the map set.

At any instant, the part of the screen that is still available for maps is in the form of either an L, a reversed L, a rectangle, or an inverted T, as shown by the unshaded area in the following diagram. The most likely case is a rectangle.



The shape and size of this area is represented internally by four variables: current line, next free line, next column from left, and next column from right.

Two other pointers are maintained that are relevant to map placement though they do not affect the area available: left reference column and right reference column, which are used when COLUMN=SAME is specified.

The trailer size is equal to the number of lines that would be occupied by the deepest trailer map in the map set currently in use. It is determined when the map set is assembled, and is copied from the map set whenever one is loaded. The trailer size is assumed to be zero if a HANDLE CONDITION OVERFLOW command is not in effect.

The area defined by trailer size is not available for mapping unless no overflow label has been specified or the map has TRAILER=YES specified in its DFHMDI macro.

If JUSTIFY=FIRST is specified, the map is placed on a new page, so that the only maps above it are the header maps used in overflow processing. The LINE operand may also be used with JUSTIFY=FIRST to place the map below the top of the page.

If JUSTIFY=LAST is specified, the map is placed as low as possible on the page. For a nontrailer map, this is immediately above the trailer area; for a trailer map, it is at the bottom of the page. In the absence of an overflow label, the trailer area is null and JUSTIFY=LAST places the map at the bottom of the page.

A map defined with JUSTIFY=LAST cannot be used in input operations unless it was previously put out without the ACCUM option, in which case JUSTIFY=LAST is ignored and the map is positioned at the top of the page. JUSTIFY=BOTTOM is equivalent to JUSTIFY=LAST for cumulative mapping, and provides a similar capability for noncumulative mapping, and for input.

For SEND MAP commands (without ACCUM) and RECEIVE MAP commands, JUSTIFY=BOTTOM causes the map to be positioned at the bottom of the screen if the number of lines in the map is specified in the SIZE operand. Space is not reserved for any trailer maps in the map set. JUSTIFY=BOTTOM is ignored if the number of lines in the map is not specified in the SIZE operand. If JUSTIFY=BOTTOM and LINE are both specified, the value specified in LINE will be ignored.

JUSTIFY=BOTTOM is intended to allow the positioning of a map at the bottom of a screen whatever the screen size, and to allow input from such a map without the application program having to take account of the screen size in use. It can be used, for example, if command input is required to be from the bottom lines of the screen on a variety of display models.

The LINE operand specifies the line of the screen on which the first line of the map is to be placed. The initial determination of this line is made without regard to the specification of the COLUMN operand or the columns available on the screen on that particular line. If the map will not fit on the chosen line, the first subsequent line that will satisfy the column requirements is selected.

If LINE=SAME or LINE=NEXT is specified, the initial line selected for the start of the map is the current line or the next free line, respectively. If a number is specified in the LINE operand, the line with that number is selected, provided the number is greater than or equal to the number of the current line; if not, the overflow condition is raised so that the map can be placed on the next page.

The line selected becomes the new current line and, if it is below the next free line, the next free line is reset to the same line; the next column from the left and right are also reset, to the left and right margins respectively.

If the line selected is such that the end of the map extends into the trailer area for a non-trailer map or beyond the end of the page for a trailer map, the overflow condition is raised and the map will be placed on the first available line of the next page when the request is reissued after handling the overflow.

The COLUMN specification may be either NEXT, SAME, or a number and is processed in conjunction with the LEFT or RIGHT specification of the JUSTIFY operand. JUSTIFY=LEFT is the default and implies that the column specification is related to the left-hand margin. Conversely, JUSTIFY=RIGHT implies that the column specification is related to the right-hand margin. For the purposes of this explanation, it is assumed hereafter that JUSTIFY=LEFT has been specified (or applied by default).

If COLUMN=NEXT is specified, the column chosen for the map is the next column from the left. If a numeric value is specified, the column with that number is chosen, counting from the left. If COLUMN=SAME is specified, the left reference column is chosen. (The left reference column is the one that was most recently specified by number with JUSTIFY=LEFT.)

The map is then checked to ensure that its right margin is not to the right of the next column from the right. If it is, the map will not fit into the remaining space, so a new line must be selected. This will be either the next full line or, if the map is too deep, the first available line on the next page.

Finally, the column pointers are updated by setting the next column from the left to the right margin of the map, and, if COLUMN=number was specified, by setting the left reference column to the specified column number.

Map positioning examples: The effects of the mechanisms described above are illustrated by the following examples. The examples show the interactions between SIZE, LINE, COLUMN, and JUSTIFY=LEFT or RIGHT. Header and trailer maps and JUSTIFY=FIRST or LAST are not brought into the examples.

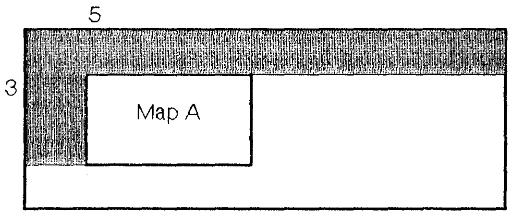
In processing a BMS command, BMS determines whether the area of the page required by the map is wholly available or whether any part of it has been used by an earlier command. "Used" means actually filled by a map or rendered unavailable.

When the LINE operand of the DFHMDI macro is coded, all lines above the specified line are unavailable.

When JUSTIFY=LEFT is coded (or applied by default), as in the following definition:

```
MAPA DFHMDI   ...,LINE=3,COLUMN=5,
                  JUSTIFY=LEFT,...
```

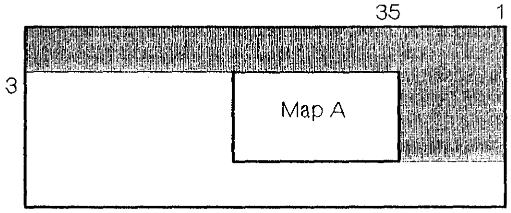
all columns to the left of the leftmost map column, for the full depth of the map, are unavailable, as shown (by the cross hatching) in the following diagram:



When JUSTIFY=RIGHT is coded, as in the following definition:

```
MAPA DFHMDI   ...,LINE=3,COLUMN=35,
                  JUSTIFY=RIGHT,...
```

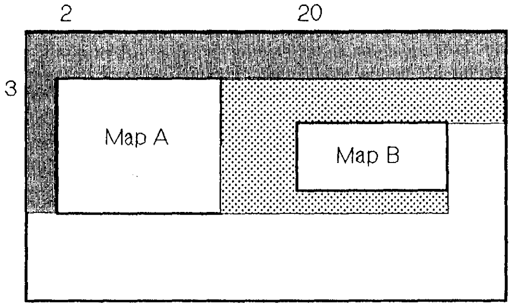
all columns to the right of the rightmost map column, for the full depth of the map, are unavailable, as shown in the following diagram:



When two or more maps are placed so that they share certain lines, as in the following definitions:

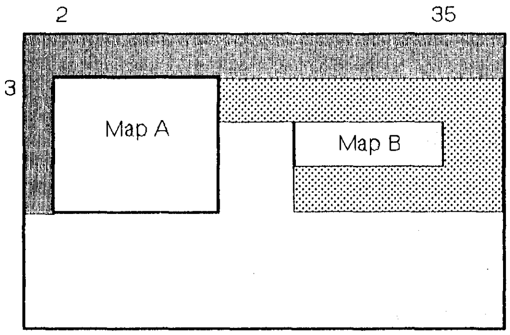
```
MAPA DFHMDI   ...,LINE=3,COLUMN=2,
                  JUSTIFY=LEFT,...
MAPB DFHMDI   ...,LINE=4,COLUMN=20,
                  JUSTIFY=LEFT,...
```

all columns beneath a map that ends higher are unavailable to the depth of the map that ends lowest, as shown in the following diagram:

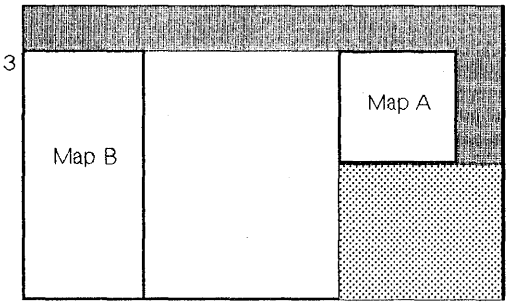


Similarly unavailable are all columns to the left (if the higher map is left justified) or to the right (if the higher map is right justified) of the "used" area beneath the higher map. The following two diagrams illustrate similar situations:

```
MAPA DFHMDI   ...,LINE=3,COLUMN=2,
                  JUSTIFY=LEFT,...
MAPB DFHMDI   ...,LINE=4,COLUMN=35,
                  JUSTIFY=RIGHT,...
```



```
MAPA DFHMDI  ...,LINE=3,COLUMN=40,
                 JUSTIFY=RIGHT,...
MAPB DFHMDI  ...,LINE=3,COLUMN=1,
                 JUSTIFY=LEFT,...
```



The effect of several different maps on one page is shown in Figure 22 on page 181.

If an area of the page directly specified for a map has already been used by a previous map, the overflow condition is raised. This condition is handled as described in the next section.

Handling page overflow: Page overflow occurs when the number of lines in the requested map plus the number of lines in the largest trailer map in the map set (if there are any trailer maps) is greater than the number of lines remaining in the page being built.

When page overflow occurs, BMS transfers control to a label in the application program. It does not call a subroutine. There is no easy way of returning from the overflow processing to the application program command that caused overflow.

The label to which control is transferred is specified by a HANDLE CONDITION OVERFLOW command. The data which was to have been mapped, but which caused the overflow, is not mapped by BMS and remains unaltered.

If partitions or LDCs are used, pages are accumulated separately for each partition or LDC. This complicates page overflow processing, as discussed in "Page overflow and partitions or LDCs."

Overflow can occur on a logical message being built for routing. Again this complicates page overflow, as discussed in "Routing and page overflow" on page 189.

This simple process is inadequate if the "body" of the page is composed of several different kinds of map. The application program must remember which map name it is about to process by a SEND MAP ACCUM command so that it can resend this map and its associated application data in the event of page overflow.

BMS maintains the overflow environment for as long as the application program issues BMS commands using maps defined as headers or trailers. While in the overflow environment, the overflow condition is not raised, as this may result in an infinite loop. The first use of a map that is not defined as a header or trailer terminates overflow processing.

If an overflow label has not been specified via a HANDLE CONDITION OVERFLOW command, no overflow occurs and new pages are forced out automatically.

An overview of overflow processing is given in Figure 23 on page 182.

Page overflow and partitions or LDCs: Pages are accumulated separately for each partition or LDC. Thus page overflow occurs on a partition or LDC basis. Page numbers are maintained on a partition or LDC basis, so that the ASSIGN PAGENUM command returns the page number for the most recently overflowed partition or LDC.

The ASSIGN PARTNPAGE command returns the partition name of the most recently overflowed partition. Similarly the ASSIGN LDCMNEM command returns the name of the most recently overflowed LDC.

If LDCs are used, the overflow processing code in the application program must send header and trailer maps to the LDC which has just overflowed. Otherwise the INVREQ condition is raised.

If partitions are used, the overflow processing code in the application program is not obliged to send header and trailer maps to the partition which has just overflowed. However, the application program must then avoid sending a header or trailer map to a different partition, which causes that partition to overflow.

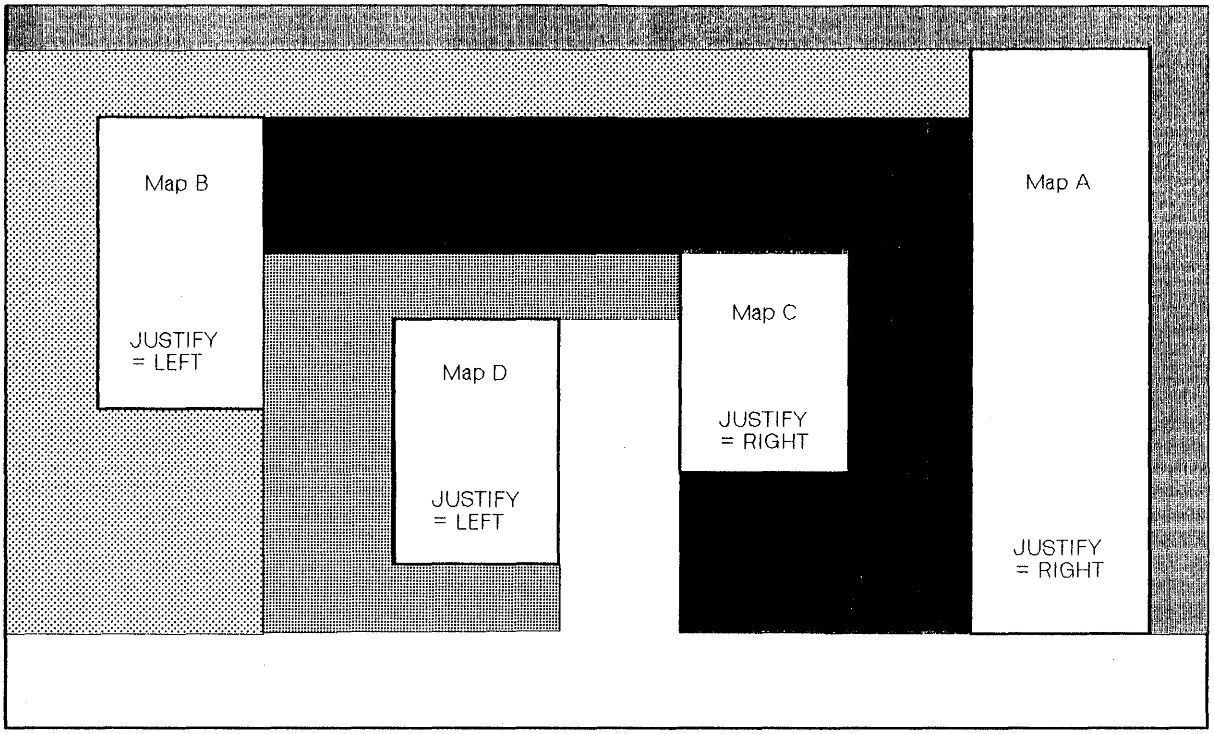


Figure 22. Many BMS maps on one page

Example of how to use paging: This section shows you how to build a BMS logical message and how to handle page overflow. An order entry application is assumed, in particular a transaction to display customer orders on a screen with 80 columns and an arbitrary number of lines.

This transaction uses the following maps:

1. A header map (called URDHEAD) containing the customer's name and address, the order number, and column headings for the following order lines. This map is to be displayed at the top of every page. It is defined with HEADER=YES, JUSTIFY=FIRST, LINE=1, and COLUMN=1.
2. A floating map (called ORDLINE) containing part number, part description, quantity, and price. A number of these will be displayed on the screen, depending on the size of the screen and the number of different parts in the customer order. This map is defined with LINE=NEXT and COLUMN=1.
3. A trailer map (called ORFTRL) containing a page number and instructions for the operator on how to view the next page. This map is displayed at the bottom of each page, including the last. This map is defined with TRAILER=YES and JUSTIFY=LAST.

The PL/I version of the program for this transaction is shown in Figure 24 on page 183.

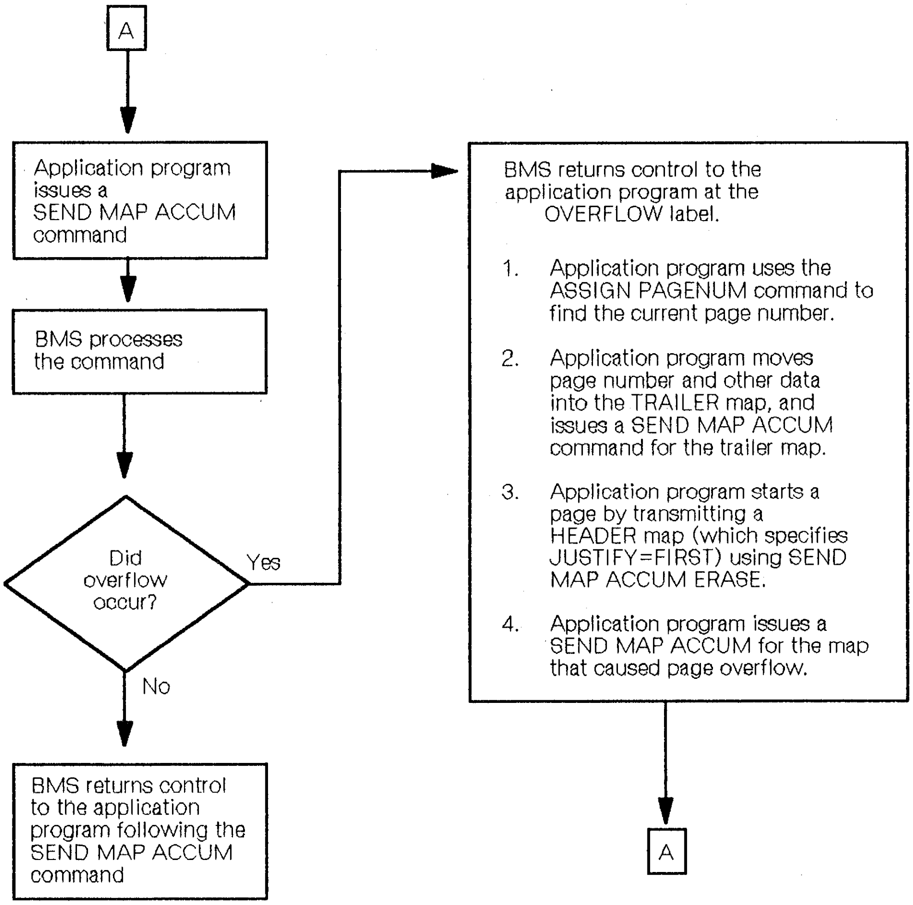
Cumulative text formatting

By specifying the ACCUM option on the SEND TEXT command, you can accumulate blocks of text from multiple SEND TEXT ACCUM commands and can format them to produce complete pages of text by BMS.

Note: The standard-function SEND TEXT command allows you to send one or more lines to the output device and is not limited by page boundary. When the SEND TEXT command is used with the ACCUM option, the output is page-bounded. This means that, if the output device is a printer, the lines of text that are printed are automatically followed by a form feed.

You can use the HEADER and TRAILER options to specify data to be placed at the top and bottom of each page. As a page boundary can occur as a result of any SEND TEXT ACCUM command, the HEADER and TRAILER options should be repeated on each SEND TEXT ACCUM command in the BMS logical message. Automatic page numbering at a user specified location in the header and trailer data is possible.

A text logical message must be terminated by a SEND PAGE or PURGE MESSAGE command in the normal way. The TRAILER option of the SEND PAGE command allows trailer data to be specified for the last partially full page.


A
Application program issues a SEND MAP ACCUM command
BMS processes the command
Did overflow occur?
Yes
No
BMS returns control to the application program following the SEND MAP ACCUM command
A

BMS returns control to the application program at the OVERFLOW label.

1. Application program uses the ASSIGN PAGENUM command to find the current page number.
2. Application program moves page number and other data into the TRAILER map, and issues a SEND MAP ACCUM command for the trailer map.
3. Application program starts a page by transmitting a HEADER map (which specifies JUSTIFY=FIRST) using SEND MAP ACCUM ERASE.
4. Application program issues a SEND MAP ACCUM for the map that caused page overflow.

Figure 23. Overflow processing

The OVERFLOW condition is not raised by the SEND TEXT command. There is no simple way for the application program to gain control at the end of each page of text (it can be done using the SET option and the RETPAGE condition as discussed in "Returning mapped data to a program before output" on page 189).

The JUSTIFY, JUSFIRST, and JUSLAST options allow the application program to position a block of text on a particular line (JUSTIFY) or to position the block of text on the top (JUSFIRST) or bottom (JUSLAST) of the page.

The data areas named in the HEADER and TRAILER options have the following format:

```
| L | L | P | C |      | PNFLD |      |
                <————DATA————>
```

where:

LL is a halfword binary field containing the length of the header or trailer data. (The value does not include the 4 bytes of LL, P, and C characters.)

P is a one-byte field whose contents indicate whether page numbering is required or not. If the field contains a character other than a blank (X'40'), page numbering is required. (X'0C', X'15', X'17', X'26', and X'FF' are reserved and cannot be used).

The character specified is the character that is embedded in the header or trailer data in the positions (a maximum of 5) where the page number is to appear. If the field contains a blank, page numbering is not required.

```
/*  OUTPUT THE FIRST HEADER MAP FOR THE FIRST PAGE                */
      Move Customer data to symbolic map for ORDHEAD;
      SEND MAPSET('SAMPLE') MAP('ORDHEAD') ACCUM PAGING ERASE;
/*  ISSUE A HANDLE CONDITION OVERFLOW                             */
      HANDLE CONDITION OVERFLOW(OVLAB);
/*  MAIN PROCESSING LOOP. OUTPUT ORDER LINES UNTIL THE ENTIRE    */
/*  ORDER HAS BEEN DISPLAYED                                      */
 LOOP:
      DO UNTIL(ALLDONE);
        Move order line data into the symbolic map for ORDLINE;
        SEND MAPSET('SAMPLE') MAP('ORDLINE') ACCUM PAGING;
        Set ALLDONE to TRUE if this is the last order line;
      END;
/*  ALL ORDER LINES OUTPUT. OUTPUT FINAL TRAILER                  */
      PAGENO=PAGENO+1;
      Move PAGENO into symbolic map for ORDTRL;
      SEND MAPSET('SAMPLE') MAP('ORDTRL') ACCUM PAGING;
/*  TERMINATE LOGICAL MESSAGE WITH A SEND PAGE                    */
      SEND PAGE RETAIN;
      GOTO CONTINUE;
OVLAB:
/*  PAGE OVERFLOW PROCESSING. OUTPUT TRAILER MAP TO END           */
/*  THE CURRENT PAGE.                                             */
      ASSIGN PAGENUM(PAGENO);
      Move PAGENO into symbolic map for ORDTRL;
      SEND MAPSET('SAMPLE') MAP('ORDTRL') ACCUM PAGING;
/*  OUTPUT A HEADER MAP WITH THE ERASE OPTION TO START THE        */
/*  NEXT PAGE.                                                    */
      SEND MAPSET('SAMPLE') MAP('ORDHEAD') ACCUM PAGING ERASE;
/*  RE ISSUE A SEND MAP FOR THE ORDER LINE WHICH CAUSED           */
/*  PAGE OVERFLOW, AND RETURN TO THE MAIN LOOP                    */
      SEND MAPSET('SAMPLE') MAP('ORDLINE') ACCUM PAGING;
      GOTO LOOP;
/*  LOGICAL MESSAGE HAS BEEN DELIVERED. CONTINUE PROCESSING       */
 CONTINUE:
```

Figure 24. Example of paging

C is a reserved one-byte field.

PNFLD is the page number field. This field can be embedded anywhere in the header or trailer data in the required page number position. It can contain from 1 through 5 occurrences of the character specified by P. These characters will be replaced by the current page number, up to a maximum of 32,767, as a page is built. A SEND PAGE command will causes the page number to be reset to 1.

DATA is the header or trailer data to be placed at the beginning or end of each page of output. Embedded new-line characters (X'15') may be used to provide multiple heading or footing lines.

The following is a PL/I example of a valid header or trailer data area:

```
DCL
 1 HEADAREA,
  2 HEADLL FIXED BIN(15) INIT(14),
  2 HEADP CHAR(1) INIT('@'),
  2 HEADPAD CHAR(1),
  2 HEAD CHAR(14) INIT('PAGE NO. @@');
```

Cumulative processing and device controls

Device controls are handled as follows for each page of cumulative BMS output:

1. The ERASE, ERASEAUP, NLEOM, and FORMFEED options are honored if they are used on ANY of the BMS SEND commands which contributed to this page.
2. The most recent values of the CURSOR, ACTPARTN, FMHPARM, and MSR options are honored for this page.

3. The most recent value of the 3270 write control character (WCC) is honored for this page. The WCC is set by the ALARM, FREEKB, PRINT, FRSET, L40, L64, L80, and HONEOM options. Some (or all) of these options may be omitted from the most recent BMS SEND command which contributes to this page. BMS does **not** merge together the WCC options for all the BMS SEND commands contributing to this page. It is essential that all the required WCC options are specified on the last BMS SEND command for each page.

Cumulative processing and partitions

BMS handles page overflow on a partition basis, using the size of the current partition's presentation space as the page size.

It is possible for a CICS transaction to build a single logical message, directed to several partitions (all of which must be in the same partition set). The logical message is terminated in the normal manner by a BMS SEND PAGE command, and is purged by a PURGE statement.

If the ACCUM option is used, pages of maps or text are built on a partition basis.

Take care when using the ACCUM option with multiple partitions, especially if headers and trailers are to appear in different partitions. At any time, there is only one page overflow exit for all partitions. Avoid an infinite loop of page overflows. These drive the overflow exit, making header or trailer partitions overflow. You can avoid such loops by coding the IGNORE CONDITION OVERFLOW command.

All the partitions in a single logical message must have the same disposition (that is, they must be all TERMINAL, all PAGING, or all SET).

A program can perform cumulative mapping in one partition of a multiple partition logical message, and cumulative text in another.

A program cannot issue a SEND PARTNSET request while building a logical message.

Cumulative processing and logical device components

If logical device components are in use, BMS cumulative processing accumulates data separately for each logical device component. Page overflow occurs on a logical device component basis. Terminal operator paging commands operate on a logical device component basis. It is also possible to accumulate map data for one logical device component, and text data for a different logical device component.

All the logical device components participate in the same BMS logical message. This is terminated by a single SEND PAGE or PURGE MESSAGE command. All pages in all logical device components are deleted when the terminal operator purges the message.

Message routing

You use message routing to build a logical message and route it to one or more terminals. The message is scheduled, for each designated terminal, to be delivered as soon as the terminal is available to receive messages, or at a specified time. Terminal operators who receive the message use terminal operator paging commands to view it. A variety of operands on the ROUTE command allow you flexibility when specifying the message destinations.

A ROUTE command initiates a message routing operation. It is followed by SEND MAP, SEND TEXT, or SEND CONTROL commands to build the logical message to be routed. These commands must specify the ACCUM option, and usually also specify the PAGING option (they can specify the SET option, though this is unlikely). A SEND PAGE command terminates the logical message, and causes it to be routed. When individual logical messages are routed to a terminal, they are not necessarily retrieved by the terminal operator in the sequence in which they were issued. If a specific sequence of pages is required, the pages must be sent as one message.

The SEND MAP, SEND TEXT, or SEND CONTROL commands that build the message must specify the ACCUM option. Other SEND MAP or SEND TEXT commands can be interleaved with these commands to send messages to the terminal that initiated the transaction while the message to be routed is being built. This is useful if a screen oriented transaction is building data for a printer. The screen oriented transaction can use normal SEND MAP (without the ACCUM option) and RECEIVE MAP commands to communicate with the screen, and can simultaneously build a routed message for a printer, using SEND MAP ACCUM PAGING commands.

Another consideration of routing to different types of terminal is the handling of overflow conditions. This is discussed in "Routing and page overflow" on page 189.

The message routing facility of BMS is useful for developing message switching and broadcasting applications, and for interacting with a screen while collecting data for a printer. CICS provides a generalized message switching application program that uses the message routing facility of BMS (see the *CICS/MVS CICS-Supplied Transactions* manual for details). It is not possible to route a multiple partition or multiple LDC logical message. Any OUTPARTN, ACTPARTN, or LDC options on the BMS SEND commands are ignored while routing is in effect.

Defining a ROUTE list

The ROUTE command is used to define a route list. It has the following syntax:

```
ROUTE
[INTERVAL(hhmmss)[1]|TIME(hhmmss)]
[ERRTERM[(name)]]
[TITLE(data-area)]
[LIST(data-area)]
[OPCLASS(data-area)]
[REQID(name)]
[LDC(name)]
[NLEOM]

Conditions: INVERRTERM, INVLDC,
INVREQ, RTEFAIL, RTESOME

[1] INTERVAL(0) is the default
```

The options LIST and OPCLASS allow the designation of those terminals or logical units, or particular operators, to which the logical message is to be scheduled for delivery. Whether the logical message will actually be delivered (that is, received at the terminal) depends on many factors, such as availability of the terminal, or of a specific operator, within a certain time after the logical message is ready to be delivered.

The LIST option specifies a list of terminals and/or operators to receive the routed logical message. If no list is provided, the logical message will be scheduled for delivery to all terminals supported by BMS (unless the OPCLASS option is specified and has some effect). The message is only delivered to operators specified in the LIST if they remained signed on at the same terminal, as they were signed on at, when the ROUTE command was issued.

There is a limit to the number of terminals to which a message can be sent. The maximum cannot be defined because it is dependent on the other operands specified on the routing command, but the transaction will be abended with an abend code of ABMC if the limit is exceeded.

The OPCLASS option specifies the classes of operators to receive the routed logical message. OPCLASS can be used alone, or in conjunction with LIST.

The uses and format of the route list and of the information to be provided in the OPCLASS option are described in "Route list and operator class codes (LIST and OPCLASS)" on page 187.

The logical message can be delivered at a specified time (TIME option) or after a certain interval has elapsed (INTERVAL option); if neither option is specified, or if INTERVAL(0) is specified, the logical message will be delivered as soon as possible.

If a logical message is to be routed to more than one type of terminal, BMS builds a different logical message containing the appropriate device dependent data stream for each terminal type. Each message is stored on temporary storage until all terminals of this terminal type have received the message.

If a terminal is scheduled to receive a message but is not eligible, the message is stored until one of the following conditions occurs:

- A change in terminal status allows the message to be sent.
- A period specified by the PRGDLAY option of the DFHSIT macro has elapsed, causing the message to be deleted by BMS.
- The message is deleted by the destination terminal operator.

If a logical message is to be routed to terminals with alternate screen sizes (for example, the 3278), the choice of alternate or default screen size is made depending on the value specified using the SCRNSZE option of CEDA (or the SCRNSZE operand of DFHPCT TYPE=ENTRY) for the transaction issuing the ROUTE command. (See the *CICS/MVS Resource Definition (Online)* manual or the *CICS/MVS Resource Definition (Macro)* manual.)

If a ROUTE command followed by one or more BMS output commands is not terminated by a SEND PAGE command before a subsequent ROUTE command is issued, the INVREQ exceptional condition occurs. A ROUTE command may be issued immediately following another ROUTE command. In this case, the first ROUTE command is nullified, and the second determines the routing environment.

Any partition or LDC related options on the BMS SEND commands specifying the ACCUM option are ignored while routing is in effect.

If a message cannot be delivered within a certain time, it will be deleted (purged); the time is specified in the PRGDLAY (purge delay) operand of the DFHSIT macro. If the PRGDLAY operand is omitted, undelivered messages await delivery indefinitely. IF PRGDLAY is specified, an error message is generated whenever a message is purged. The destination of this error message depends on the ERRTERM specification in the ROUTE command as follows:

> If ERRTERM(*name*) is specified, the error message is sent to the named terminal.
>
> If ERRTERM is specified without a name, the error message is sent to the terminal associated with the task that sent the purged message.
>
> If ERRTERM is omitted, the error message is not sent.

In all cases, CICS lets the master terminal operator know how many messages have been deleted for each destination.

Disposition and message routing

A logical message can be built using either of two dispositions: PAGING or SET. The first BMS output command following the ROUTE command (with some exceptions noted below) determines the disposition of the logical message. Once established, the disposition must remain unchanged until the logical message is completed by a SEND PAGE command, or is deleted by a PURGE MESSAGE command. An output request specifying a disposition that is not in effect results in the INVREQ condition.

PAGING is the normal disposition and results in the logical message being written to temporary storage, and the terminal operator paging transaction being initiated for each terminal in the route list.

The SET option is rarely used in conjunction with routing. SET causes the logical message to be returned to the application program which is then responsible for its delivery, as discussed in "Returning mapped data to a program before output" on page 189.

Interleaving conversation with message routing

A task can converse with the terminal to which it is currently attached while it is building a logical message, for example for a printer. The attached terminal is known as the direct terminal; a terminal to which the message is to be routed is known as a routing terminal. If any RECEIVE MAP, RECEIVE PARTN, or RECEIVE commands are encountered while the message is being built, they are processed as usual.

The following rules apply to a direct terminal:

- TERMINAL must be specified or implied in any SEND command that is to go to the direct terminal.
- The ACCUM option with a disposition of TERMINAL is invalid and results in the INVREQ condition.
- The direct terminal may be included in the routing environment without impairing the ability to converse with it while under ROUTE. Data routed to the direct terminal will be delivered as though the ROUTE command had been issued from another terminal.

The following shows an example of a sequence of commands for a logical message, and summarizes briefly what action CICS takes in response to each.

```
SEND TEXT TERMINAL - Transmit to direct
     terminal.

ROUTE - Establish routing environment.

SEND MAP TERMINAL - Transmit to
     direct terminal.

RECEIVE MAP - Receive from direct
     terminal.

SEND TEXT PAGING ACCUM - First output
     command eligible for routing
     establishes disposition of
     PAGING.

SEND MAP TERMINAL - Transmit to
     direct terminal.

SEND TEXT SET(A) - Invalid request;
     routed logical message has
     already established a disposition
     of PAGING.

SEND TEXT PAGING ACCUM - Continue
     building routed logical message.

SEND MAP(Y) PAGING ACCUM - Invalid
     request; routed logical message
     cannot be built with both SEND
     TEXT and SEND MAP commands.

SEND MAP(Y) TERMINAL ACCUM - Invalid
     request; cannot issue SEND MAP
     ACCUM or SEND TEXT ACCUM command
     to direct terminal while
     building a routed logical
     message.

SEND TEXT PAGING ACCUM - Continue
     building routed logical message.

SEND PAGE - Complete and send
     logical message and terminate
     routing operation.

SEND TEXT TERMINAL - Send to
     direct terminal.
```

TITLE option of the ROUTE command

The title named in the TITLE option is displayed with the logical message identifier when the terminal operator page query command is entered (see the *CICS/MVS CICS-Supplied Transactions* manual). This title serves as an additional message identifier, displayed upon request with the message identifier, not on the logical message.

The value in the 2-byte length field preceding the title includes the bytes used for the length field. The length field and title, in total, may be up to 64 bytes long. For example:

```
|X'001A'|MONTHLYbINVENTORYbREPORT|

  2-byte          24-byte
 length title   field     field
```

Route list and operator class codes (LIST and OPCLASS)

The system programmer specifies the terminal or logical unit identifiers for all the terminals of the CICS system in the terminal control table (TCT). (For logical units with LDC support, LDC mnemonics are specified in the LDC table.) Also, an operator identifier must be specified for each operator, and up to 24 operator class codes (in the range 1 through 24) can be specified for particular operators, using the OPIDENT and OPCLASS operands, respectively, of the sign-on-table system macro (DFHSNT TYPE=ENTRY).

When an operator signs on at a terminal, CICS associates the operator and the optional class codes with that terminal until the operator signs off again. The application program can provide a route list in the LIST option to specify which terminals, or logical units, or operators are to receive the logical message; alternatively, or in addition, up to 24 operator class codes can be specified for use with a ROUTE operation, by using the OPCLASS option.

Before a logical message is delivered, all of the following conditions must be fulfilled:

- The terminal or logical unit must be supported by BMS and be operational.
- The logical message must be ready for delivery (TIME or INTERVAL options satisfied).
- The purge delay must not have expired.

Whether or not a logical message will be delivered at a specific terminal then depends on the use of the LIST and OPCLASS options, as follows:

- LIST and OPCLASS are omitted. All terminals will receive the message.
- LIST is specified but OPCLASS is omitted. The route list can contain three types of entry, each type having a different effect. All three types of entry can be included in the same list. The types of entry are:
  - Entries specifying a particular terminal (or logical unit) identifier but no operator identifier. Each specified terminal will receive the message.
  - Entries specifying a particular terminal (or logical unit) identifier and an operator identifier. Each specified terminal will receive the message if or when the specified operator is signed on at the terminal.
  - Entries specifying only an operator identifier. Each specified operator must be signed on at a terminal supported by BMS when the ROUTE command is issued; otherwise the route list entry for that operator is ignored (skipped). CICS will then schedule the message for delivery to each terminal at which a specified operator is signed on. If a particular operator is signed on at more than one terminal, CICS will schedule the message for delivery to the one whose entry appears first in the terminal control table. Each terminal for which the message is scheduled will then receive the message (when it is ready for delivery if the specified operator is still signed on at the terminal or when the operator signs on again.
- LIST is omitted but OPCLASS is specified. CICS will schedule the message for delivery to all terminals at which an operator having at least one of the specified operator class codes is signed on when the ROUTE command is issued. Each terminal for which the message is scheduled will then receive the message (when it is ready for delivery) if or when an operator (not necessarily the same one as before) having at least one of the specified operator class codes is signed on at the terminal.
- LIST and OPCLASS are both specified. The effect of the OPCLASS specification for the different types of route list entries is as follows:
  - Entries specifying no operator identifier. The effect is the same as if only the OPCLASS option were specified, but is restricted to those terminals (or logical units) specified in the route list. However, in this case, an operator with a matching operator class does not need to be signed on.
  - Entries specifying an operator identifier (and possibly a terminal or logical unit identifier). The OPCLASS specification is ignored for these route list entries, and the effect is the same as if only the LIST option were specified.

Route list format

The route list specified in the LIST option must conform to a fixed format. The list consists of 16-byte entries as follows:

Bytes	Contents
0-3	Terminal or logical unit identifier (4 characters, including trailing blanks), or blanks
4,5	LDC mnemonic (2 characters) for logical units with LDC support, or blanks
6-8	Operator identifier, or blanks
9	Status flag for the route entry
10-15	Reserved; must contain blanks

The end of the list is designated by a binary halfword initialized to −1.

The status flag (byte 9) indicates to the application program the status of the destination when the ROUTE command is issued. Upon return, the application program can investigate the status flag byte for each entry and take appropriate action. The status flag byte settings and their meanings are as follows:

ENTRY SKIPPED

A route list entry was excluded. If an entry has been excluded, another flag indicating why the entry was skipped may be on in the status byte. This second flag could be any of the other flags shown in the table. If the OPERATOR NOT SIGNED ON flag is on, only an operator identifier was specified in the route list entry and the specified operator was not signed on at any terminal. The settings are X'80' for ASM, 12-0-1-8 for COBOL, and 10000000 for PL/I.

INVALID TERMINAL IDENTIFIER

indicates that the terminal identifier specified in the route list entry does not have a corresponding entry in the terminal control table. This entry is also flagged as ENTRY SKIPPED. The settings are X'40' for ASM, no punches for COBOL, and 01000000 for PL/I.

TERMINAL NOT SUPPORTED UNDER BMS

indicates that the terminal identifier specified in the route list entry is for a type of terminal that is not supported under BMS; or the terminal table entry indicated that the terminal was not eligible for routing. This entry is also flagged as ENTRY SKIPPED. The settings are X'20' for ASM 11-0-1-8-9 for COBOL, and 00100000 for PL/I.

OPERATOR NOT SIGNED ON

indicates that the specified operator is not signed on. Any one of the following conditions causes this flag to be set:

- Both an operator identifier and a terminal identifier were specified, and the specified operator was not signed on at the terminal. This entry is not skipped.
- An operator identifier was specified without a terminal identifier, and the operator was not signed on at any terminal. This entry is also flagged as ENTRY SKIPPED.
- The OPCLASS option was specified with the ROUTE command and a terminal identifier was specified in the route list entry, but the operator signed on at the terminal did not have any of the specified operator classes. This entry is not skipped.

The settings are X'10' for ASM, 12-11-1-8-9 for COBOL, and 00010000 for PL/I.

OPERATOR SIGNED ON AT UNSUPPORTED TERMINAL

indicates that only an operator identifier was specified in the route list entry, and that operator was signed on a terminal not supported by BMS. This entry is also flagged as ENTRY SKIPPED. The unsupported terminal identifier is returned in that route list entry's terminal identifier field. The settings are X'08' for ASM, 12-8-9 for COBOL, and 00001000 for PL/I.

INVALID LDC MNEMONIC

indicates that one of the following situations exists:

- The LDC mnemonic specified in the route list does not appear in the LDC list associated with the TCT.
- The device type generated in the system LDC table for the specified or implied LDC mnemonic is not the same as the device type for the first LDC specified in the route environment.

The settings are X'04' for ASM, 12-4-9 for COBOL, and 00000100 for PL/I.

A symbolic storage definition of the user-supplied route list is available in the source library (or libraries) under the member name DFHURLDS. This definition can be used as an aid in building the route list, and if necessary, in testing the status flag byte for each entry upon return from a ROUTE command that refers to a list.

The list can be supplied in noncontiguous areas called segments, in which case every segment except the last is terminated with (at least) an 8-byte entry with contents as follows:

Bytes	Contents
0,1	ASM: binary halfword initialized to -2 COBOL: PIC S9(4) COMP VALUE -2. PL/I: DCL FIXED BIN(15) INIT(-2)
2,3	Reserved
4-7	Chain address to the first entry of the next segment

The last segment (that is, the end of the route list) ends with a binary halfword initialized to −1.

Routing and page overflow

The routing process builds a separate logical message containing the appropriate device dependent data stream for each terminal type mentioned in the route list. Because different types of terminal may have different page sizes, the overflow condition is likely to occur at different times in page building. BMS returns control to an overflow label in the application program, where the application program can determine by appropriate ASSIGN options which type of terminal caused the overflow, the current page number for that terminal type, and the total number of terminal types in the route list.

This is done using the ASSIGN command with either the DESCOUNT or PAGENUM options, as follows:

1. The ASSIGN DESTCOUNT command may be issued following a ROUTE command. It returns a count of the number of terminal types to receive the routed message. This count tells the application program how many logical messages will be built by BMS, and hence how many 'overflow control areas' the application program should allocate. These overflow control areas may be useful for the application program to remember, for example, the current page number for each terminal type. However, it is not necessary to use overflow control areas.
2. The ASSIGN DESTCOUNT command may be issued following page overflow to return the relative overflow control number of the terminal type that has encountered the overflow. This number indicates which overflow control area should be referenced, perhaps through one or more trailer maps.
3. The ASSIGN PAGENUM command returns the page number for the terminal type that has encountered the overflow.

Message switching transaction (CMSG)

CICS provides a message switching transaction (CMSG), which uses BMS text, routing, and paging. This transaction allows a terminal operator to send a text message to one or more other terminal operators. This transaction is discussed in the *CICS/MVS CICS-Supplied Transactions* manual.

Returning mapped data to a program before output

SET option

The SET option of the SEND MAP, SEND TEXT, and SEND CONTROL commands causes completed pages of a device dependent data stream to be returned to the application program, and sets a pointer to the address of a list of completed pages. The application program can use the SET option to:

1. Implement its own terminal operator paging scheme. It will thus save the returned pages in temporary storage, and subsequently retrieve them from temporary storage and send them to the terminal by a SEND TEXT MAPPED command.
2. Fill a screen with text data, and gain control when the screen is full. This can be done by issuing:
   a. A HANDLE CONDITION RETPAGE(label) command. BMS passes control to the specified label on the SEND TEXT ACCUM SET command which causes "page overflow".
   b. SEND TEXT ACCUM SET commands to send the text.
   c. A SEND PAGE SET command followed by a SEND TEXT MAPPED command in the code which handles the RETPAGE condition.
3. Modify the device dependent data stream returned by BMS. This is not recommended because the format of the data stream is not guaranteed to remain unchanged.

A single BMS command can generate more than one page of output; there may be more than one entry in the list for a given type of terminal. (Pages may be built for multiple terminal types by a single BMS command if routing is in effect.)

The data stream returned by BMS contains device-dependent data, so it is recommended that EXEC CICS SEND TEXT MAPPED be used to send the data to the device, rather than SEND TEXT NOEDIT or a terminal control SEND.

The entries for each type of terminal immediately follow one another in the list. Each entry contains a single byte terminal code (described in the next section) and a 3-byte address of a terminal input/output area (TIOA) containing a device dependent data stream plus header information. The layout of the TIOA is as follows:

Field	Contents
TIOASAA	8 bytes of storage accounting information
TIOATDL	2 bytes indicating length of data field TIOADBA, 2 bytes reserved
TIOADBA	Data field containing device-dependent data stream

The page list is terminated by an entry with a X'FF' value for the terminal code. The page list is reused following a SEND PAGE or PURGE MESSAGE command.

At this point, the page buffers addressed by the page list are on the user's storage chain, and are disassociated from BMS control. The application program should free these pages by means of FREEMAIN commands, when the pages are no longer needed. The address specified in the FREEMAIN command should be the address in the storage chain plus 8 bytes; that is, beginning from TIOATDL, not TIOASAA (which is contained in the first 8 bytes, and must not be FREEMAINed). See the TIOA layout earlier in this section for a description of these areas. The storage containing the page list should not be freed; the list will be reused by BMS to reduce processing time. The page list may be altered by the next BMS output command specifying the SET option.

Terminal code table

A terminal code table is established within BMS for reference in servicing BMS-supported terminals. There is one entry in this table for each terminal supported under BMS. A terminal code appears in the list of completed pages made available to the application program when the SET option is specified in a BMS output command. The code is available also in the EIBRCODE field of the EXEC interface block when the INVMPSZ condition occurs; for a description of this field, see Appendix A, "EXEC interface block" on page 339. This terminal type code is only of interest if BMS message routing is in effect. The codes are as follows:

Code	**Terminal or Logical Unit**
A	CRLP or TRMTYPE=TCAM terminals
B	Magnetic Tape
C	Sequential Disk
D	TWX Model 33/35
E	1050
F	2740-1,-2 (no buffer receive)
G	2741
H	2740-2 (with buffer receive)
I	2770
J	2780
K	3780
L[1]	3270 (40-character width)
M[2]	3270 (80-character width)
N	Not used
O	Not used
P[3]	3767/70 Interpreter LU
Q	2980 Models 1 and 2
R	2980 Model 4
S	Not used
T	Not used
U	3600 (3601) LU
V	3650 Host Convers (3653) LU
W	3650 Interpreter LU
X	3650 Host Convers (3270) LU
Y[1]	3770 Batch LU
Z	Not used

[1] Used also for 3770 and 3790 batch data interchange logical units, and LUTYPE4 logical units.

[2] Includes all LUTYPE2 and LUTYPE3 logical units.

[3] Used also for the 3790 full function logical unit and the SCS printer logical unit.

SEND TEXT MAPPED command

This command sends a page of a device dependent data stream previously built by BMS, and returned to the application program via the SET option. The command syntax is:

```
SEND TEXT
MAPPED
FROM(data-area)
[LENGTH(data-value)]
[PAGING|TERMINAL[WAIT]]
[REQID(name)]

Conditions: IGREQCD, IGREQID,
RETPAGE, TSIOERR, WRBRK
```

This command must only be used to output a device dependent data stream previously built by BMS. It references a 4-byte 'Page Control Area (PCA)' which BMS placed at the end of the device dependent data stream. The length of device dependent data stream set in the TIOATDL field of the page buffer returned by the SET option, does not include the PGA. The LENGTH option of the SEND TEXT MAPPED command should be set from this TIOATDL, and hence does not include the PGA. However, if the application program copies the page buffer returned by the SET option, it should include the PCA in the copied data.

SEND TEXT NOEDIT command

This command sends a page of a device dependent data stream built by the application program. The data stream cannot contain structured fields. This command differs from a terminal control SEND, as the data stream may be written to temporary storage and interfaced to the terminal operator paging transaction (specify the PAGING option). Also the device dependent data stream may be sent to a partition (specify the OUTPARTN option).

The syntax of this command is:

```
SEND TEXT
NOEDIT
FROM(data-area)
[LENGTH(data-value)]
[REQID(name)]
[OUTPARTN(name)]
[PAGING|TERMINAL[WAIT]]
[ERASE]
[PRINT]
[FREEKB]
[ALARM]

Conditions: IGREQCD, IGREQID,
INVREQ, RETPAGE, TSIOERR, WRBRK
```

The device dependent data stream in the FROM area cannot use structured fields.

If the OUTPARTN option is specified, the data stream is sent to the specified partition.

Chapter 3.2-5. BMS macro and command reference summary

This chapter shows the syntax of each BMS macro and command, separating the various operands and options into those appropriate to minimum, standard, and full function BMS. It describes the purpose and format of each macro and its operands, each command and its options, and points to related guidance information in the other BMS chapters. This chapter is for reference only; it contains no guidance information.

Map set, map, and field definition

This section describes the three map definition macros DFHMSD, DFHMDI, and DFHMDF. It shows the syntax of the macros, then lists and defines the operands.

Ensure that the names of maps, and names of fields within a map set (or within multiple map sets that are copied into one application program) are unique.

Map set definition macro (DFHMSD)

A DFHMSD macro defines a map set; it begins:

```
DFHMSD TYPE=MAP    (or TYPE=DSECT)
```

and ends:

```
DFHMSD TYPE=FINAL
```

The syntax of the DFHMSD macro is:

```
Minimum BMS

mapset DFHMSD
       TYPE={DSECT|MAP}
     [,MODE={IN|OUT|INOUT}]
     [,LANG={ASM|COBOL|PLI|RPG}]
     [,STORAGE=AUTO|,BASE=name]
     [,CTRL=([PRINT][,length]
       [,FREEKB][,ALARM][,FRSET])]
     [,EXTATT={NO|MAPONLY|YES}]
     [,COLOR={DEFAULT|color}]
     [,HILIGHT={OFF|BLINK|
       REVERSE|UNDERLINE}]
     [,PS={BASE|psid}]
     [,VALIDN=([MUSTFILL]
       [,MUSTENTER][,TRIGGER])]
     [,TERM=type|,SUFFIX=n]
     [,TIOAPFX={YES|NO}]
     [,MAPATTS=(attr1,attr2,...)]
     [,DSATTS=(attr1,attr2,...)]
     [,OUTLINE={BOX|([LEFT][,RIGHT]
       [,OVER][,UNDER])}]
     [,SOSI={NO|YES}]
     [,TRANSP={YES|NO}]

Standard BMS

     [,PARTN=(name[,ACTIVATE])]
     [,LDC=mnemonic]
     [,OBFMT={YES|NO}]
     [,HTAB=tab[,tab]...]
     [,VTAB=tab[,tab]...]
     [,DATA={FIELD|BLOCK}]
     [,FLDSEP=[char|X'hex-char']]
```

'mapset' is the 1- through 7-character name of the map set.

A DFHMSD macro contains one or more map definition macros, each of which contains one or more field definition macros.

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. The syntax of this macro is:

```
Minimum BMS

map     DFHMDI
     [,SIZE=(line,column)]
     [,CTRL=([PRINT][,length]
     [,FREEKB][,ALARM][,FRSET])]
     [,EXTATT={NO|MAPONLY|YES}]
     [,COLOR={DEFAULT|color}]
     [,HILIGHT={OFF|BLINK|
       REVERSE|UNDERLINE}]
     [,PS={BASE|psid}]
     [,VALIDN={[MUSTFILL]
       [,MUSTENTER][,TRIGGER]}]
     [,COLUMN=number]
     [,LINE=number]
     [,FIELDS=NO]
     [,MAPATTS=(attr1,attr2,...)]
     [,DSATTS=(attr1,attr2,...)]
     [,OUTLINE={BOX|([LEFT][,RIGHT]
       [,OVER][,UNDER])}]
     [,SOSI={NO|YES}]
     [,TRANSP={YES|NO}]
     [,JUSTIFY=BOTTOM]

Standard BMS

     [,PARTN=(name[,ACTIVATE])]
     [,OBFMT={YES|NO}]
     [,DATA={FIELD|BLOCK}]
     [,TIOAPFX={YES|NO}]
     [,FLDSEP=[char|X'hex-char']]

Full BMS

     [,COLUMN={number|NEXT|SAME}]
     [,LINE={number|NEXT|SAME}]
     [,JUSTIFY=([{LEFT|RIGHT}]
       [,{FIRST|LAST}])]
     [,HEADER=YES]
     [,TRAILER=YES]
```

'map' is the 1- through 7-character name 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.) For more information, see "Getting storage for a data structure" on page 151.

Field definition macro (DFHMDF)

The DFHMDF macro defines a field within a map defined by the previous DFHMDI macro. The syntax of this macro is:

```
Minimum BMS

[fld]  DFHMDF
     [,POS={number|(line,column)}]
     [,LENGTH=number]
     [,JUSTIFY=([{LEFT|RIGHT}]
       [,{BLANK|ZERO}])]
     [,INITIAL='char data'|
       XINIT=hex data]
     [,ATTRB=
     ([{ASKIP|PROT|UNPROT[,NUM]}]
       [,{BRT|NORM|DRK}]
       [,DET][,IC][,FSET])]
     [,COLOR={DEFAULT|color}]
     [,PS={BASE|psid}]
     [,HILIGHT={OFF|BLINK|REVERSE|
       UNDERLINE}]
     [,VALIDN=([MUSTFILL]
       [,MUSTENTER][,TRIGGER])]
     [,GRPNAME=group-name]
     [,OCCURS=number]
     [,PICIN='value']
     [,PICOUT='value']
     [,OUTLINE={BOX|([LEFT][,RIGHT]
       [,OVER][,UNDER])}]
     [,SOSI={NO|YES}]
     [,TRANSP={YES|NO}]
     [,CASE=MIXED]
```

'fld' is the 1-through 7-character name 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 input mapping operations is optimized if DFHMDF macros are arranged in numeric order of the POS operand.

You cannot define more than 1023 named fields for a COBOL or PL/I input/output map.

Ending a map set definition

A map set definition ends with a macro of the form:

```
[mapset] DFHMSD TYPE=FINAL
```

'mapset' is optional, but if used it must be the same as that on the DFHMSD macro that began the map set.

Map definition macro operand summary

This section lists and describes the operands of the three map definition macros, DFHMSD, DFHMDI, and DFHMDF.

ATTRB

is applicable only to fields to be displayed on a 3270 (it is ignored if sent to a non-3270 terminal) and specifies device dependent characteristics and attributes, such as the capability of a field to receive data or the intensity to be used when the field is output. If 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. Therefore, the ATTRB specification refers to all of the fields in a group as one field rather than as individual fields. See *An Introduction to the IBM 3270 Information Display System* for further information.

This operand applies only to 3270 data stream devices; it will be ignored for other devices, except that ATTRB=DRK is honored for the SCS Printer Logical Unit. It will also be ignored (except for ATTRB=DRK) if the NLEOM option is specified on the SEND MAP command for transmission to a 3270 printer. In particular, ATTRB=DRK should not be used as a method of protecting secure data on output on non-3270, non-SCS printer terminals. 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 and NUM are the only valid options; all others are ignored.

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. 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, and prevents entry of nonnumeric data if the Keyboard Numeric Lock feature is installed.

BRT specifies that a high intensity display of the field is required. By virtue of the 3270 attribute character bit assignments, a field specified as BRT is also potentially detectable. However, for the field to be recognized as detectable by BMS, DET must also be specified.

NORM specifies that the field intensity is to be normal.

DRK specifies that the field is nonprint/nondisplay. DRK cannot be specified if DET is specified.

DET specifies that the field is potentially detectable.

The first character of a 3270 detectable field must be one of the following:

`? > & blank`

If ? or >, the field is a selection field; if & or blank, the field is an attention field. (See the publication *An Introduction to the IBM 3270 Information Display System* for further details of detectable fields.)

A field for which BRT is specified is potentially detectable to the 3270, by virtue of the 3270 attribute character bit assignments, but is not recognized as such by BMS unless DET is also specified.

DET and DRK are mutually exclusive.

If DET is specified for a field on a map with MODE=IN, only one data byte is reserved for each input field. This byte is set to X'00', and remains unchanged if the field is not selected. If the field is selected the byte is set to X'FF'.

No other data is supplied, even if the field is a selection field and the ENTER key has been pressed.

If the data in a detectable field is required, all of the following conditions must be fulfilled:

1. The field must begin with one of the following characters:

   `? > & blank`

   and DET must be specified in the output map.
2. The ENTER key (or some other attention key) must be pressed after the field has been selected, although the ENTER key is not required for detectable fields beginning with a & or a blank.
3. DET must not be specified for the field in the input map. DET must, however, be specified in the output map. See "Chapter 3.2-2. Minimum function BMS" on page 139 for more information on BMS support of the light pen.

IC specifies that the cursor is to be placed in the first position 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.

This option 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 should 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.

Either of two sets of defaults may apply when a field to be displayed on a 3270 is being defined but not all parameters are specified. If no ATTRB parameters are specified, ASKIP and NORM are assumed. If any parameter is specified, UNPROT and NORM are assumed for that field unless overridden by a specified parameter.

BASE = name

specifies that the same storage base will be 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 will also overlay one another.

This operand is not valid for assembler language programs, and cannot be used when STORAGE = AUTO has been specified.

For example, assume that the following macros are used to generate symbolic description maps for two map sets:

```
MAPSET1 DFHMSD TYPE=DSECT,
        TERM=2780,LANG=COBOL,
        BASE=DATAREA1,MODE=IN

MAPSET2 DFHMSD TYPE=DSECT,
        TERM=3270,LANG=COBOL,
        BASE=DATAREA1,MODE=OUT
```

The symbolic description maps of this example might be referred to in a COBOL application program as follows:

```
LINKAGE SECTION.
01 DFHBLLDS COPY DFHBLLDS.
   .
   .
   02 TIOABAR PIC S9(8) COMP.
   02 MAPBASE1 PIC S9(8) COMP.
   .
   .
01 DFHTIOA COPY DFHTIOA.
01 DATAREA1 PIC X(1920).
01 name COPY MAPSET1.
01 name COPY MAPSET2.
   .
   .
EXEC CICS GETMAIN LENGTH(1000)
          SET(MAPBASE) INITIMG(0)
          END-EXEC
```

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.

The GETMAIN command should specify enough storage to contain the largest map of all those that share the same storage base. In this case, the programmer has decided that the largest map requires 1000 bytes of storage.

In PL/I application programs, the name specified in the BASE operand is used as the name of the pointer variable on which the symbolic description map is based. If this operand is omitted, the default name (BMSMAPBR) is used for the pointer variable. The PL/I programmer is responsible for establishing addressability for the based structures.

Note: The BASE operand is also described under "Getting storage for a data structure" on page 151.

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.

This should be specified if a field is known to contain lowercase latin characters but may be displayed on a Katakana display. It should not be specified if the field may contain valid Katakana characters.

COLOR

indicates the individual color, or the default color for the map set (where applicable). This is overridden by the COLOR operand of the DFHMDI macro, which is in turn overridden by the COLOR operand of the DFHMDF macro.

The valid colors are blue, red, pink, green, turquoise, yellow, and neutral.

If COLOR is specified when EXTATT=NO, a warning is issued and the option ignored. If COLOR is specified, but EXTATT is not, EXTATT=MAPONLY will be assumed.

The COLOR operand is ignored unless the terminal supports color, as indicated by the FEATURE operand of the DFHTCT TYPE=TERMINAL system macro.

COLUMN

specifies the column in a line at which the map is to be placed, that is, it establishes the left or right map margin. The JUSTIFY operand of the DFHMDI macro controls whether map and page margin selection and column counting are to be from the left or right side of the page. The columns between the specified map margin and the page margin are not available for subsequent use on the page for any lines included in the map.

number is the column from the left or right page margin where the left or right map margin is to be established.

NEXT indicates that the left or right map margin is to be placed in the next available column from the left or right on the current line.

SAME indicates that the left or right map margin is to be established in the same column as the last nonheader or nontrailer map used that specified COLUMN=number and the same JUSTIFY parameters as this macro.

For input operations, the map will be positioned either at the extreme left hand or right hand side depending on whether JUSTIFY=LEFT or RIGHT has been specified.

See "Map positioning" on page 177 for a detailed description of BMS map positioning.

CTRL

defines characteristics of IBM 3270 terminals. The CTRL operand on the DFHMSD macro is overridden by the CTRL option on the DFHMDI macro, which is in turn overridden by the ALARM, FREEKB and so on, options on the SEND MAP command.

If CTRL is used with cumulative BMS paging (that is, the ACCUM option is used on the BMS SEND MAP commands), it must be specified on the last (or only) map of a page, unless it is overridden by the ALARM, FREEKB and so on, options on the SEND MAP command.

PRINT must be specified if the printer is to be started; if omitted, the data is sent to the printer buffer but is not printed. This operand is ignored if the map set is used with 3270 displays without the Printer Adapter feature.

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 this option is omitted, BMS will set the line length from the TCT page size. This is further discussed under "Printed output" on page 156.

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 3270 audible alarm. For non-3270 VTAM terminals it sets the alarm flag in the FMH. (This feature is not supported by interactive and batch logical units.)

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.

DATA

specifies the format of the data.

FIELD specifies that the data is passed as contiguous fields each field having the format:

```
|LL|A|data field|
```

"LL" is two bytes specifying the length of the data as input from the terminal (these two bytes are ignored in output processing). "A" is a byte into which the programmer can place an attribute to override that specified in the map used to process this data (see copy book DFHBMSCA in "BMS related constants" on page 205).

If you specify EXTATT=YES, the field will have the form

```
|LL|A|C|P|H|V|data field|
```

where C, P, H, and V are the color, program symbol, highlight, and validation attribute bytes, respectively. See "Chapter 3.2-2. Minimum function BMS" on page 139 for further information on field data.

BLOCK specifies that the data is passed as a continuous stream in the following format:

```
|A|data field|space|
```

This stream is processed as line segments of the length specified in the map used to process the data. The data is in the form that it appears on the terminal; that is, it contains data fields and interspersed blanks corresponding to any spaces that are to appear between the fields on output. EXTATT=YES cannot be used if DATA=BLOCK is specified.

Block data is further discussed in "Chapter 3.2-3. Standard function BMS" on page 159. Its use is not recommended.

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 should also be included in MAPATTS.

EXTATT
this operand is supported for compatibility with previous releases. For new maps, the operands DSATTS and MAPATTS should be used 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
specifies whether or not the map contains 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.

FLDSEP
can be up to 4 characters indicating the field separator sequence for input from non-3270 devices. Input from non-3270 devices can be entered as a single string of data with the field separator sequence delimiting fields. The data between the field separators is moved to the input fields in the map in order.

GRPNAME
is the 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 be also specified to ensure the fields follow each other. All the DFHMDF macros defining the fields of a group must be placed together, and in the correct order (upward numeric order of the POS operand).

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.

Field groups are described under "Field groups" on page 145.

HEADER
allows the map to be used during page building without terminating the overflow condition (see "Floating maps, header and trailer maps" on page 176 for further details). This operand may be specified for more than one map in a map set.

HILIGHT
specifies the default highlighting attribute for all fields in all maps in a map set. This is overridden by the HILIGHT operand of the DFHMDI, which is in turn overridden by the HILIGHT operand of the DFHMDF.

OFF is the default and indicates that no highlighting is used.

BLINK specifies that the field must blink.

REVERSE specifies that the character or field is displayed in reverse video, for example, on a 3278, black characters on a green background.

UNDERLINE specifies that a field is underlined.

If HILIGHT is specified when EXTATT=NO, a warning is issued and the option ignored. If HILIGHT is specified, but EXTATT is not, EXTATT=MAPONLY will be assumed.

The HILIGHT operand is ignored unless the terminal supports highlighting, as indicated by the FEATURE operand of the DFHTCT TYPE=TERMINAL system macro.

HTAB
specifies one or more tab positions for use with interactive and batch logical units and SCS printers having horizontal forms control.

INITIAL (or XINIT)
specifies constant or default data for an output field. INITIAL is used to specify data in character form; XINIT is used to specify data in hexadecimal form. INITIAL and XINIT are mutually exclusive.

For fields with the DET attribute, initial data that begins with one of the following characters:

```
? > & blank
```

should be supplied.

The number of characters that can be specified in the INITIAL operand is restricted to the continuation limitation of the assembler to be used or to the value specified in the LENGTH operand (whichever is the smaller).

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 will be padded on the right with blanks. For example, XINIT=C1C2 might result in an initial field of 'AB '.

If hexadecimal data is specified that corresponds with line or format control characters, the results will be unpredictable. The XINIT operand should therefore be used with care.

JUSTIFY

There are two uses for the operand. On DFHMDI, you use it to specify the position of the map on the page. On DFHMDF, it specifies the position of a field within a BMS map.

These are the keywords you can specify on DFHMDI:

LEFT specifies that the map is to be positioned starting at the specified column from the left margin on the specified line.

RIGHT specifies that the map is to be positioned starting at the specified column from the right margin on the specified line.

FIRST specifies that the map is to be positioned as the first map on a new page. Any partially formatted page from preceding BMS commands is considered to be complete. This operand can be specified for only one map per page.

LAST indicates that the map is to be positioned at the bottom of the current page. This operand can be specified for multiple maps to be placed on one page. However, maps other than the first map for which it is specified must be able to be positioned horizontally without requiring that more lines be used.

BOTTOM for a SEND MAP ACCUM command has the same effect as LAST, above. For a SEND MAP command (without ACCUM) and a RECEIVE MAP command, JUSTIFY=BOTTOM will position the map at the bottom of the screen if the number of lines in the map is specified in the SIZE operand. No account will be taken of trailer maps in the map set. 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. JUSTIFY=BOTTOM is ignored if the number of lines is not specified as well. If JUSTIFY=BOTTOM and LINE are both specified, the value specified in LINE will be ignored.

LEFT and RIGHT are mutually exclusive, as are FIRST and LAST. If neither FIRST nor LAST is specified, the data is mapped at the next available position as determined by other parameters of the map definition and the current mapping operation. FIRST and LAST are ignored unless ACCUM is specified on SEND MAP commands; otherwise only one map is placed on each page.

See "Map positioning" on page 177 for a more detailed description.

The JUSTIFY operand on DFHMDF specifies the field justifications for input operations. This operand is ignored for TCAM-supported 3600 and 3790, and for VTAM-supported 3600, 3650, and 3790 terminals, because input mapping is not available.

These are the keywords you can specify on DFHMDF:

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 parameters are specified 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 keyed as input will only overwrite equivalent length existing data; surplus existing data will remain in the field and could cause unexpected interpretation of the new data.

LANG

specifies the source language of the application programs into which the symbolic description maps in the map set will be copied. This option need only be coded for DFHMSD TYPE=DSECT. 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.

LDC
specifies the code to be used by CICS to determine the logical device mnemonic that is to be used for a BMS output operation and transmitted in the function management header to the logical unit if no LDC operand has been specified on any previous BMS output in the logical message. This operand is used only for TCAM and VTAM-supported 3600 terminals, and batch logical units. For more information see "Logical device components" on page 169.

LENGTH
specifies the length (1 through 256 bytes) of the field. This specified length should be the maximum length required for application program data to be entered into the field; it should not include the one-byte attribute indicator appended to the field by CICS 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. A length of zero may be specified only if the label (field name) is omitted from the DFHMDF; that is, the field is not part of the application data structure, and the application program cannot modify the attributes of the field. A field with zero length may be used to delimit an input field on a map.

The map dimensions specified in the SIZE operand of the DFHMDI macro instruction defining a map may be smaller than the actual page size or screen size as defined for the terminal. The LENGTH specification in a DFHMDF macro cannot cause the map-defined boundary on the same line to be exceeded. That is, the length declared for a field cannot exceed the number of positions available from the starting position of the field to the final position of the map-defined line. For example, given an 80-position page line, the following map definition and field definition are valid:

```
DFHMDI SIZE=(2,40),...
DFHMDF POS=22,LENGTH=17,...
```

but the following definitions are not acceptable:

```
DFHMDI  SIZE=(2,40),...
DFHMDF  POS=22,LENGTH=30,...
```

LINE
specifies the starting line on a page in which data for a map is to be formatted.

number is a value from 1 to 240, specifying a starting line number. A request to map data on a line and column that has been formatted in response to a preceding BMS command causes the current page to be treated as though complete. The new data is formatted at the requested line and column on a new page.

NEXT specifies that formatting of data is to begin on the next available completely empty line. If LINE=NEXT is specified in the DFHMDI macro, it is ignored for input operations and LINE=1 is assumed.

SAME specifies that formatting of data is to begin on the same line as that used for a preceding BMS command. If COLUMN=NEXT is specified in the DFHMDI macro, it is ignored for input operations and COLUMN=1 is assumed. If the data does not fit on the same line, it is placed on the next available completely-empty line.

See "Map positioning" on page 177 for a detailed description of map positioning.

MAPATTS
specifies the attribute types to be included in the physical map. These types can be one or more of the following: COLOR, HILIGHT, 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 these values will be deduced from operands already specified in the DFHMSD and DFHMDI macros. For example, if COLOR=BLUE has been specified, MAPATTS=COLOR will be assumed.

MODE
specifies whether the map is to be used for input, output, or both.

OBFMT
specifies whether outboard formatting is to be used. This operand is available only for 3650 logical units, or for an 8100 series processor running DPS Release 2 and defined to CICS as an LUTYPE2 logical unit. For more information, see "Chapter 3.2-3. Standard function BMS" on page 159.

The OBFMT option on DFHMSD is overridden by the OBFMT option on DFHMDI.

YES specifies that all maps within this map set can be used in outboard formatting, except those for which OBFMT=NO is specified in the DFHMDI macro.

NO specifies that no maps within this map set can be used in outboard formatting, except those for which OBFMT=YES is specified in DFHMDI.

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.

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. If this operand is omitted, a value of 1 is assumed.

PARTN
specifies the default partition to be associated with maps in this map set. If the ACTIVATE option is specified, the specified partition will also be activated when maps in this map set are output to a terminal which supports partitions. This option is overridden by the PARTN option of the DFHMDI macro, which is in turn overridden by any OUTPARTN and/or ACTPARTN option on the SEND MAP command, or the INPARTN option on a RECEIVE MAP command.

The PARTN option is ignored if the target terminal does not support partitions, or if there is no partition set associated with the transaction.

PICIN (COBOL and PL/I only)
specifies a picture to be applied to an input field in an IN or INOUT map; this picture serves as an editing specification which is passed to the application program, thus permitting the user to exploit the editing capabilities of COBOL or PL/I. 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 PICIN or PICOUT is not coded for the field definition, a character definition of the field is automatically generated regardless of other operands that are coded, such as ATTRB=NUM.

As an example, assume 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) COMP.
    02  F1A     PIC X.
    02  FILLER REDEFINES F1A.
     03  F1F    PIC X.
    02  F1I     PIC X(30).
    02  FILLER  PIC X.
    02  F2L     PIC S9(4) COMP.
    02  F2A     PIC X.
    02  FILLER REDEFINES F2A.
     03  F2F    PIC X.
    02  F2I     PIC X(10).
    02  FILLER  PIC X.
    02  F3L     PIC S9(4) COMP.
    02  F3A     PIC X.
    02  FILLER REDEFINES F3A.
     03  F3F    PIC X.
    02  F3I     PIC 9999V99.
    02  FILLER  PIC X.
01  MAPO REDEFINES MAPI.
    02  FILLER  PIC X(3).
    02  F1O     PIC X(30).
    02  FILLER  PIC X.
    02  FILLER  PIC X(3).
    02  F2O     PIC $$$,$$0.00.
    02  FILLER  PIC X.
    02  FILLER  PIC X(3).
    02  F3O     PIC ZZ9.99.
    02  FILLER  PIC X.
```

Note: The valid picture values for COBOL maps are:

A P S V X 9 / and (

The valid picture values for PL/I maps are:

A B E F G H I K M P R S T V
X Y and Z

1 2 3 6 7 8 9 / + - , . *
$ and (

See the appropriate language reference manual for the correct syntax of the PICTURE attribute.

PICOUT (COBOL and PL/I 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 maps are:

```
A B E P S V X Z 0 9 , . + - $
CR DB / and (
```

The valid picture values for PL/I maps are:

```
A B E F G H I K M P R S T V
X Y and Z

1 2 3 6 7 8 9 / + - , . * $
CR DB and (
```

See the appropriate language reference manual for the correct syntax of the PICTURE attribute.

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 dependent on DFHMDI parameters as well.

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 the 3270. 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 should point to the last character position in the previous field in the group.

When a position number is specified which represents the last character position in the 3270, two special rules apply:

- The IC attribute should not be coded. The cursor may 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 DATA=ONLY specification, 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 is overridden by the PS operand of the DFHMDI macro, which is in turn overridden by the PS operand of the DFHMDF macro.

BASE specifies that the base symbol set is to be used.

psid specifies a single EBCDIC character, or a hexadecimal code of the form X'nn', that identifies the set of programmed symbols to be used.

If PS is specified when EXTATT=NO, a warning is issued and the option ignored. If PS is specified, but EXTATT is not, EXTATT=MAPONLY will be assumed.

The PS operand is ignored unless the terminal supports programmed symbols, as indicated by the FEATURE operand of the DFHTCT TYPE=TERMINAL system macro.

SIZE

specifies the size of a map.

line is a value from 1 through 240, specifying the depth of a map as a number of lines.

column is a value from 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 referred to in a SEND MAP command with the ACCUM option.
- The map is to be used when referring to input data from other than a 3270 terminal in a RECEIVE MAP command.

SOSI

indicates 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 provided 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 upon the language in which application programs are written, as follows:

assembler language specifies that individual maps within a map set are to occupy separate areas of storage instead of overlaying one another.

COBOL 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.

PL/I 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 BASED.

You cannot specify both BASE=name and STORAGE=AUTO for the same map set. For more information, see "Getting storage for a data structure" on page 151. If STORAGE=AUTO is specified and TIOAPFX is not, TIOAPFX=YES is assumed.

SUFFIX
specifies a one character user-defined device dependent suffix for this map set, as an alternative to a suffix generated by the TERM operand. The suffix specified by this option should match the value of the ALTSFX option of the DFHTCT TYPE=TERMINAL macro. Use a numeric value to avoid conflict with suffixes generated by the TERM operand.

Map set suffixing is discussed under "Map set suffixing" on page 146.

TERM
specifies the type of terminal or logical unit (LU) associated with the map set. 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 the table.

In addition, note the following:

For TCAM-connected terminals (other than 3270 or SNA devices), use either CRLP or ALL; for TCAM-connected 3270s or SNA devices, select the appropriate parameter in the normal way.

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 will be applied. For example, some terminals are limited to 480 bytes, others to 1920 bytes; the 3604 is limited to six lines of 40 characters each. Within these guidelines, use of ALL can offer important advantages. Because an assembly run is required for each map generation, the use of ALL, indicating that one map is to be used for more than one terminal, can result in significant time and storage savings.

However, better run time performance for maps used by single terminal types will be achieved if the terminal type (rather than ALL) is specified. Alternatively, BMS support for device dependent map sets can be bypassed by specifying NODDS in the BMS operand of the DFHSIT system macro. For more information, see the *CICS/MVS Resource Definition (Macro)* manual.

TIOAPFX
specifies whether BMS should include a filler in the symbolic description maps to allow for the unused TIOA prefix.

TYPE	Suffix
CRLP[1]	A
TAPE	B
DISK	C
TWX	D
1050	E
2740	F
2741	G
2770	I
2780	J
3780	K
3270-1 (40-column)	L
3270-2 (80-column)	M
INTLU/3767/3770I/SCS[2]	P
2980	Q
2980-4	R
<u>3270</u>[3]	blank
3601	U
3653[4]	V
3650UP[5]	W
3650/3270[6]	X
BCHLU/3770B[7]	Y
ALL (all the above)	blank

[1] Card-reader-in/line-printer-out

[2] All interactive LUs including 3790 full function LU and SCS printer LUs (3270 and 3790).

[3] Default if TERM omitted. Same as ALL; used when no need to distinguish between models.

[4] Plus host-conv (3653) LU.

[5] Plus interpreter LU.

[6] Plus host-conv (3270) LU.

[7] Plus all batch and BDI LUs.

YES specifies that the filler should be included in the symbolic description maps. If TIOAPFX=YES is specified, all maps within the map set have the filler, except when TIOAPFX=NO is specified on the DFHMDI macro. TIOAPFX=YES should **always** be used for command level application programs.

NO is the default and specifies that the filler is not to be included. The filler may still be included for a map if TIOAPFX=YES is specified on DFHMDI.

TRAILER
allows the map to be used during page building without terminating the overflow condition (see "Floating maps, header and trailer maps" on page 176). This operand may be specified for more than one map in a map set. If a trailer map is used

other than in the overflow environment, the space normally reserved for overflow trailer maps is not reserved while mapping the trailer map.

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.

TYPE
specifies the type of map to be generated using the definition. Both types of map must be generated before the map set can be used by an application program. If aligned symbolic description maps are required, you must ensure that you specify SYSPARM=ADSECT and SYSPARM=AMAP when you assemble the symbolic and physical maps respectively.

DSECT specifies that a symbolic description map is to be generated. Symbolic description maps must be copied into the source program before it is translated and compiled.

MAP specifies that a physical map is to be generated. Physical maps must be assembled or compiled, link edited, and cataloged in the CICS program library before an application program can use them.

If both map and DSECT are to be generated in the same job, the SYSPARM option can be used in the assembler job execution step, as described in the *CICS/MVS Installation Guide.*

VALIDN
specifies that validation is to be used on an 8775 terminal. This is overridden by the VALIDN operand of the DFHMDI macro, which is in turn overridden 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, will raise the inhibit input condition.

MUSTENTER specifies that data must be entered into the field, though need not fill it. An attempt to move the cursor from an empty field will raise the inhibit input condition.

TRIGGER specifies that this field is a trigger field. Trigger fields are discussed in "Chapter 3.2-3. Standard function BMS" on page 159.

The VALIDN operand is ignored unless the terminal supports validation, as indicated by the FEATURE operand of the DFHTCT TYPE=TERMINAL system macro.

VTAB
specifies one or more tab positions for use with interactive and batch logical units and SCS printers having vertical forms control.

XINIT
See INITIAL, earlier in the list.

Partition set definition

Partitions are defined by coding the macros DFHPSD (partition set definition) and DFHPDI (partition definition). Each partition definition must be part of a partition set definition.

Partition set definition macro (DFHPSD)

This section shows the syntax of the DFHPSD macro, which defines a partition set. Each partition set definition contains a single DFHPSD macro, followed by one or more DFHPDI macros, and ending with a DFHPSD TYPE=FINAL macro.

The format of the DFHPSD macro is:

```
partnset DFHPSD
          [SUFFIX=user-suffix]
         [,ALTSCRN=(lines,columns)]
         [,CHARSZE=(vpels,hpels)]
```

'partnset' is a 1 through 6 character partition set name.

Partition definition macro (DFHPDI)

A partition set contains one or more partitions. Each partition is defined by coding a partition definition macro. The format of the macro is:

```
[partn] DFHPDI
          VIEWPOS=(lines,columns)
         ,VIEWSZE=(lines,columns)
        [,BUFSZE=(lines,columns)]
        [,CHARSZE=(vpels,hpels)]
        [,MAPSFX=mapset-suffix]
        [,ATTRB=ERROR]
```

'partn' is a 1 or 2 character partition name. It allows you to refer to the partition in your application programs.

Every partition in a partition set must have a different name. Only the error partition can be unnamed (see ATTRB=ERROR operand).

Ending a partition set definition

This macro ends a partition set definition. Its format is:

```
[partnset] DFHPSD TYPE=FINAL
```

The partnset name (if specified) must match that specified on the DFHPSD macro that started the partition set definition.

Partition definition macro operand summary

The operands have the following meanings:

ALTSCRN(lines,columns)
specifies the size, in characters, of the usable area of the target terminal. This is normally the same as the ALTSCRN operand of the DFHTCT TYPE=TERMINAL entry for the terminal. You use ALTSCRN to ensure that the viewports of partitions within a partition set fit into the usable area of the screen.

ATTRB=ERROR
specifies that error messages are to be directed to this partition whenever possible. The partition is cleared before an error message is displayed. Attributes specified on the ERRATT option of the DFHTCT TYPE=TERMINAL macro will be honored, but the LASTLINE option will be ignored.

BUFSZE=(lines,columns)
specifies the size of the presentation space for the partition. Device limitations mean that the "columns" value must be equal to the "columns" value specified by the VIEWSZE operand. The "lines" value can be greater than or, by default, equal to the value specified by the VIEWSZE operand. A greater lines value implies that the target terminal supports vertical scrolling. The default value of "lines" is the same as the value specified by the VIEWSZE operand.

CHARSZE(vpels,hpels)
specifies the size of the character cell to be reserved for each character displayed in a partition. This operand can be specified on the DFHPSD macro alone, or on both the DFHPSD and DFHPDI macros. Values specified on the DFHPDI macro override, for that partition only, the default values specified on the DFHPSD macro. For guidance on using CHARSZE, see "Character cells in partitions" on page 168.

MAPSFX=mapset-suffix
is the partition's 1-character map set suffix. BMS uses the suffix to select map set versions in the same way as the ALTSFX operand of the DFHTCT TYPE=TERMINAL macro. If this operand is omitted, a suffix L is assumed if the "columns" value of the BUFSZE operand is less than or equal to forty; otherwise M is assumed. Suffixing in general is discussed more fully in "Map set suffixing" on page 146.

SUFFIX=user-suffix
is a 1-character user suffix for this version of the partition set. It allows different versions of a partition set to be associated with different terminals. When the partition set is to be loaded, CICS looks for a version whose suffix matches the ALTSFX operand specified on the DFHTCT TYPE=TERMINAL macro. If it cannot find the correct partition set version, it loads a version with the default suffix (M or L). If it cannot find a suffixed version either, it loads an unsuffixed one. If it cannot find this, it raises the APCT abend.

VIEWPOS=(lines,columns)
specifies the position of the top left hand corner of this partition's viewport. You specify the position in numbers of lines and numbers of columns.

The DFHPDI macro checks that viewports do not overlap. If the ALTSCRN operand of the DFHPSD macro has been coded, DFHPDI also checks that all viewports fit within the usable area of the terminal screen.

VIEWSZE=(lines,columns)
specifies the size, in lines and columns, of the partition's viewport. The DFHPDI macro checks that viewports do not overlap. If you code the ALTSCRN operand of the DFHPSD macro, DFHPDI will check that the partitions all fit within the usable area of the terminal screen.

Note: The information given here on positioning viewports is necessarily brief. For more information, consult the component description for the device you are using.

BMS related constants

Standard attribute and printer control character list (DFHBMSCA)

The standard list DFHBMSCA simplifies the provision of field attributes and printer control characters. The list is obtained by copying copy book DFHBMSCA into the application program. The symbolic names for the various combinations of attributes and control characters are given in Table 3 on page 206. Combinations other than shown must be generated separately; a bit map to help you do this is given in Figure 25 on page 207.

The value of an attribute constant can be determined by reference to the *3274 Control Unit Reference Summary.*

For assembler language users, the list consists of a set of EQU statements. For COBOL users, the list consists of a set of 01 statements that can be copied into the working storage section. For PL/I users, the list consists of

DECLARE statements defining elementary character variables.

The symbolic name DFHDFT must be used 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, the symbolic names DFHDFCOL, DFHBASE, or DFHDFHI should be used.

Table 3. Standard list DFHBMSCA

Constant	Meaning
DFHBMPEM	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
DFHBMASF	Autoskip and MDT set
DFHBMASB	Autoskip and bright
DFHBMPSO	Shift out value X'0E'
DFHBMPSI	Shift in value X'0F'
DFHBMEOF	Field erased
DFHBMDET	Field detected
DFHSA[1]	Set attribute (SA) order
DFHERROR	Error code
DFHCOLOR[1]	Color
DFHPS[1]	Programmed symbols
DFHHLT[1]	Highlight
DFH3270[1]	Base 3270 field attribute
DFHVAL	Validation
DFHOUTLN	Field outlining attribute code
DFHBKTRN	Background transparency 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
DFHBASE[1]	Base programmed symbols
DFHDFHI[1]	Normal
DFHBLINK	Blink
DFHREVRS	Reverse video
DFHUNDLN	Underscore
DFHMFIL[2]	Mandatory fill
DFHMENT[2]	Mandatory enter
DFHMFE	Mandatory fill and mandatory enter
DFHMT	Trigger
DFHMFT	Mandatory fill and trigger
DFHMET	Mandatory enter and trigger
DFHUNNOD	Unprotected, nondisplay, nonprint, nondetectable, MDT
DFHUNIMD	Unprotected, intensify, light pen detectable, MDT
DFHUNNUM	Unprotected, numeric, MDT
DFHUNINT	Unprotected, numeric, intensify, light pen detectable, MDT
DFHUNNON	Unprotected, numeric, nondisplay, nonprint, nondetectable, MDT
DFHPROTI	Protected, intensify, light pen detectable
DFHPROTN	Protected, nondisplay, nonprint, nondetectable
DFHMFET	Mandatory fill and mandatory enter and trigger
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
DFHTRANS	Background transparency
DFHOPAQ	No background transparency

Notes:

1. For text processing only. Use for constructing embedded set attribute orders in user text.
2. Cannot be used in set attribute orders.

prot	a/n	hi	spd	ndp	mdt	ebcd	asci	char
U						40	20	b (blank)
U					Y	C1	41	A
U			Y			C4	44	D
U			Y		Y	C5	45	E
U		H	Y			C8	48	H
U		H	Y		Y	C9	49	I
U				Y		4C	3C	<
U				Y	Y	4D	28	(
U	N					50	26	&
U	N				Y	D1	4A	J
U	N		Y			D4	4D	M
U	N		Y		Y	D5	4E	N
U	N	H	Y			D8	51	Q
U	N	H	Y		Y	D9	52	R
U	N			Y		5C	2A	*
U	N			Y	Y	5D	29	)
P						60	2D	- (hyphen)
P					Y	61	2F	/
P			Y			E4	55	U
P			Y		Y	E5	56	V
P		H	Y			E8	59	Y
P		H	Y		Y	E9	5A	Z
P				Y		6C	25	%
P				Y	Y	6D	5F	_ (underscore)
P	S					F0	30	0
P	S				Y	F1	31	1
P	S		Y			F4	34	4
P	S		Y		Y	F5	35	5
P	S	H	Y			F8	38	8
P	S	H	Y		Y	F9	39	9
P	S			Y		7C	40	@
P	S			Y	Y	7D	27	' (apostrophe)

The attributes in the headings are:

prot = protected
a/n = autoskip or numeric
hi = high intensity

spd = selector pen detectable
ndp = non-display print
mdt = modified data tag

The hex codes in the headings are:

ebcd = extended binary-coded decimal interchange code
asci = American National Standard Code for Information Interchange
char = graphic character equivalent to hex code

The characters in the body of the above table mean the following:

H = High
N = Numeric
S = Automatic skip
P = Protected
U = Unprotected
Y = Yes

Figure 25. Bit map for attributes other than those listed in DFHBMSCA

Magnetic slot reader (MSR) control value constants (DFHMSRCA)

A selection of MSR control bit patterns has been created for CICS and stored in copy book DFHMSRCA. The patterns are stored as named constants that can be loaded by simple application program commands. Provision of such constants saves the programmer from having to build a commonly used bit pattern whenever it is required. The constants supplied in DFHMSRCA are as follows:

Standard list DFHMSRCA

Constant	Meaning
DFHMSRST	MSR reset. All lights and buzzers off. MSR available for input.
DFHMSCON	Transaction ready for more input. Green and yellow on; emit short buzz; IN PROCESS (user) mode set.
DFHMSFIN	Input complete. Green on; emit short buzz; IN PROCESS mode reset.
DFHMSALR	Operator alert. Green, yellow, and red on; emit long buzz; IN PROCESS mode reset.
DFHMSALS	Operator alert. Green, yellow, and red on; emit long buzz; IN PROCESS mode set.
DFHMSIPY	IN PROCESS state set. Yellow on.
DFHMSIPN	IN PROCESS state reset.
DFHMSLKY	MSR operation inhibited. Yellow on.
DFHMSLKN	MSR input allowed. Green on. Yellow on.
DFHMSAEY	MSR auto enter on. Yellow on.
DFHMSAEN	MSR auto enter off. Yellow on.
DFHMSLBN	Long buzzer suppressed. Yellow on.
DFHMSLBY	Long buzzer permitted. Yellow on.
DFHMSSBN	Short buzzer suppressed. Yellow on.
DFHMSSBY	Short buzzer permitted. Yellow on.
DFHMSNOP	Leave all MSR settings unchanged.

Attention identifier constants (DFHAID)

The constants supplied in copy book DFHAID are as follows:

Standard list DFHAID

Constant	Meaning
DFHENTER	ENTER key
DFHCLEAR	CLEAR key
DFHPA1-DFHPA3	PA1-PA3 keys
DFHPF1-DFHPF24	PF1-PF24 keys
DFHOPID	OPERID or MSR
DFHMSRE	Extended(standard) MSR
DFHTRIG	Trigger field
DFHPEN	SELECTOR PEN or CURSOR SELECT key
DFHCLRP	CLEAR PARTITION key
DFHSTRF	Structured field pseudo-AID

Input commands

RECEIVE MAP command: The full syntax of the RECEIVE MAP command is shown below. The keywords are separated into those supported by minimum, standard, and full BMS.

```
Minimum BMS

RECEIVE MAP(name)
[MAPSET(name)]
[INTO(data-area)|SET(ptr-ref)]
[FROM(data-area) LENGTH(data-value)|
  TERMINAL[ASIS]]

Standard BMS

[INPARTN(name)]

Conditions: EOC, EODS, INVMPSZ,
INVPARTN, INVREQ, MAPFAIL,
PARTNFAIL, RDATT, UNEXPIN
```

The RECEIVE MAP command maps input data from a terminal into a data area in an application program. The process is described in "Receiving data from a display" on page 153. The INPARTN option is fully described in "Setting the expected input partition" on page 167.

When you use the RECEIVE MAP FROM command, the value of the LENGTH option must not exceed the length of the FROM data area.

Data from certain logical units is not mapped, but is left unaltered. See the appropriate CICS subsystem guide to see if this is true for a particular logical unit.

Following a RECEIVE MAP command, the inbound cursor position is placed in EIBCPOSN, and the terminal attention identifier (AID) placed in EIBAID.

RECEIVE PARTN command: The full syntax of the RECEIVE PARTN command is shown below. This command is available only on standard and full function BMS.

```
Standard BMS

RECEIVE PARTN(data-area)
{INTO(data-area)|SET(ptr-ref)}
LENGTH(data-area)
[ASIS]

Conditions: EOC, EODS, INVPARTN,
INVREQ, LENGERR
```

This command reads data from a partition on an 8775 terminal. It indicates which partition the data came from, and puts the data into the INTO or SET data area. You can then treat the data as though it had originated from a terminal in base (unpartitioned) state. This command is fully described in "Determining the actual input partition" on page 168.

Following a RECEIVE PARTN command, the inbound cursor position is placed in EIBCPOSN, and the terminal attention identifier (AID) placed in EIBAID.

Output commands

SEND PARTNSET command: The full syntax of the SEND PARTNSET command is shown below. This command is available only on standard and full function BMS.

```
Standard BMS

SEND PARTNSET[(name)]

Conditions: INVPARTNSET, INVREQ
```

This command associates the partition set specified by the PARTNSET option with the application program. If the partition set name is omitted, the terminal is reset to the base (unpartitioned) state. This command is fully described in "Loading the application partition set" on page 167.

SEND MAP command: The full syntax of the command is shown below. The options are separated into those supported by minimum, standard, and full function BMS.

```
Minimum BMS

SEND MAP(name)
[MAPSET(name)]
[FROM(data-area) LENGTH(data-value)|
  DATAONLY|MAPONLY]
[CURSOR[(data-value)]]
[FORMFEED]
[ERASE|ERASEAUP]
[PRINT]
[FREEKB]
[ALARM]
[FRSET]

Standard BMS

[NLEOM]
[MSR(data-value)]
[OUTPARTN(name)]
[ACTPARTN(name)]
[FMHPARM(name)]
[LDC(name)]

Full BMS

[ACCUM]
[SET(ptr-ref)|PAGING|
 TERMINAL[WAIT][LAST]]
[REQID(name)]
[L40|L64|L80|HONEOM]

Conditions: IGREQCD, IGREQID,
INVLDC, INVMPSZ, INVPARTN, INVREQ,
OVERFLOW, RETPAGE, TSIOERR, WRBRK
```

With the FROM option, the data area must contain a 12-byte TIOA prefix. The LENGTH data value includes the length of the 12-byte TIOA prefix.

You use the SEND MAP command to send mapped output data to a terminal. The process is described in "Sending data to a display" on page 150.

See page 197, 'CTRL', for information on the priorities by which certain options override others.

SEND TEXT command: The full syntax of the SEND TEXT command is shown below. The options are separated into those supported by standard and full BMS.

```
Standard BMS

SEND TEXT
FROM(data-area)
LENGTH(data-value)
[CURSOR(data-value)]
[FORMFEED]
[ERASE]
[PRINT]
[FREEKB]
[ALARM]
[NLEOM]
[LDC(name)]
[OUTPARTN(name)]
[ACTPARTN(name)]
[MSR(data-value)]

Full BMS

[SET(ptr-ref)|PAGING|
  TERMINAL[WAIT][LAST]]
[REQID(name)]
[HEADER(data-area)]
[TRAILER(data-area)]
[JUSTIFY(data-value)|JUSFIRST|
  JUSLAST]
[ACCUM]
[L40|L64|L80|HONEOM]

Conditions: IGREQCD, IGREQID,
INVLDC, INVPARTN, INVREQ, RETPAGE,
TSIOERR, WRBRK
```

This command is used to send 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. It is described under "Text processing" on page 159.

The command can be used to accumulate blocks of text to produce pages of text. This use is described in detail in "Cumulative text formatting" on page 181.

SEND TEXT MAPPED command: The full syntax of the SEND TEXT MAPPED command is shown below. This command is only available on full BMS.

```
Full BMS

SEND TEXT MAPPED
FROM(data-area)
LENGTH(data-value)
[PAGING|TERMINAL[WAIT][LAST]]
[REQID(name)]

Conditions: IGREQID, RETPAGE,
TSIOERR, WRBRK
```

This command is used to send data previously generated by a BMS SEND command specifying the SET option. It is described under "SEND TEXT MAPPED command" on page 190.

SEND TEXT NOEDIT command: The full syntax of the SEND TEXT NOEDIT command is shown below. This command is only available on full BMS.

```
Full BMS

SEND TEXT NOEDIT
FROM(data-area)
LENGTH(data-value)
[ERASE]
[PRINT]
[FREEKB]
[ALARM]
[OUTPARTN(name)]
[PAGING|TERMINAL[WAIT][LAST]]
[REQID(name)]
[L40|L64|L80|HONEOM]

Conditions: IGREQCD, IGREQID,
INVPARTN, TSIOERR, WRBRK
```

This command is used to output a user generated data stream. It differs from a terminal control SEND in that data may be output to temporary storage (using the PAGING option), or routed like any other BMS data. It is described under "SEND TEXT NOEDIT command" on page 191.

SEND CONTROL command: The full syntax of the SEND CONTROL command is shown below. The options are separated into those supported by minimum, standard, and full BMS.

```
Minimum BMS

SEND CONTROL
[CURSOR[(data-value)]]
[FORMFEED]
[ERASE|ERASEAUP]
[PRINT]
[FREEKB]
[ALARM]
[FRSET]

Standard BMS

[MSR(data-value)]
[OUTPARTN(name)]
[ACTPARTN(name)]
[LDC(name)]

Full BMS

[ACCUM]
[SET(ptr-ref)|PAGING|
  TERMINAL[WAIT][LAST]]
[REQID(name)]
[L40|L64|L80|HONEOM]

Conditions: IGREQCD, IGREQID,
INVLDC, INVPARTN, INVREQ, RETPAGE,
TSIOERR, WRBRK
```

This command is used to send device controls to a terminal, without also sending map or text data. It is described under "Sending device controls without display data" on page 153.

SEND PAGE command: The syntax of the SEND PAGE command is shown below. This command is only available on full BMS.

```
Full BMS

SEND PAGE
[RELEASE[TRANSID(name)]|RETAIN]
[TRAILER(data-area)]
[SET(ptr-ref)]
[AUTOPAGE[CURRENT|ALL]|NOAUTOPAGE]
[OPERPURGE]
[FMHPARM(name)]
[LAST]

Conditions: IGREQCD, INVREQ,
RETPAGE, TSIOERR, WRBRK
```

This command is used to complete a BMS logical message. If this is a paging message, the last page of the logical message is transmitted to temporary storage and the terminal operator paging transaction is initiated. If it is a terminal logical message, the last page is transmitted to the terminal. Its basic form is described under "SEND PAGE command" on page 174.

PURGE MESSAGE command: The syntax of the PURGE MESSAGE command is shown below. This command is only available on full BMS.

```
Full BMS

PURGE MESSAGE

Condition: TSIOERR
```

This command is used to discontinue the building of a BMS logical message. The portions of the logical message already built in main storage or in temporary storage are deleted. It is described under "PURGE MESSAGE command" on page 174.

ROUTE command: The syntax of this command is shown below. This command is only available on full BMS.

```
Full BMS

ROUTE
[INTERVAL(hhmmss)[1]|TIME(hhmmss)]
[ERRTERM[(name)]]
[TITLE(data-area)]
[LIST(data-area)]
[OPCLASS(data-area)]
[REQID(name)]
[LDC(name)][2]
[NLEOM]

Conditions: INVERRTERM, INVLDC,
INVREQ, RTEFAIL, RTESOME

[1] INTERVAL(0) is the default

[2] Logical units only.
```

This command is used to route a BMS logical message to one or more terminals and/or terminal operators. Its uses are discussed under "Message routing" on page 184.

BMS related ASSIGN options

The following options of the ASSIGN command may be useful to BMS application programs. The ASSIGN command is described in "Chapter 1.6. Access to system information" on page 51.

DESTCOUNT
this option has two uses.

- Following a BMS ROUTE command, it specifies that the value required is the number of different terminal types in the route list, and hence the number of overflow control areas that may be required.
- Within BMS overflow processing, specifies that the value required is the relative overflow control number of the destination that has encountered overflow. If this option is specified when overflow processing is not in effect, the value obtained will be meaningless. If no BMS commands have been issued, the INVREQ condition occurs. The format of the value is halfword binary.

INPARTN
specifies that the value required is the name of the most recent input partition. The format of the value is a one or two byte character.

LDCMNEM
specifies that the value required is the mnemonic of the LDC which overflowed most recently. For more information, see "Page overflow and partitions or LDCs" on page 180. The returned value is meaningless unless overflow processing is being performed. The format of the value is a one or two byte character.

LDCNUM
specifies that the value required is the LDC numeric value of the destination that has encountered overflow. This indicates the type of the LDC, such as printer or console. If this option is specified when overflow processing is not in effect, the value obtained will be meaningless. No exceptional condition occurs. The format of the value is a one-byte character.

MAPCOLUMN
specifies that the value required is the 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 BMS routing is in effect, the INVREQ condition is raised. The format of the value is halfword binary.

MAPHEIGHT
specifies that the value required is the height of the most recently positioned map. If no map has yet been positioned, or if BMS routing is in effect, the INVREQ condition is raised. The format of the value is halfword binary.

MAPLINE
specifies that the value required is the 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 BMS routing is in effect, the INVREQ condition is raised. The format of the value is halfword binary.

MAPWIDTH
specifies that the value required is the width of the most recently positioned map. If no map has yet been positioned, or if BMS routing is in effect, the INVREQ

condition is raised. The format of the value is halfword binary.

PAGENUM
specifies that the value required is the current page number for the destination that has encountered an overflow. If this option is specified when overflow processing is not in effect, the value obtained will be meaningless. If no BMS commands have been issued, the INVREQ condition occurs. The format of the value is halfword binary.

PARTNPAGE
specifies that the value required is the one through two character name of the partition that most recently caused page overflow. A blank value is returned if partitions are not in use.

PARTNSET
specifies that the value required is the one through six character name of the application partition set. A blank value is returned if there is no application partition set.

BMS options

ACCUM
specifies that this command is one of a number of commands that are used to build a logical message. The logical message is completed by a SEND PAGE command, or deleted by a PURGE MESSAGE command. For more details see "Logical message handling" on page 173.

ACTPARTN(name)
specifies the 1-or 2-character name of the partition that is to be activated. Activating a partition moves the cursor into the specified partition, and unlocks the keyboard for the specified partition.

This option is ignored if the target terminal does not support partitions, or if there is no application partition set.

ALARM
specifies that the 3270 audible alarm feature is to be activated. For logical units supporting FMHs (except interactive and batch logical units), ALARM signals BMS to set the alarm flag in the FMH.

ALL
specifies that if the ATTN key on a 2741 is pressed while a BMS logical message is being sent to the terminal, and the WRBRK condition is not active, transmission of the current page is to cease and no additional pages are to be transmitted. The logical message is deleted.

ASIS
specifies that the specification FEATURE=UCTRAN in the TCT for the terminal is to be overridden. Lowercase characters in the data stream are not translated to uppercase.

This option is not applicable to the initial input data for a transaction. For example, if a transaction is initiated by another transaction, and begins by receiving data originally output by that transaction, it cannot suppress upper case translation on the data.

AUTOPAGE
specifies that each page of a BMS logical message is to be sent to the terminal as soon as it is available. If paging upon request is specified for the terminal by the PGESTAT operand of DFHTCT TYPE=TERMINAL macro, AUTOPAGE overrides it for this logical message.

AUTOPAGE is assumed for 3270 printers; it does not apply to 3270 display terminals. If neither AUTOPAGE nor NOAUTOPAGE is specified, the terminal has the paging status specified for it by the DFHTCT TYPE=TERMINAL macro.

CURRENT
specifies that if the ATTN key on a 2741 is pressed while a BMS logical message is being sent to the terminal, and the WRBRK condition is not active, transmission of the current page is to cease and transmission of the next page (if any) is to begin.

CURSOR[(data-value)]
specifies the position to which the 3270 or 3604 cursor is to be returned upon completion of a SEND MAP, SEND TEXT, or SEND CONTROL command.

The data value must be a halfword binary value that specifies the cursor position relative to zero; the range of values that can be specified depends on the size of the screen being used. If no data value is specified, symbolic cursor positioning is assumed, as described in "Cursor positioning" on page 153.

This option overrides any IC option of the ATTRB operand of the DFHMDF macro. If ACCUM is being used, the most recent value of CURSOR specified is used to position the cursor.

DATAONLY
specifies that only application program data is to be written. The attribute characters (3270 only) 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 will be unchanged. Any default data or attributes from the map are ignored. This is described under "Refreshing and modifying displays" on page 151.

ERASE
specifies that the screen printer buffer or partition is to be erased and the cursor returned to the upper left corner of the screen before this page of output is

displayed. (This option applies only to the 3270, or 8775, and to the 3604 Keyboard Display.) The first output operation in any transaction, or in a series of pseudoconversational transactions, should always specify ERASE. For transactions attached to 3270 screens or printers, this will also ensure that the correct screen size is selected, as defined for the transaction by the SCRNSZE keyword of CEDA DEFINE PROFILE or the SCRNSZE operand of DFHPCT.

ERASEAUP
specifies that before this page of output is displayed, all unprotected character locations in the partition or the entire screen are to be erased. (This option applies only to the 3270 and 8775.)

ERRTERM[(name)]
specifies the name of the terminal to be notified if the message is deleted because it is undeliverable. The message number, title identification, and destination are indicated. If no name is specified, the originating terminal is assumed.

This option is operative only if the PRGDLAY operand has been specified in the DFHSIT macro.

FMHPARM(name)
specifies the name (1 through 8 characters) of the outboard map to be used. (This option applies only to 3650 logical units with outboard formatting.)

FORMFEED
specifies that a new page is required. For 3270 printers and displays, the formfeed character is positioned at the start of the buffer. The application program must thus ensure that this buffer position is not overwritten by map or text data.

This option is fully discussed under "Printed output" on page 156. It is ignored if the target terminal does not support formfeed (that is, the DFHTCT TYPE=TERMINAL system macro does not specify FF=YES).

The FORMFEED option can appear on any SEND TEXT ACCUM command. You need specify it only once within a physical page because it always forces a formfeed at the start of the physical page. To force a formfeed at the start of a particular SEND TEXT ACCUM command, use the JUSFIRST option (see below) instead.

FREEKB
specifies that the 3270 keyboard should be unlocked after the data is written. If FREEKB is omitted, the keyboard remains locked.

The keyboard lock status is maintained separately for each partition on a terminal which supports partitions.

FROM(data-area)
specifies the data area containing the data to be processed by a SEND MAP or SEND TEXT command, or data to be mapped by a RECEIVE MAP command.

FRSET
specifies that the modified data tags (MDTs) of all fields currently in the 3270, (or partition) 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.

HEADER(data-value)
specifies the header data to be placed at the beginning of each page of text data. The format of the header is described under "Cumulative text formatting" on page 181.

HONEOM
specifies that the default printer line length is to be used. This length should be the same as that specified in the PGESIZE or ALTPAGE operand of the DFHTCT TYPE=TERMINAL system macro, and the same as the printer platen width, otherwise the data may not format correctly.

INPARTN (name)
specifies the 1 or 2 character name of the partition in which the terminal operator is expected to enter data. If the terminal operator enters data in some other partition, the INPARTN partition is activated, the keyboard is unlocked for the partition, and an error message is output to any error message partition. This option is ignored if the terminal does not support partitions, or if there is no application partition set.

INTERVAL(hhmmss)
specifies the interval of time after which the data is to be transmitted to the terminals specified in the ROUTE command.

INTO(data-area)
on a RECEIVE MAP command specifies the data area into which the mapped data is to be written.

On a RECEIVE PARTN command, the INTO option specifies the area into which the input data stripped of partition controls is to be written. The length of this area must be specified by the LENGTH option. If the area is not large enough to hold the input data, the input data is truncated, and the LENGERR condition raised. The length option data area is set to the length of data received, prior to any truncation.

JUSFIRST
specifies that the text data is to be placed at the top of the page. Any partially formatted page from previous requests is considered to be complete. If the HEADER option is specified, the header precedes the data. See also the description of the JUSTIFY option.

JUSLAST

specifies that the text data is to be positioned at the bottom of the page. The page is considered to be complete after the request has been processed. If the TRAILER option is specified, the trailer follows the data. See also the description of the JUSTIFY option.

JUSTIFY(data-value)

specifies the line of the page at which the text data is to be positioned. The data value must be a halfword binary value in the range 1 through 240. Although they may not be specified as constants, the special values −1 and −2 can be supplied dynamically to signify JUSFIRST or JUSLAST, respectively.

LAST

specifies that this is the last output operation for a transaction and, therefore, the end of a bracket. If RELEASE is specified, LAST is assumed unless the SEND PAGE command is terminating a routing operation. This option applies to logical units only.

LDC(name)

specifies a two-character mnemonic to be used to determine the logical device code (LDC) to be transmitted in the FMH to the logical unit. The mnemonic represents an LDC entry specified in the DFHTCT TYPE=LDC system macro.

When an LDC is specified, BMS uses the device type, the page size, and the page status associated with the LDC mnemonic to format the message. These values are taken from the extended local LDC table for the LU, if it has one. If the LU has only a local (unextended) LDC table, the values are taken from the system LDC table. The numeric value of the LDC is obtained from the local LDC table, unless this is an unextended table and the value is not specified, in which case it is taken from the system table.

If the LDC option of a SEND MAP or ROUTE command is omitted, the LDC mnemonic specified in the DFHMSD macro is used. If the LDC option has also been omitted from the DFHMSD macro, the action depends on the type of logical unit, as follows:

3601 LU − the first entry in the local or extended local LDC table is used, if there is one. If a default cannot be obtained in this way, a null LDC numeric value (X'00') is used. The page size used is the value that is specified in the DFHTCT TYPE=TERMINAL system macro, or (1,40) if such a value is not specified.

LUTYPE4 LU, batch LU, or batch data interchange LU − the local LDC table is not used to supply a default LDC; instead, the message is directed to the LU console (that is, to any medium that the LU elects to receive such messages. For a batch data interchange LU, this does not imply sending an LDC in an FMH). The page size is obtained in the manner described for the 3601 LU.

For message routing, the LDC option of the ROUTE command takes precedence over all other sources. If this option is omitted and a route list is specified (LIST option), the LDC mnemonic in the route list is used; if the route list contains no LDC mnemonic, or no route list is specified, a default LDC is chosen as described above.

LENGTH(data-value)

specifies the length of the data to be formatted as a halfword binary value.

On a RECEIVE PARTN command, the LENGTH option must be set to the length of any INTO area prior to the command. After the command, BMS sets the LENGTH option to the length of data received prior to any truncation if the INTO area is too small.

LIST(data-area)

specifies the data area that contains a list of terminals and/or operators to which data is to be directed. If this option is omitted, all terminals supported by BMS receive the data (unless the OPCLASS option has some effect). For more information, see "Route list and operator class codes (LIST and OPCLASS)" on page 187.

L40

specifies the line length for a 3270 printer; a carrier return and line feed are forced after 40 characters have been printed on a line. Unexpected results will occur if this differs from the page width specified by the PAGESIZE or ALTPGE options of the DFHTCT TYPE=TERMINAL macro.

L64

specifies the line length for a 3270 printer; a carrier return and line feed are forced after 64 characters have been printed on a line. Unexpected results will occur if this differs from the page width specified by the PAGESIZE or ALTPGE options of the DFHTCT TYPE=TERMINAL system macro.

L80

specifies the line length for a 3270 printer; a carrier return and line feed are forced after 80 characters have been printed on a line. Unexpected results will occur if this differs from the page width specified by the PAGESIZE or ALTPGE options of the DFHTCT TYPE=TERMINAL system macro.

MAP(name)

specifies the name (1 through 7 characters) of the map to be used.

MAPONLY

specifies that only default data from the map is to be written. If this option is specified, the FROM option must not be specified.

MAPSET(name)

specifies the unsuffixed name (1 through 7 characters) of the map set to be used. The map set must reside in the CICS program library, and an entry for it must exist in the processing program table (PPT). If this option is not specified, the name given in the MAP option is assumed to be that of the map set.

MSR(data-value)
specifies the 4-byte data value which controls the 10/63 magnetic stripe reader attached to an 8775 or 3643 terminal. A set of constants is provided in DFHMSRCA to assist in setting this 4 byte area. This option is ignored if the terminal's DFHTCT TYPE=TERMINAL macro does not specify FEATURE=MSRCNTRL. You will find more information about this under "10/63 magnetic slot reader control" on page 170.

NLEOM
specifies that data for a 3270 printer or a 3275 display with the printer adapter feature should be built with blanks and new-line (NL) characters, and that an end-of-message (EM) character should 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 MAP or SEND TEXT command used to build a logical message, and in the ROUTE command if the message is to be routed. The option is ignored if the device receiving the message (direct or routed) is not one of those noted above.

If this option is used, buffer updating and attribute modification of fields previously written into the buffer are not allowed. CICS includes the ERASE option with every write to the terminal.

The NL character occupies a buffer position. A number of buffer positions, equivalent to the value of the PGESIZE or ALTPGE operand of the DFHTCT system macro for that terminal, is unavailable for data. This may cause data to wrap around in the buffer; if this occurs, the PGESIZE value must be reduced.

The NLEOM option overrides the ALARM option if the latter is present.

The NLEOM option is further discussed under "Printer support" on page 160.

NOAUTOPAGE
specifies that pages of a BMS logical message are to be sent one at a time to the terminal. BMS sends the first page to the terminal when the terminal becomes available or upon request of the terminal operator. Subsequent pages are sent to the terminal in response to requests from the terminal operator. (See the *CICS/MVS CICS-Supplied Transactions* manual for further information.)

If automatic paging is specified for the terminal by the PGESTAT operand of the DFHTCT TYPE=TERMINAL system macro, NOAUTOPAGE overrides it for this logical message. For logical units, NOAUTOPAGE applies to all pages for all LDCs in the logical message.

NOAUTOPAGE does not apply to 3270 printers.

OPCLASS(data-area)
specifies the data area that contains a list of operator classes to which the data is to be routed. The classes are supplied in a three-byte field, each bit position corresponding to one of the codes in the range 1 through 24 but in reverse order, that is, the first byte corresponds to codes 24 through 17, the second byte to codes 16 through 9, and the third byte to codes 8 through 1.

OPERPURGE
specifies that CICS is to delete the BMS logical message only when the terminal operator requests deletion. If the option is omitted, CICS deletes the message if the operator enters data that is not a paging command.

OUTPARTN (name)
specifies the 1-through 2-character name of the partition to which data is to be sent. This option is ignored if the terminal does not support partitions, or if there is no application partition set associated with the terminal. If there is an application partition set, and the OUTPARTN option is omitted, data is sent to the partition named by the PARTN operand of the DFHMSD or DFHMDI map definition macro. If maps are not used, or if there is no PARTN operand, the output is sent to the first partition in the partition set.

PAGING
specifies that the output data is not to be sent immediately to the terminal, but is to be placed in temporary storage and displayed in response to paging commands entered by the terminal operator.

If PAGING is specified with a REQID that is defined as recoverable in the temporary storage table (TST), CICS provides message recovery for logical messages if the task has reached a sync point.

PARTN (data area)
is set by BMS to the 1 or 2 character name of the input partition.

PARTNSET[(name)]
specifies the 1 through 6 character name of a partition set to be associated with the application program. If no name is specified, the terminal is set to unpartitioned state.

PRINT
specifies that a print operation is to be started at a 3270 printer or at a 3275 with the printer adapter feature, or that data on an LUTYPE2 (3274/76 or 3790) 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.

RELEASE

specifies that, after a SEND PAGE command, control is to be returned to the program at the next higher logical level, or to CICS (if the issuing program is at the highest logical level), after the pages have been written to the terminal. For more details of the effect of this option, see "Logical messages for terminal operator paging" on page 175.

REQID(name)

specifies a 2-character prefix to be used as part of a temporary storage identifier for CICS message recovery. Only one prefix can be specified for each logical message. The default prefix is **.

BMS message recovery is provided for a logical message only if the PAGING option is specified in the BMS SEND commands and if the sync point has been reached.

RETAIN

specifies that, after a SEND PAGE command, control is to be returned to the application program after the pages have been written to the terminal. For more details of the effect of this option, see "Logical messages for terminal operator paging" on page 175.

SET(ptr-ref)

specifies the pointer that is to be set to the address of the input or output data.

For input, the pointer is set to the address of the 12-byte prefix to the mapped data.

For output, the SET option specifies that the completed pages are to be returned to the application program. The pointer is set to the address of a list of completed pages. See "Chapter 3.2-4. Full function BMS" on page 173 for further details.

The application program regains control either immediately following the BMS SEND command (if the current page is not yet completed), or at the label specified in a HANDLE CONDITION RETPAGE command if the page has been completed.

TERMINAL

specifies that input data is to be read from the terminal that originated the transaction, or that output data is to be sent to that terminal.

TIME(hhmmss)

specifies the time of day at which data is to be transmitted to the terminals specified in the ROUTE command.

TITLE(data-area)

specifies the data area that contains the title to be used with a routing logical message. This title will appear as part of the response to a page query command. For the format of the title, see "TITLE option of the ROUTE command" on page 186.

TRAILER(data-area)

specifies the text data area that contains trailer data to be placed at the bottom of each output page (with a SEND TEXT command) or at the bottom of the last page only (with a SEND PAGE command). For the format of the trailer data, see "Cumulative text formatting" on page 181.

TRANSID(name)

specifies the transaction identifier to be used with the next input message from the terminal to which the task is attached. The identifier can consist of up to 4 alphanumeric characters; it must have been defined in the program control table (PCT). TRANSID is valid only if SEND PAGE RELEASE is specified.

If this option is specified in a program that is not at the highest logical level, the specified transaction identifier will be used only if a new transaction identifier is not provided in another SEND PAGE command (or in a RETURN program control command) issued in a program at a higher logical level.

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 will return to the application program once the output operation has started. A subsequent input or output command (terminal control, BMS, or batch data interchange) will cause the application program to wait until the previous command has been completed.

BMS exceptional conditions

Some of the following exceptional conditions may occur in combination with others. CICS checks for these conditions in the following order:

```
LENGERR
OVERFLOW
IGREQCD
TSIOERR
INVREQ
RETPAGE
MAPFAIL
RTEFAIL
RTESOME
INVERRTERM
IGREQID
INVLDC
INVMPSZ
EODS
INVPARTNSET
INVPARTN
PARTNFAIL
UNEXPIN
EOC
```

If more than one of these conditions occurs, only the first one found to be present is passed to the application program.

EOC
occurs if the request/response unit (RU) is received with the end-of-chain (EOC) indicator set. It applies only to logical units.

Default action: ignore the condition.

EODS
occurs if no data is received (only an FMH). It applies only to 3770 batch LUs and to 3770 and 3790 batch data interchange LUs.

Default action: terminate the task abnormally.

IGREQCD
occurs when an attempt is made to execute a SEND MAP, SEND PAGE, SEND TEXT, or SEND CONTROL command after a SIGNAL data flow control command with an RCD (request change direction) code has been received from an LUTYPE4 LU.

Default action: terminate the task abnormally.

IGREQID
occurs if the prefix specified in the REQID option on a BMS SEND command is different from that established by a previous REQID option, or by default for this logical message (REQID (**)).

Default action: terminate the task abnormally.

INVERRTERM
occurs if the terminal identifier specified in the ERRTERM option of a ROUTE command is invalid or is assigned to a type of terminal that does not support BMS.

Default action: terminate the task abnormally.

INVLDC
occurs if the specified LDC mnemonic is not included in the LDC list for the logical unit.

Default action: terminate the task abnormally.

INVMPSZ
occurs if the specified map is too wide for the terminal, or if a HANDLE CONDITION OVERFLOW command is active and the specified map is too long for the terminal.

Default action: terminate the task abnormally.

INVPARTN
occurs if the specified partition is not defined in the partition set associated with the application program.

Default action: terminate the task abnormally.

INVPARTNSET
occurs if the partition set named in the SEND PARTNSET command is not a valid partition set (for example, it may be a map set).

Default action: terminate the task abnormally.

INVREQ
occurs if a request for BMS services is invalid for any of the following reasons:

- The disposition (TERMINAL, PAGING, or SET) of a BMS logical message is changed prior to its completion by a SEND PAGE command.
- A SEND PARTNSET command is issued while a logical message is active.
- Text data is output to the same partition or LDC as mapped data while a BMS logical message is active. If neither partitions nor LDCs are in use, text data is output to the same logical message as mapped data.
- A separate SEND TEXT ACCUM or SEND MAP ACCUM command is issued to the terminal that originated the transaction while a routed logical message is being built.
- The TRAILER option is specified in a SEND PAGE command when terminating a logical message built with SEND MAP commands only.
- An output mapping command is issued for a map without field specifications by specifying the FROM option without the DATAONLY option.
- During overflow processing data is sent to a different LDC than the LDC that caused page overflow.
- Partitions are in use, the OUTPARTN option has not been coded on a SEND MAP command, but the PARTN operand has been coded in the map set definition. If the condition arises, it suggests that different versions of the map set have different PARTN values, and that the suffix deduced for the partition is not the same as the suffix of the loaded map set.
- The length of a header on a SEND TEXT command is negative.
- The length of a trailer on a SEND TEXT or SEND PAGE command is negative.
- Bytes 10 through 15 of a route list entry do not contain blanks on a ROUTE command.
- RECEIVE MAP and RECEIVE PARTN commands cannot be issued in a nonterminal task, because these tasks do not have a TIOA or a TCTTE.

Default action: terminate the task abnormally.

LENGERR
occurs if the INTO area of a RECEIVE PARTN command is not large enough to hold the input data.

Default action: truncate the data to fit within the INTO area.

MAPFAIL

occurs if the data to be mapped has a length of zero or does not contain a set-buffer-address (SBA) sequence. It applies only to 3270 devices. The receiving data area will contain the unmapped input data stream. The amount of unmapped data moved to the user's area will be limited to the length specified in the LENGTH option of the RECEIVE MAP command.

This condition also arises if a program issues a RECEIVE MAP command to which the terminal operator responds by pressing a CLEAR or PA key or pressing ENTER or a PF key without entering data.

Default action: terminate the task abnormally.

OVERFLOW

occurs if the mapped data does not fit on the current page. This condition is only raised if a HANDLE CONDITION OVERFLOW command is active.

Default action: ignore the condition.

PARTNFAIL

occurs if the terminal operator persists in entering data in a partition other than that specified by the INPARTN option of the RECEIVE MAP command.

Default action: terminate the task abnormally.

RDATT

occurs if a RECEIVE MAP command is terminated by the operator using the ATTN key rather than the RETURN key. It applies only to the 2741 Communications Terminal, and only if 2741 read attention support has been generated for CICS.

Default action: ignore the condition.

RETPAGE

occurs if the SET option is specified and one or more completed pages are ready for return to the application program.

Default action: return control to the application program at the point immediately following the BMS SEND command.

RTEFAIL

occurs if a ROUTE command would result in the message being sent only to the terminal that initiated the transaction.

Default action: return control to the application program at the point immediately following the ROUTE command.

RTESOME

occurs if any of the terminals specified by options of a ROUTE command will not receive the message.

Default action: return control to the application program at the point immediately following the ROUTE command.

TSIOERR

occurs if there is an unrecoverable temporary storage input/output error.

Default action: terminate the task abnormally.

UNEXPIN

Raised when unexpected or unrecognized data is received. This only applies to batch data interchange terminals.

Default action: terminate the task abnormally.

WRBRK

occurs if a SEND command is interrupted by the terminal operator pressing the ATTN key. It applies only to the 2741 Communication Terminal, and only if write break support has been generated for CICS.

Default action: ignore the condition.

Chapter 3.3. Terminal control

The CICS terminal control program provides for communication between user-written application programs and terminals and logical units, by means of terminal control commands.

Terminal control uses the standard access methods available with the host operating system, as follows:

- **BTAM (Basic Telecommunications Access Method)** is used by CICS for most start-stop and BSC terminals. Optionally, **TCAM (Telecommunications Access Method)** can be specified.
- **SAM (Sequential Access Method)** is used where keyboard terminals are simulated by sequential devices such as card readers and line printers.
- **VTAM (Virtual Telecommunications Access Method)** or **TCAM (Telecommunications Access Method)** is used for systems network architecture (SNA) terminal systems.

Terminal control polls terminals to see if they are ready to transmit or receive data. Terminal control handles code translation, transaction identification, synchronization of input and output operations, and the line control necessary to read from or write to a terminal.

The application program is freed from having to physically control terminals. During processing, an application program is connected to one terminal for one task and the terminal control program monitors which task is associated with which terminal. The task to be initiated is determined as described in "Terminal-oriented task identification" on page 223.

Terminal control is used for communication with terminals. In SNA systems, however, it is used also to control communication with logical units or with another CICS system.

A logical unit (LU) represents either a terminal directly, or a program stored in a subsystem controller which in turn controls one or more terminals.

The CICS application program communicates, by means of the logical unit, either with a terminal or with the stored program. For example, a 3767 terminal is represented by a single logical unit without an associated user-written application program. In contrast, a 3790 subsystem is represented by a 3791 controller, user-written 3790 application programs, and one or more 3790 terminals; when the subsystem is configured, one or more logical units are designated by the user.

Terminal control is used also for communicating with a remote system by means of distributed transaction processing (DTP). SNA protocols are available, through terminal control commands, to initiate and terminate a conversation (a session) with a remote LU6.1 Logical Unit.

This conversation is carried on between a principal facility and one or more alternate facilities.

A **principal facility** for a task is a terminal, LU6.1 session, or LU6.2 session that is made available to the application program when the task is initiated.

An **alternate facility** for a task is an LU6.1 session, or an LU6.2 session acquired as needed by the application program. In general, terminal-control commands that refer to an alternate facility should include the SESSION option, or for LU6.2, the CONVID option.

The ALLOCATE and FREE commands allow the application program to acquire and release these alternate facilities and allow both principal and alternate facilities to be used at the same time.

For LU6.1, the BUILD ATTACH and EXTRACT ATTACH commands, together with the ATTACHID option of the SEND command, allow the application program to attach a transaction in a remote system.

For LU6.2, the CONNECT PROCESS command allows the application program to attach a transaction in the remote system.

Fields in the EIB allow access to indicators that give the status of the conversation after execution of RECEIVE or CONVERSE commands. For example, EIBEOC, EIBATT, and EIBFMH provide more information about the received data, and EIBSYNC, EIBFREE, and EIBRECV provide more information about the session.

The INVITE option of the SEND command allows the optimization of SNA flows that occur when communicating with another transaction, or with IMS.

Distributed transaction processing is described fully in the *CICS/MVS Intercommunication Guide*.

Commands and options that apply specifically to logical units are described later in the chapter.

Terminal control commands are provided to request the following services that are applicable to most, or all, of the types of terminal or logical unit supported by CICS:

- Read data from a terminal or logical unit (RECEIVE).
- Write data to a terminal or logical unit (SEND).
- Converse with a terminal or logical unit (CONVERSE).
- Synchronize terminal input/output for a transaction (WAIT TERMINAL).
- Send an asynchronous interrupt (ISSUE SIGNAL).

- Relinquish use of a communication line (ISSUE RESET).
- Disconnect a switched line or terminate a session with a logical unit (ISSUE DISCONNECT).

It is possible to read records from a card reader and read records from or write records to a disk data set, magnetic tape unit, or a line printer defined by the system programmer as a card-reader-in/line-printer-out (CRLP) terminal. For more information, see "Sequential terminal support" on page 311.

Other services available in response to terminal control commands apply to specific types of terminal. The permissible commands and options that can be used by specific terminal types are detailed later in the chapter. Because many types of terminal are supported by CICS, many special services are provided. (For a list of terminals supported by CICS, see the *CICS/MVS Facilities and Planning Guide*.) In particular, a large number of commands are provided for communicating with display devices such as the IBM 3270 Information Display System; these are described in "Display device operations" on page 230.

The options that follow the command depend on the terminal or logical unit (and sometimes, access method) used and the operations required. Options included in a terminal control command that do not apply to the device being used will be ignored.

Exceptional conditions that occur during the execution of terminal control commands are handled as described in "Chapter 1.5. Exceptional conditions" on page 43.

Commands and options for terminals and logical units

The commands and options described in this section apply to all terminals and logical units. There may, however, be others that apply to specific devices. If so, details are given later in the chapter under headings for the device types.

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, the corresponding alternative can be specified instead:

Option	Alternative
LENGTH	FLENGTH
TOLENGTH	TOFLENGTH
FROMLENGTH	FROMFLENGTH
MAXLENGTH	MAXFLENGTH

Application programs should be consistent in their use of fullword and halfword options on terminal control commands. The maximum value that can be specified as a parameter on any length keyword is 32,767.

Read from terminal or logical unit (RECEIVE)

The 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 is available to the task until the | next terminal I/O command. However, the area does not belong to the task and will be released by CICS while processing the next request. Therefore, this area cannot be passed back to CICS for further processing.

The application can use MAXLENGTH to specify the maximum length of data that the program will accept. If the MAXLENGTH option is omitted on a RECEIVE command for which the INTO option is specified, the maximum length of data the program will accept can be specified in the LENGTH option. If the MAXLENGTH option is omitted on a RECEIVE command for which the SET option is specified, CICS will acquire 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 will be made available to satisfy subsequent RECEIVE commands. If NOTRUNCATE is not specified, the data is truncated and the LENGERR condition is raised. 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. The first RECEIVE command in a task started by a terminal will not issue a terminal control read but will simply copy the input buffer, even if the data length is zero. A second RECEIVE command must be issued to cause a terminal control read.

When a PA key is defined in the SIT to mean PRINT, and that key is pressed in response to a RECEIVE command, it has no effect on the application program. The RECEIVE command is satisfied, and the application allowed to continue, when another attention (that is, one of the other PA keys, any of the PF keys, the ENTER key, or the light pen) is made at the keyboard.

Write to terminal or logical unit (SEND)

The SEND command is used to write data to a terminal or logical unit. The options FROM and LENGTH 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), a SEND command should always precede the first RECEIVE in a transaction.

WAIT option of SEND command: Unless the WAIT option is specified also, the transmission of the data associated with the SEND command is deferred until a later event, such as a sync point, occurs. This deferred transmission reduces the flows of data by allowing data flow controls to be transmitted with the data.

Transmission is not deferred for distributed transaction processing when interregion communication (IRC) is in use. See the *CICS/MVS Intercommunication Guide* for further information.

Synchronize terminal I/O for a transaction (WAIT TERMINAL)

This command is used to ensure that a terminal operation has completed before further processing occurs in a task under which more than one terminal or logical unit operation is performed. Alternatively, the WAIT option can be specified in a SEND command. (A wait is always carried out for a RECEIVE command.)

Either method may cause execution of a task to be suspended. If suspension is necessary, control is returned to CICS. Execution of the task is resumed when the operation is completed.

Even if the WAIT option is not specified in a SEND command, the EXEC interface program will ensure that the operation is completed before issuing a subsequent RECEIVE or SEND command.

It is the user's responsibility to code an explicit wait between two terminal control operations if mixing command and macro level requests.

Converse with terminal or logical unit (CONVERSE)

For most terminals or logical unit types a conversational mode of communication can be used. The CONVERSE command is used for this purpose. In general, the CONVERSE command can be considered as a combination of a SEND command followed immediately by a WAIT TERMINAL command and then by a RECEIVE command. However, not all options of the SEND and RECEIVE commands are valid for the CONVERSE command. Specific rules are given in the syntax descriptions for different devices later in the chapter. The TOLENGTH option is equivalent to the LENGTH option of the RECEIVE command, and the FROMLENGTH option is equivalent to the LENGTH option of the SEND command.

Send an asynchronous interrupt (ISSUE SIGNAL)

This command is used, in a transaction in receive mode, to signal to the sending transaction that a mode change is needed. The execution of the command will raise the SIGNAL condition on the next SEND or RECEIVE command executed in the sending transaction, and a previously executed HANDLE CONDITION command for this condition can be used either to action the request or to ignore it.

Relinquish a communication line (ISSUE RESET)

This command is used to relinquish use of a communication line. The command applies only to binary synchronous devices using BTAM. The next BTAM operation will be a read or write initial.

Disconnect a switched line (ISSUE DISCONNECT)

This command is used to break a line connection between a terminal and the processor, or to break a session between TCAM or ACF/VTAM logical units, when the transaction is completed. If the terminal is a buffered device, the data in the buffers will be lost.

When used with a VTAM terminal, ISSUE DISCONNECT, which does not become effective until the task completes, signs off the terminal, frees the COMMAREA, clears the next TRANID, stops any BMS paging, and, if autoinstall is in effect, deletes the terminal definition

Terminal-oriented task identification

When CICS receives input from a terminal to which no task is attached, it has to determine which transaction should be initiated. The methods by which the user can specify the transaction to be initiated and the sequence in which CICS checks these specifications are as follows (see also Figure 26 on page 224).

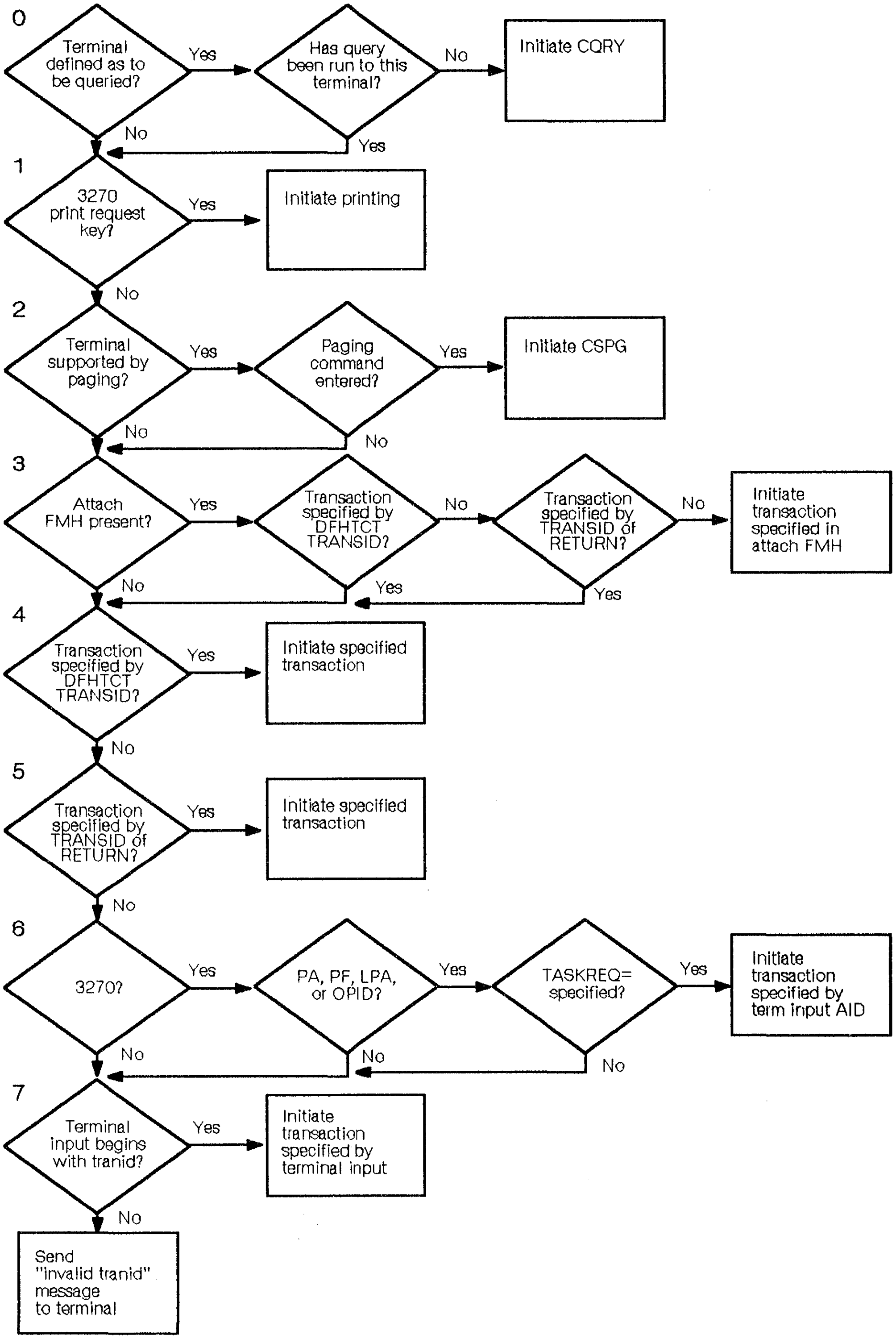

0
Terminal defined as to be queried?
Yes
Has query been run to this terminal?
No
Initiate CQRY
No
Yes
1
3270 print request key?
Yes
Initiate printing
No
2
Terminal supported by paging?
Yes
Paging command entered?
Yes
Initiate CSPG
No
No
3
Attach FMH present?
Yes
Transaction specified by DFHTCT TRANSID?
No
Transaction specified by TRANSID of RETURN?
No
Initiate transaction specified in attach FMH
No
Yes
Yes
4
Transaction specified by DFHTCT TRANSID?
Yes
Initiate specified transaction
No
5
Transaction specified by TRANSID of RETURN?
Yes
Initiate specified transaction
No
6
3270?
Yes
PA, PF, LPA, or OPID?
Yes
TASKREQ= specified?
Yes
Initiate transaction specified by term input AID
No
No
No
7
Terminal input begins with tranid?
Yes
Initiate transaction specified by terminal input
No
Send "invalid tranid" message to terminal

Figure 26. Terminal oriented task identification

The system macros referred to in the following tests are described in the *CICS/MVS Resource Definition (Macro)* manual. Where applicable, the CEDA transaction can be used, instead of system macros, to define the resources, as described in the *CICS/MVS Resource Definition (Online)* manual.

Test 0:

(a) Is this terminal defined as to be queried?

(b) Has query been run to this terminal?

If yes to (a) and no to (b), the query transaction, CQRY, is initiated before any other transaction.

Test 1:

Is the input from a PA key (of a 3270 terminal) that has been defined at system initialization as the print request key?

If yes, printing of the data displayed on the screen is initiated.

Test 2:

(a) Is this terminal of a type supported by BMS terminal paging?

(b) Is the input a paging command? (The terminal operator can enter paging commands defined in the DFHSIT system macro.)

If yes to both (a) and (b), the transaction CSPG, which processes paging commands, is initiated.

Test 3:

If an attach FMH is present in the data stream and Tests 4 and 5 are not fulfilled, the transaction specified in the attach FMH is initiated. The architectured attach names are converted to "CSMI".

Test 4:

Does the terminal control table entry for the terminal include a transaction identifier (specified by the TRANSID operand of the DFHTCT TYPE=TERMINAL system macro).

If yes, the specified transaction is initiated.

Test 5:

Is a transaction specified by the TRANSID option of a program control RETURN command (or by the application program moving the transaction name into TCANXTID)?

If yes, the specified transaction is initiated.

Test 6:

(a) Is the terminal a 3270 (including 3270 logical unit and 3650 host-conversational (3270) logical unit, connected via VTAM)?

(b) Is the input from a PA key, PF key, light pen attention, or operator identification card reader?

(c) Is this input specified by the TASKREQ operand of the DFHPCT TYPE=ENTRY system macro?

If yes to (a), (b), and (c), the program specified by the PROGRAM operand of the same DFHPCT TYPE=ENTRY system macro is given control.

Test 7:

Is a valid transaction identification specified by the first one to four characters of the terminal input?

If yes, the specified transaction is initiated.

For all PA keys and some LPAs (light-pen attention) there cannot be terminal input. If there is no terminal input an "invalid transaction identification" message is sent to the terminal.

If none of the above tests is met, an invalid transaction identification exists, and message DFH2001 (INVALID TRANSACTION IDENTIFICATION xxxx – PLEASE RESUBMIT) is sent to the terminal.

The IBM 3735 Programmable Buffered Terminal makes an exception to this sequence when operating in inquiry mode. The test of input from the terminal (test 7 above) is given highest priority.

Commands and options for logical units

An application program communicates with a TCAM or VTAM logical unit in much the same way as it does with a TCAM or BTAM terminal (that is, by using the terminal control commands described above). 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 types of logical units and the related protocols for each of the SNA subsystems supported by CICS are described in the CICS guides for the subsystems. (See the "CICS/MVS 2.1.2 library" on page vii.)

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 commands 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.

If the above protocols are not followed, the transaction will be abended, unless the read ahead queueing feature (RAQ=YES) is specified by the system programmer. This feature allows the application program to ignore the EIBRECV indicator and to send and receive at any time. However, it should only be used with transactions that support both bisynchronous devices and logical units.

For displays, the transaction would normally be in send mode, provided that the INVITE option is not used, and can ignore the EIBRECV indicator. Displays work with a subset of the full protocols (see the *CICS/MVS Facilities and Planning Guide* for further information).

Send/receive protocol (invite option)

The INVITE option of a 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 a RECEIVE command as its next operation.

Chaining of input data

The unit of data from a logical unit is the request/response unit (RU). One or more RUs can be grouped together and treated as a chain.

The last RU in a chain (even if it is the only RU in the chain) raises an end-of-chain (EOC) condition. When this occurs, a HANDLE CONDITION EOC command will give control to a user-written routine, which can do any additional processing required when the complete chain has been received.

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 and on whether the terminal control table CHNASSY operand is specified by the system programmer. The syntax descriptions for individual logical units in this chapter omit the EOC condition unless it is likely that meaningful use may be made of the fact that it has not been received. The IGNORE CONDITION command can be used to ignore the EOC condition in cases where it is raised on every RECEIVE command.

The EOC condition may occur simultaneously with the EODS (end-of-data-set) and/or INBFMH (inbound-FMH) conditions. When this happens, the user-written routine for the EODS or INBFMH conditions will be given control rather than the EOC routine.

The system programmer specifies, in the TCTTE, whether or not chaining is to occur. If chain assembly is specified, 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.

Chaining of output data

As in the case of input data, output data is transmitted as request/response units (RUs). If the length of the data to be sent exceeds the RU size, CICS breaks up the data into RUs and transmits these RUs as a chain. During transmission from CICS to the logical unit, the RUs are marked FOC (first-of-chain), MOC (middle-of-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 OC (only-in-chain).

If the system programmer specified that the application program can control the chaining of outbound data, the application program uses the CNOTCOMPL (chain-not-complete) option of the SEND command to indicate continuation of 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 FMH, LAST, or (for the 3600 (3601) logical unit) LDC options until the beginning of the next chain (that is, the first output request following an output request in which CNOTCOMPL is omitted).

For BTAM terminals, it indicates that the block sent as a result of the SEND command does not complete the message. If this option is omitted, the message will be regarded as complete when the SEND command has been fulfilled.

Logical record presentation

Each RECEIVE command results in one RU (or one chain of RUs if chain assembly is specified) being presented to the application program. An RU may consist of one or more logical records. If an RU contains more than one logical record, the records will be separated by new line (NL), interrecord separator (IRS), or transparent (TRN) characters. Except for LUTYPE4 devices, a logical record cannot be transmitted in more than one RU; the end of the RU is always the end of the logical record. Data from an LUTYPE4 may contain logical records that span RUs, in which case, chain assembly should be specified.

The system programmer can specify in the PCT, for specific application programs, that the application program will be presented with logical records instead of with RUs or chains. For those application programs for which this option is specified, each RECEIVE command results in one logical record being presented to the application program, regardless of whether chain assembly is specified or not.

If the logical records are separated by IRS or TRN characters, these are removed, and do not appear in the data. Therefore, a blank card will have a length of zero. If NL characters are used to separate the logical records, they are not removed, and the NL character from the end of each logical record appears at the end of the data. If the delimiter is a transparent (TRN) character, CICS will pass up to 256 bytes in one logical record. This logical record can contain any characters, including NL and IRS characters, all of which will be treated as data.

All communication features for logical units are still in operation, that is, notification of end-of-chain conditions, and (for batch logical units only) notification of end-of-data-set conditions and presentation of the inbound FMH at the beginning of a chain, still occurs.

If chain assembly has been specified, a logical record ends with a delimiter (NL, IRS, or TRN), or the end of the assembled chain. The end of chain notification occurs in the last logical record of the chain.

Definite response

The type of response requested by CICS for outbound data is generally determined by the system programmer in the PCT (DFHPCT TYPE=OPRGRP); it can be specified that all outbound data for an application program will require a definite response, or that exception response protocol is to be used, that is, a response will be made only if an error occurs.

If exception response protocol is used, an exception response may not be received and handled immediately after it arises.

The use of definite response protocol has some performance disadvantages, but may be necessary for some application programs. To provide a more flexible method of specifying the protocol to be used, the DEFRESP option is provided for use on the SEND command. One example of the use of this option is to request a definite response for every tenth output command, 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.

Function management header (FMH)

A function management header (FMH) is a field that can be included at the beginning of an input or output message. It is used to convey information about the message and how it should be handled. For some logical units, the use of an FMH is mandatory, for others it is optional, and in some cases FMHs cannot be used at all.

For output, the FMH can be built by the application program or by CICS. For input, the FMH can be passed to the application program or it can be suppressed by CICS.

The FMH option of the SEND command is used to specify that the application program will provide the FMH in the data to be transmitted.

Note: If the FMH option is used, the FMH data must conform to the SNA standards for FMH data, otherwise abend ATCY or unpredictable results can occur.

The ATTACHID option specifies a set of values that CICS puts into an LU6 attach FMH which is concatenated ahead of the user data.

Further information about FMHs is given in the CICS guides for the subsystems. (See the "CICS/MVS 2.1.2 library" on page vii.)

Inbound FMH: An application program can request notification when an FMH is included in the data received from a batch logical unit.

Whether or not inbound FMHs will be passed to the application program is specified in the INBFMH attribute of the PROFILE definition, when using RDO, or in the INBFMH operand of the DFHPCT TYPE=ENTRY system macro. It can be specified that no inbound FMHs will be passed, or that only the FMH at the end of the data set will be passed, or that all inbound FMHs will be passed.

If inbound FMHs are to be passed to the application program, a HANDLE CONDITION INBFMH command will allow control to be passed to a user-written routine whenever an inbound FMH is received. These user-written routines can investigate the contents of the FMH and take some action depending on, for example, the device from which the data has come. The contents of the FMH can be accessed also by means of the EIBFMH field of the EIB.

If an inbound FMH, containing an attach FMH, is passed to the application program, the attach FMH can be removed as long as this has been allowed for by the system programmer in the PCT. The values of the attach FMH may be examined by using the EXTRACT ATTACH command.

When input data is received as a chain of RUs, only the first (or only) RU of the chain is preceded by an FMH.

Outbound FMH: When CICS builds the outbound FMH, the application must reserve the first 3 bytes of the message for the FMH. When sending output data to a logical unit, use the FMH option of the SEND command when the application builds the FMH, but omit the option when CICS builds the FMH.

Unsolicited input

If unsolicited input arrives from a logical unit, it is queued and used to satisfy future input requests for that logical unit. However, for 3270 logical units, unsolicited input will be discarded if the PUNSOL operand is specified in the DFHSG PROGRAM=TCP system macro.

Bracket protocol (LAST option)

Bracket protocol prevents the interruption of a transaction between CICS and a logical unit. A bracket can, generally, be begun either by CICS or by the logical unit, or ended only by CICS unless it is for an LU6.1 or LU6.2 Logical Unit, in which case the logical unit can end it. A bracket also can 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 some logical units. The use of brackets is usually transparent 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 SEND command. The last output request is defined as either the last SEND command specified for a task without chain control; or as the output request that transmits the FOC or OC 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 FOC or OC is also specified.

Including a FREE command after a SEND command with the LAST option may be useful if the transaction does not terminate immediately after issuing the SEND command. This allows another transaction to be initiated from the LU or from CICS.

Suspend a task (WAIT SIGNAL)

```
WAIT SIGNAL

Condition: SIGNAL
```

This command is used, for a principal facility only, to suspend a task until a SIGNAL condition occurs. Some logical units can interrupt the normal flow of data to the application program by a SIGNAL data-flow-control command to CICS, signaling an attention, which in turn causes the SIGNAL condition to occur.

The HANDLE CONDITION SIGNAL command will cause a branch to an appropriate user-written routine when an attention is received.

Terminate a session (ISSUE DISCONNECT)

```
ISSUE DISCONNECT
SESSION(name)

Conditions: NOTALLOC, TERMERR
```

This command is used to terminate a session between CICS and a logical unit, but only if the system programmer has specified RELREQ=(,YES) in the DFHTCT TYPE=TERMINAL system macro for the logical unit, or DISCREQ=YES in DEFINE TYPETERM for RDO.

VTAM application routing (ISSUE PASS)

```
ISSUE PASS
LUNAME(name|data-area)
[FROM(data-area) LENGTH(data-value)]

Conditions: INVREQ, LENGERR
```

This command is used to disconnect the terminal from CICS after the task has terminated, and transfer it to the VTAM application defined in the LUNAME option.

This command requires that AUTH=PASS is coded in the VTAM APPL macro for the CICS system that owns the terminal, and DISCREQ=YES in DEFINE TYPETERM or RELREQ=(,YES) in the DFHTCT TYPE=TERMINAL macro for any terminal where this function might be used.

If the LUNAME specified is the name of another CICS system, you can use the EXTRACT LOGONMSG command to access the data referred to by this command.

Because of a VTAM limitation, the maximum length of the user data is restricted to 255 bytes.

Note: There is a SIT operand (`CLSDSTP=NOTIFY |NONOTIFY`) that allows you to have the node error program (NEP) notified of whether the PASS was successful or not. The NEP can be coded to reestablish a session ended by an unsuccessful PASS. For full details of how this is done, see the section on the NEP in the *CICS/MVS Customization Guide*.

Sync point processing (ISSUE PREPARE)

```
ISSUE PREPARE
[CONVID(data-area)|SESSION(name)]

Conditions: INVREQ, NOTALLOC
```

The ISSUE PREPARE command applies only to distributed transaction processing over LUTYPE6.2 links. It enables a sync point initiator to prepare a sync point slave for syncpointing by sending only the first flow (prepare-to-commit) of the sync point exchange. Depending on the reply from the sync point slave, the initiator can proceed with the sync point by issuing a SYNCPOINT command, or initiate back out by issuing a SYNCPOINT ROLLBACK command. For further details, see the *CICS/MVS Intercommunication Guide*.

Receipt of VTAM logon data (EXTRACT LOGONMSG)

```
EXTRACT LOGONMSG
{INTO(data-area)|SET(ptr-ref)}
LENGTH(data-area)
```

This command is used to access VTAM logon data. This data may have been specified by the terminal operator at logon or in the ISSUE PASS command, for example. This data is only available if LGNMSG=YES is specified in the SIT. The data can only be extracted once. It is possible to force the first transaction that runs on the terminal to be that which issues EXTRACT LOGONMSG by using the GMTRAN option in the SIT.

All the logon data will be extracted and its length placed in the field specified by the LENGTH option. Because the LENGTH option cannot be used to limit the amount of data extracted, it is recommended that a field of 256 bytes is always used for this option.

Return a facility to CICS (FREE)

```
FREE
[CONVID(data-area)|SESSION(name)]

Conditions: INVREQ, NOTALLOC
```

This command is used to return a facility (a principal facility or a previously allocated alternate facility) to CICS when a transaction owning it no longer requires it. The facility then can be allocated for use by other transactions.

If you omit to specify either SESSION or CONVID, the principal facility will be freed. Facilities not freed explicitly will be freed by CICS when the task terminates.

If you are running EDF in dual terminal mode and the transaction issues the FREE command, EDF will be switched off without warning.

TCAM-supported terminals and logical units

Because TCAM permits many applications to share a single network, the CICS-TCAM interface supports data streams rather than specific terminals or logical units.

Operations for terminals supported by TCAM use the same options as the terminals supported by other access methods. With the exception of the BUFFER option for the 3270, all options applicable for input operations are supported by CICS-TCAM. However, the exceptional conditions ENDINPT and EOF will not occur.

All output requests are the same for TCAM as for other CICS supported terminals, except that:

- The ISSUE RESET command cannot be used
- The ISSUE COPY and ISSUE PRINT commands for the 3270 cannot be used
- The DEST option is available on the SEND command, in addition to other appropriate options.

With the exception of 3650 Logical Units, operations for logical units supported by TCAM use the same options as logical units supported by VTAM.

BTAM programmable terminals

When BTAM is used by CICS for programmable binary synchronous communication line management, CICS initializes the communication line with a BTAM read initial (TI); the terminal response must be a write initial (TI) or the equivalent. If an application program makes an input request, CICS issues a read continue (TT) to that line; if the application program makes an output request, CICS issues a read interrupt (RVI) to that line. If end of transmission (EOT) is not received on the RVI, CICS issues a read continue (TT) until the EOT is received.

When TCAM is used, all of this line control is handled by the message control program (the MCP) rather than by CICS.

The programmable terminal response to a read interrupt must be "end of transmission" (EOT). The EOT response may, however, be preceded by writes, in order to exhaust the contents of output buffers; this is provided the input buffer size is not exceeded by this data. The input buffer size is specified by the system programmer during preparation of the TCT. CICS issues a read continue until it receives an EOT, or until the input message is greater than the input buffer (an error condition).

After receiving an EOT, CICS issues a write initial (TI) or the equivalent (depending on the type of line). The programmable terminal response must be a read initial (TI) or the equivalent.

If the application program makes another output request, CICS issues a write continue (TT) to that line. If the application program makes an input request after it has made an output request, CICS turns the line around with a write reset (TR). (CICS does not recognize a read interrupt.)

To ensure that binary synchronous terminals (for example, System/370, 1130, 2780) remain coordinated, CICS processes the data collection or data transmission transaction on any line to completion, before polling other terminals on that line.

The programmable terminal actions required for the above activity, with the corresponding user application program commands and CICS actions, are summarized in Figure 27 on page 231.

Automatically initiated transactions attached to a device will cause message

```
DFH2503 AUTO OUTPUT HAS BEEN REQ,
        PLEASE PREPARE TO RECEIVE
```

to be sent to the device which must be prepared to receive it.

Input data is deblocked to ETX, ETB, RS, and US characters. These characters are moved with the data but are not included in the data length. Characters such as NL (new line), CR (carriage return), LF (line feed), and EM are included as data in a CICS application program.

Teletypewriter programming

The teletypewriter (World Trade only) uses two different control characters for print formatting, as follows:

```
<   carriage return, (X'22' in ITA2
    code or X'15' in EBCDIC)

≡   line feed, (X'28' in ITA2 code
     or X'25' in EBCDIC)
```

The character < should always be used first otherwise following characters (data) may be printed while the typebar is moving to the left.

Message format

Message begin: To start a message on a new line at the left margin, the message text must begin with X'1517' (EBCDIC). CICS recognizes the X'17' and changes it to X'25' (X'17' is an IDLE character).

Message body: To write several lines with a single transmission, the lines must be separated by X'1525', or if multiple blank lines are required, by X'152525...25'.

Message end before next input: To allow input of the next message on a line at the left margin, the preceding message must end with X'1517'. CICS recognizes X'15' and changes the character following it to X'25'.

Message end before next output: In the case of two or more successive output messages, the "message begin" and the "message end" look the same; that is X'1517', except for the last message (see above). To make the "message end" of the preceding message distinguishable from the "message begin" of the next message, the next to last character of the "message end" must not be X'15'.

Message length

Messages for teletypewriter terminals should not exceed a length of about 3000 bytes or approximately 300 words.

Connection through VTAM

Both the TWX Model 33/35 Common Carrier Teletypewriter Exchange and the WTTY Teletypewriter (World Trade only) can be connected to CICS through BTAM, or through VTAM using NTO.

If a device is connected through VTAM using NTO, the protocols used are the same as for the 3767 Logical Unit, and the application program can make use of these protocols (for example, HANDLE CONDITION SIGNAL). However, the data stream is not translated to a 3767 data stream but remains as that for a TWX/WTTY.

Display device operations

Besides the standard terminal control commands for sending and receiving data, several additional commands and lists are provided for use with display devices such as the 3270.

The commands are:

- Print displayed information (ISSUE PRINT).
- Copy displayed information (ISSUE COPY).
- Erase all unprotected fields (ISSUE ERASEAUP).
- Input operation without data (RECEIVE).
- Handling attention identifiers (HANDLE AID).

The lists are:

- Standard Attention Identifier List (DFHAID).
- Standard Attribute and Printer Control Character List (DFHBMSCA).

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 by CICS table generation. These values can be obtained by means of the ASSIGN command, described in "Chapter 1.6. Access to system information" on page 51.

The ERASE option should always be included in the first SEND command to clear the screen and format it according to the transmitted data. This first SEND with ERASE will select also the screen size to be used, as specified in the PCT and TCT. If ERASE is omitted, the screen size will be the same as its previous setting, which may be incorrect.

Use of the CLEAR key outside of a transaction will set the screen to its default size.

User Application Program Command	CICS[1] Actions	Programmable Terminal Action
	Read initial (TI)	Write initial (TI)
RECEIVE	Read continue (TT)	Write continue (TT)
SEND	Read interrupt (RVI)[2]	Write reset (TR) or
	Read continue (TT)[3]	Write continue Write reset
	Write initial (TI)	Read initial (TI)
SEND	Write continue (TT)	Read continue (TT)
RECEIVE	Write reset (TR)[4]	Read continue (TT)
	Read initial (TI)	Write initial (TI)

[1] CICS issues the macro shown, or, if the line is switched, the equivalent. The user-written programmable terminal program must issue the equivalent of the BTAM operation shown.

[2] An RVI sequence is indicated by the DECFLAGS field of the data event control block (DECB) being set to X'02' and a completion code of X'7F' being returned to the event control block (ECB).

[3] The read continue is issued only if the EOT character is not received on the read interrupt.

[4] Write reset is issued only for point-to-point terminals.

Figure 27. BTAM programmable terminal programming

Print displayed information (ISSUE PRINT)

If the 3270 print request facility is included in the terminal control program at CICS system generation, the ISSUE PRINT command will cause the displayed data to be printed on the first available, print-request-eligible printer.

For a BTAM-supported 3270, this is a printer on the same control unit.

For a 3270 Logical Unit or a 3650 host-conversational (3270) Logical Unit, it is a printer defined by the PRINTTO or ALTPRT operands of the DFHTCT TYPE=TERMINAL system macro, by RDO, or by a printer supplied by the autoinstall user program.

For a 3270-display Logical Unit with the PTRADAPT feature (LUTYPE2 specified in the TRMTYPE operand and PTRADAPT specified in the FEATURE operand of the DFHTCT TYPE=TERMINAL system macro) used with a 3274 or 3276, it is a printer allocated by the printer authorization matrix. See *An Introduction to the IBM 3270 Information Display System.*

For a 3790 (3270-display) Logical Unit, it is a printer allocated by the 3790.

For a printer to be available, it must be in service and not currently attached to a task.

For a BTAM printer to be eligible, it must be attached to the same control unit as the display, must have a buffer capacity equal to or greater than that of the display, and must have FEATURE=PRINT specified in the associated DFHTCT TYPE=TERMINAL system macro.

For a 3270 logical unit to be eligible, it must have been specified by the PRINTTO or ALTPRT operand of the DFHTCT TYPE=TERMINAL system macro, by RDO, or by a printer supplied by the autoinstall user program, and it must have the correct buffer capacity; FEATURE=PRINT is not necessary. If COPY is specified with the ALTPRT or PRINTTO operands, the printer must be on the same control unit.

If an ISSUE PRINT command is executed, then the printer involved must be owned by the same CICS system that owns the terminal that is running the transaction.

For some 3270 displays, it is possible also to print the displayed information without using CICS. See *An Introduction to the IBM 3270 Information Display System.*

Copy displayed information (ISSUE COPY)

The ISSUE COPY command is used to copy the format and data contained in the buffer of a specified terminal into the buffer of the terminal that started the transaction. This command cannot be used for an LUTYPE2. Both terminals must be attached to the same remote control unit. The terminal whose buffer is to be copied is identified in the TERMID option. If the terminal identifier is invalid, that is, it does not exist in the TCT, the TERMIDERR condition will occur. The copy function to be performed is defined by the copy control character (CCC) specified in the CTLCHAR option of the ISSUE COPY command.

The WAIT option of the ISSUE COPY command ensures that the operation has been completed before control is returned to the application program.

Erase all unprotected fields (ISSUE ERASEAUP)

The ISSUE ERASEAUP command is used to erase all unprotected fields of a 3270 buffer, by the following actions:

1. All unprotected fields are cleared to nulls (X'00').
2. The modified data tags (MDTs) in each unprotected field are reset to zero.
3. The cursor is positioned to the first unprotected field.
4. The keyboard is restored.

The WAIT option of the ISSUE ERASEAUP command ensures that the operation has been completed before control is returned to the application program.

Input operation without data (RECEIVE)

The RECEIVE command with no options causes input to take place and the EIB to be updated. However, data received by CICS is not passed on to the application program and is lost. A wait will be 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 in the EIB. The cursor address is in the form of a halfword binary value and remains until updated by a new input operation.

Attention identifier (EIBAID) – For every terminal control (or BMS) input operation associated with a display device, an attention identifier (AID) is placed in field EIBAID in the EIB. The AID indicates which method 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, the light pen, and so on. The field contents remain unaltered until updated by a new input operation. Field 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, the HANDLE AID command can be used to pass control to specified labels when the AIDs are received. The standard attention identifier list and the HANDLE AID command are described in the next two sections.

Standard attention identifier list (DFHAID)

The standard attention identifier list, DFHAID, simplifies testing the contents of the EIBAID field. The following list is obtained by copying DFHAID into the application program and shows the symbolic name for the attention identifier (AID) and the corresponding 3270 function.

Constant	**Meaning**
DFHENTER	ENTER key
DFHCLEAR	CLEAR key
DFHPA1-DFHPA3	PA1-PA3 keys
DFHPF1-DFHPF24	PF1-PF24 keys
DFHOPID	Operid or MSR
DFHMSRE	Extended (standard) MSR
DFHTRIG	Trigger field
DFHPEN	Light pen attention

For COBOL users, the list consists of a set of 01 statements that must be copied into the working-storage section. For PL/I users, the list consists of DECLARE statements defining elementary character variables.

Handling attention identifiers (HANDLE AID)

```
HANDLE AID
option[(label)]...
```

This command 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, any data received in addition to the AID has been passed to the application program.

In the absence of a HANDLE AID command, control returns to the application program at the point immediately following the input command. You can suspend the HANDLE AID command by means of the PUSH and POP commands as described in "Chapter 1.5. Exceptional conditions" on page 43.

In an assembler language application program, a branch to a label caused by receipt of an AID, for which a HANDLE AID command is active, will restore the registers in the application program to their values in the program at the point where the command that received the AID is issued. On MVS/XA, the addressing mode will be set to that in effect at the point where the HANDLE AID command is issued.

No more than 16 options are allowed in the same command.

A HANDLE AID command will take precedence over a HANDLE CONDITION command. The HANDLE CONDITION command is described in "Chapter 1.5. Exceptional conditions" on page 43. If an AID is received during an input operation for which a HANDLE AID command is active, control will pass to the label specified in that command regardless of any conditions that may have occurred (but which did not stop receipt of the AID).

A print key specified in the SIT will take precedence over a HANDLE AID command.

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)
- CLRPARTN (for the key of that name)
- ENTER (for the key of that name)
- LIGHTPEN (for a light pen attention)
- OPERID (for the operator identification card reader, the magnetic slot reader (MSR), or the extended MSR)
- PA1, PA2, or PA3 (any of the program access keys)
- PF1 through PF24 (any of the program function keys)
- TRIGGER (for a trigger field attention).

The HANDLE AID command for a given AID applies only to the program in which it is specified, remaining active until the program is terminated, or until another HANDLE AID command for the same AID is encountered, 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 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. For more information about logical levels see "Chapter 4.4. Program control" on page 289.

If no HANDLE AID command is active for any PA key, any PF key, or the CLEAR key, but one is active for ANYKEY, control will be passed to the label specified for ANYKEY. A HANDLE AID command for an AID overrides the HANDLE AID ANYKEY command for that AID.

The following example shows a HANDLE AID command that specifies one label for the PA1 key, a second label for PA2 and PA3, all of the PF keys except PF10, and the CLEAR key. If a PF10 AID is received, control returns to the application program at the instruction immediately following the input command.

```
EXEC CICS HANDLE AID PA1(LAB1)
     ANYKEY(LAB2) PF10
```

If a task is initiated from a terminal by means of an AID, the first RECEIVE command in the task will not read from the terminal but will copy 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.

A BMS RECEIVE MAP command with the FROM option will not cause a HANDLE AID command to be invoked because no terminal input is involved.

Standard attribute and printer control character list (DFHBMSCA)

The standard list DFHBMSCA simplifies the provision of field attributes and printer control characters. The list is obtained by copying copy book DFHBMSCA into the application program. The symbolic names for the various combinations of attributes and control characters are given below. Combinations other than shown must be generated separately.

Constant	Meaning
DFHBMPEM	Printer end-of-message
DFHBMPNL	Printer new line
DFHBMASK	Autoskip
DFHBMUNP	Unprotected
DFHBMUNN	Unprotected and num
DFHBMPRO	Protected
DFHBMBRY	Bright
DFHBMDAR	Dark
DFHBMFSE	MDT set
DFHBMPRF	Protected and MDT set
DFHBMASF	ASKP and MDT set
DFHBMASB	Auto and bright
DFHBMPSO	Shift out value X'0E'
DFHBMPSI	Shift in value X'0F'
DFHBMEOF	Field erased
DFHBMDET	Field detected
DFHSA[1]	Set attribute (SA) order
DFHERROR	Error code
DFHCOLOR[1]	Color
DFHPS[1]	PS
DFHHLT[1]	Highlight
DFH3270[1]	Base 3270 field attribute
DFHVAL	Validation
DFHOUTLN	Field outlining attr code
DFHBKTRN	Background transp attr 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
DFHBASE[1]	Base PS
DFHDFHI[1]	Normal
DFHBLINK	Blink
DFHREVRS	Reverse video
DFHUNDLN	Underscore
DFHMFIL[2]	Mandatory fill
DFHMENT[2]	Mandatory enter
DFHMFE	Mandatory fill and mandatory enter
DFHMT	Trigger
DFHMFT	Mandatory fill and trigger
DFHMET	Mandatory enter and trigger
DFHUNNOD	Unprotected non-display non-print non-detectable MDT
DFHUNIMD	Unprotected intensify light pen detectable MDT
DFHUNNUM	Unprotected numeric MDT
DFHUNINT	Unprotected numeric intensify light pen detectable MDT
DFHUNNON	Unprotected numeric intensify light pen non-display non-print non-detectable MDT
DFHPROTI	Protected intensify light pen detectable
DFHPROTN	Protected non-display non-print non-detectable
DFHMFET	Mandatory fill and mandatory enter and trigger
DFHDFFR	Default outline
DFHUNDER	Under
DFHRIGHT	Right
DFHOVER	Overline
DFHLEFT	Left
DFHBOX	Left+over+right+under lines
DFHSOSI	SOSI=yes
DFHTRANS	Background transparency
DFHOPAQ	No background transparency

[1] For text processing only.
Use for
constructing embedded set attribute orders in user text

[2] Cannot be used in set attribute orders

For assembler language users, the list consists of a set of EQU statements. For COBOL users, the list consists of a set of 01 statements that must be copied into the working storage section. For PL/I users, the list consists of DECLARE statements defining elementary character variables.

The symbolic name DFHDFT must be used 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, the symbolic names DFHDFCOL, DFHBASE, or DFHDFHI should be used.

Standard CICS terminal support (BTAM or TCAM)

```
RECEIVE
{INTO(data-area)|SET(ptr-ref)}
LENGTH(data-area)
[MAXLENGTH[(data-value)]]
[NOTRUNCATE]

Conditions: LENGERR, NOTALLOC


SEND
FROM(data-area)
LENGTH(data-value)
[DEST(name)]
[WAIT]


CONVERSE
FROM(data-area)
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
TOLENGTH(data-area)
[DEST(name)]
[MAXLENGTH[(data-value)]]
[NOTRUNCATE]

Condition: LENGERR


ISSUE RESET


ISSUE DISCONNECT
```

These commands can be used by all terminals supported by CICS that are not dealt with separately in the following sections.

LUTYPE4 logical unit

RECEIVE
```
{INTO(data-area)|SET(ptr-ref)}
LENGTH(data-area)
[MAXLENGTH[(data-value)]]
[NOTRUNCATE]
```

Conditions: EOC, EODS, INBFMH, LENGERR, SIGNAL, TERMERR

SEND
```
FROM(data-area)
LENGTH(data-value)
[WAIT]
[INVITE|LAST]
[CNOTCOMPL|DEFRESP]
[FMH]
```

Conditions: IGREQCD, SIGNAL, TERMERR

CONVERSE
```
FROM(data-area)
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
TOLENGTH(data-area)
[DEFRESP]
[MAXLENGTH[(data-value)]]
[FMH]
[NOTRUNCATE]
```

Conditions: EOC, EODS, IGREQCD, INBFMH, LENGERR, SIGNAL, TERMERR

WAIT SIGNAL

Conditions: SIGNAL, TERMERR

ISSUE DISCONNECT

Conditions: SIGNAL, TERMERR

LUTYPE6.1 logical unit

RECEIVE
```
[SESSION(name)]
{INTO (data-area)|SET(ptr-ref)}
LENGTH(data-area)
[MAXLENGTH[(data-value)]]
[NOTRUNCATE]
```

Conditions: EOC, INBFMH, NOTALLOC, LENGERR, SIGNAL, TERMERR

SEND
```
[SESSION(name)]
[WAIT]
[INVITE|LAST]
[ATTACHID(name)]
[FROM(data-area)]
[LENGTH(data-area)]
[FMH]
[DEFRESP]
```

Conditions: CBIDERR, NOTALLOC, SIGNAL, TERMERR

CONVERSE
```
[SESSION(name)]
[ATTACHID(name)]
[FROM(data-area)]
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
TOLENGTH(data-area)
[MAXLENGTH[(data-value)]]
[NOTRUNCATE]
[FMH]
[DEFRESP]
```

Conditions: CBIDERR, EOC, INBFMH, LENGERR, NOTALLOC, SIGNAL, TERMERR

ALLOCATE
```
{SESSION(name)|SYSID(name)}
[PROFILE(name)]
[NOQUEUE|NOSUSPEND]
```

Conditions: CBIDERR, EOC, INVREQ, SESSBUSY, SESSIONERR, SYSBUSY, SYSIDERR

LUTYPE6.1 Logical Unit (Continued)

```
BUILD ATTACH
ATTACHID(name)
[PROCESS(name)]
[RESOURCE(name)]
[RPROCESS(name)]
[RRESOURCE(name)]
[QUEUE(name)]
[IUTYPE(name)]
[DATASTR(name)]
[RECFM(name)]


EXTRACT ATTACH
[ATTACHID(name)|SESSION(name)]
[PROCESS(data-area)]
[RESOURCE(data-area)]
[RPROCESS(data-area)]
[RRESOURCE(data-area)]
[QUEUE(data-area)]
[IUTYPE(data-area)]
[DATASTR(data-area)]
[RECFM(data-area)]

Conditions: CBIDERR, INVREQ,
NOTALLOC


EXTRACT TCT
NETNAME(name)
{SYSID(data-area)|TERMID(data-area)}

Condition: INVREQ


FREE
[SESSION(name)]

Conditions: INVREQ, NOTALLOC,


POINT
[SESSION(name)]

Condition: NOTALLOC


WAIT SIGNAL


WAIT TERMINAL
[SESSION(name)]

Conditions: NOTALLOC, SIGNAL
```

LUTYPE6.1 Logical Unit (Continued)

```
ISSUE DISCONNECT
[SESSION(name)]

Conditions: NOTALLOC, TERMERR


ISSUE SIGNAL
[SESSION(name)]

Conditions: NOTALLOC, TERMERR
```

The ALLOCATE command is used to acquire an alternate facility and to select optionally a set of terminal control processing options. If SYSID is specified, CICS will make available to the application program one of the sessions associated with the named system. The name of this session can be obtained from field EIBRSRCE in the EIB. If SESSION is specified, CICS will make the named session available.

The BUILD ATTACH command is used to specify a set of values to be placed in the named attach header control block. This control block contains values that are to be sent in an LU6 attach FMH which is constructed by CICS, and is sent only when a SEND ATTACHID or CONVERSE ATTACHID command is executed. The specified values override existing values in the control block; unspecified values are set to default values.

The EXTRACT ATTACH command is used to retrieve a set of values held in an attach header control block or that have been built previously. This control block contains values received in an attach FMH or that have been built previously.

The EXTRACT TCT command is used to allow the eight-character VTAM network name for a terminal or logical unit to be converted into a corresponding four-character name by which it is known in the local CICS system.

The FREE command is used to return a facility to CICS when a transaction owning it no longer requires it. The facility can then be allocated for use by other transactions. A facility can be freed only when it is in free mode (EIBFREE set to X'FF').

The POINT command is used to obtain information about a named facility, such as whether it owns the given facility. All these commands, except EXTRACT TCT, WAIT SIGNAL, ISSUE SIGNAL, and ISSUE DISCONNECT, can be used on an MRO session. For more information on MRO and IRC see the *CICS/MVS Intercommunication Guide*.

Session status information

This information consists of several fields that contain application-oriented and session-oriented information when an LU6.1 session is in progress. These fields are located in the EIB.

Session status information is set to zeros at the start of execution of every command and is updated whenever a RECEIVE or CONVERSE command naming an LU6.1 session is executed. If the information is to be retained across the execution of several commands, the user must take steps to preserve it.

Application-oriented information

The application-oriented information determines the action taken by function processing logic. The information consists of, for example, indicators (such as end-of-chain), an attach header, and user data.

The user data is moved to an area specified in the application program; alternatively the address of the user data is passed to the application program.

The indicators, together with an attach header indicator, are passed to the application program in the EIB. The EXTRACT ATTACH command (described earlier in the chapter) can be used to process the attach header data if such data exists.

The following application-oriented fields, each one byte in length, appear in the EIB: EIBATT, EIBEOC, and EIBFMH.

Session-oriented information

The session-oriented information determines the action taken by session-handling logic, for example, sync point requested. This information is available to the application program in fields EIBSYNC, EIBFREE, EIBRECV, and EIBSIG in the EIB, and should be processed in that order, before further operations, such as SEND, RECEIVE, CONVERSE, or FREE, are performed on the session.

LUTYPE6.2 logical unit (VTAM only)

```
RECEIVE
[CONVID(data-area)]
{INTO(data-area)|SET(ptr-ref)}
LENGTH(data-area)
[MAXLENGTH[(data-value)]]
[NOTRUNCATE]

Conditions: EOC, LENGERR,
NOTALLOC, SIGNAL, TERMERR


SEND
[CONVID(data-area)]
[FROM(data-area)]
[LENGTH(data-value)]
[INVITE|LAST]
[CONFIRM|WAIT]

Conditions: INVREQ, LENGERR,
NOTALLOC, SIGNAL, TERMERR


CONVERSE
[CONVID(data-area)]
FROM(name)
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
TOLENGTH(data-area)
[MAXLENGTH[(data-value)]]
[NOTRUNCATE]

Conditions: EOC, LENGERR, NOTALLOC,
SIGNAL, TERMERR
```

```
ALLOCATE
SYSID(name)
[PROFILE(name)]
[NOQUEUE|NOSUSPEND]

Conditions: CBIDERR, INVREQ,
SESSBUSY, SYSBUSY, SYSIDERR


FREE
[CONVID(data-area)]

Conditions: INVREQ, NOTALLOC,


CONNECT PROCESS
CONVID(data-area)
PROCNAME(data-area)
PROCLENGTH(data-value)
SYNCLEVEL(data-value)
[PIPLIST(data-area)
  PIPLENGTH(data-value)]

Conditions: INVREQ, NOTALLOC,
LENGERR


EXTRACT PROCESS
[PROCNAME(data-area)
  PROCLENGTH(data-area)]
[CONVID(data-area)]
[SYNCLEVEL(data-area)]
[PIPLIST(ptr-ref)
  PIPLENGTH(data-area)]

Conditions: INVREQ, NOTALLOC


ISSUE ABEND
[CONVID(data-area)]

Conditions: INVREQ, NOTALLOC,
TERMERR


ISSUE CONFIRMATION
[CONVID(data-area)]

Conditions: INVREQ, NOTALLOC,
TERMERR
```

```
ISSUE ERROR
[CONVID(data-area)]

Conditions: INVREQ, NOTALLOC,
TERMERR


ISSUE SIGNAL
[CONVID(data-area)]

Conditions: NOTALLOC, TERMERR


WAIT
CONVID(data-area)]

Condition: NOTALLOC
```

A program written to communicate across an LU6.1 link can be migrated to communicate across an LU6.2 link. For further details see the *CICS/MVS Intercommunication Guide*.

Synchronization levels

LU6.2 application programs can run at three synchronization levels, as follows:

0 No synchronization capability

1 Commit only synchronization

2 Full synchronization.

For further details of synchronization levels, see the *CICS/MVS Intercommunication Guide.*

The ALLOCATE command is used to acquire an alternate facility and to select optionally a set of terminal control processing options. CICS will make available to the application program one of the sessions associated with the named system. CICS returns, in EIBRSRCE in the EIB, the 4-byte CONVID (conversation identifier) that the application program uses in all subsequent commands that relate to the conversation.

The CONNECT PROCESS command allows an application to specify a process name and synchronization level to be passed to CICS and used when the remote process (or transaction) is attached.

The EXTRACT PROCESS command allows an application program to access conversation related data that is 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 ISSUE ABEND command allows an application program to abend the conversation with the connected LU6.2 system.

The ISSUE CONFIRMATION command allows an application program to respond positively when the CONFIRM option has been specified on a SEND command executed by a process in a connected LU6.2 system.

The ISSUE ERROR command allows an application program to inform a process in a connected LU6.2 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 LU6.2 system.

The WAIT CONVID command allows an application program to ensure that any accumulated application data from a SEND or CONNECT PROCESS command is transmitted to the connected LU6.2 process before further processing continues.

Session-oriented information

For LU6.2 programs, this information is available in fields EIBSYNC, EIBSYNRB, EIBFREE, EIBRECV, EIBSIG, EIBCONF, EIBERR, and EIBERRCD in the EIB, and should be processed before further operations, such as SEND, RECEIVE, CONVERSE, or FREE are performed on the session.

Guidance on writing applications for LU6.2, including the use of these fields and other fields in the EIB, is given in the *CICS/MVS Intercommunication Guide*.

System/3

```
RECEIVE
{INTO(data-area)|SET(ptr-ref)}
LENGTH(data-area)
[MAXLENGTH[(data-value)]]
[NOTRUNCATE]
[ASIS]

Condition: LENGERR


SEND
FROM(data-area)
LENGTH(data-value)
[DEST(name)]
[WAIT]
[ASIS]
[CNOTCOMPL]


CONVERSE
FROM(data-area)
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
TOLENGTH(data-area)
[DEST(name)]
[MAXLENGTH[(data-value)]]
[NOTRUNCATE]

Condition: LENGERR
```

System/370

Support and command syntax as for System/3.

System/7

```
RECEIVE
{INTO(data-area)|SET(ptr-ref)}
LENGTH(data-area)
[PSEUDOBIN]¹
[ASIS]

Condition: LENGERR


SEND
FROM(data-area)
LENGTH(data-value)
[DEST(name)]
[WAIT]
[PSEUDOBIN]¹
[ASIS]


CONVERSE
FROM(data-area)
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
TOLENGTH(data-area)
[DEST(name)]

Condition: LENGERR


ISSUE RESET


ISSUE DISCONNECT

¹ Start-stop only
```

Transactions are normally initiated from the System/7 by issuing a four-character transaction code which is transmitted in BCD mode. Pseudobinary mode can be used only while communicating with an active CICS transaction; it cannot be used to initiate the transaction. The message length is given as the number of words to be transmitted (not as the number of characters).

When a transaction is initiated on a System/7, CICS services that System/7 only for the duration of the transaction; that is, to ensure efficient use of the line, any other System/7s on the same line are locked out for the duration of the transaction. CICS application programs for the multipoint System/7 should be designed with the shortest possible execution time.

The first word (two characters) of every message received by the System/7 must be an identification word, except words beginning with "@"(X'20') which are reserved by CICS.

When the PSEUDOBIN option is specified, the length of the data area provided by the application program must be at least twice that of the data to be read.

In the case of a System/7 on a dial-up (switched) line, the System/7 application program must, initially, transmit a four-character terminal identification. (This terminal identification is generated during preparation of the TCT through use of the DFHTCT TYPE=TERMINAL, TRMIDNT=parameter specification.) CICS responds with either a "ready" message, indicating that the terminal identifier is valid and that the System/7 may proceed as if it were on a leased line, or an INVALID TERMINAL IDENTIFICATION message, indicating that the terminal identifier sent by the System/7 did not match the TRMIDNT=parameter specified.

Whenever CICS initiates the connection to a dial-up System/7, CICS writes a null message, consisting of three idle characters, prior to starting the transaction. If there is no program resident in the System/7 capable of supporting the Asynchronous Communication Control Adapter (ACCA), BTAM error routines cause a data check message to be recorded on the CICS (host) system console. This is normal if the task initiated by CICS is to IPL the System/7. Although the data check message is printed, CICS ignores the error and continues normal processing. If a program capable of supporting the ACCA is resident in the System/7 at the time this message is transmitted, no data check occurs.

When a disconnect is issued to a dial-up System/7, the "busy" bit is sometimes left on in the interrupt status word of the ACCA. If the line connection is reestablished by dialing from the System/7 end, the 'busy' condition of the ACCA prevents message transmission from the System/7. To overcome this problem, the System/7 program must reset the ACCA after each disconnect and before message transmission is attempted. This can be done by issuing the following instruction:

```
PWRI   0,8,3,0    RESET ACCA
```

This procedure is not necessary when the line is reconnected by CICS (that is, by an automatically initiated transaction).

2260 display station

```
RECEIVE
{INTO(data-area)|SET(ptr-ref)}
LENGTH(data-area)
[MAXLENGTH[(data-value)]]
[NOTRUNCATE]
[LEAVEKB]

Condition: LENGERR


SEND
FROM(data-area)
LENGTH(data-value)
[CTLCHAR(data-value)]
[DEST(name)]
[LINEADDR(data-value)]
[WAIT]
[LEAVEKB]


CONVERSE
FROM(data-area)
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
TOLENGTH(data-area)
[MAXLENGTH[(data-value)]]
[NOTRUNCATE]
[CTLCHAR(data-value)]
[DEST(name)]
[LINEADDR(data-value)]

Condition: LENGERR


ISSUE RESET


ISSUE DISCONNECT
```

The LINEADDR option specifies on which line of a 2260 screen writing is to begin. A line number in the range 1 through 12 must be provided in the application program.

2265 display station

Support and command syntax for the 2265 is as for the 2260 Display Station except that a line number in the range 1 through 15 must be provided in the application program.

2741 communication terminal

```
RECEIVE
{INTO(data-area)|SET(ptr-ref)}
LENGTH(data-area)
[MAXLENGTH[(data-value)]]
[NOTRUNCATE]

Conditions: LENGERR, RDATT
(not TCAM)


SEND
FROM(data-area)
LENGTH(data-value)
[DEST(name)]
[WAIT]

Condition: WRBRK


CONVERSE
FROM(data-area)
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
TOLENGTH(data-area)
[MAXLENGTH[(data-value)]]
[NOTRUNCATE]
[DEST(name)]

Conditions: LENGERR, RDATT, WRBRK


ISSUE RESET


ISSUE DISCONNECT
```

Read attention

If the terminal operator presses the attention key on the 2741 after typing a message, it is recognized as a read attention if:

- Read attention support is generated into the system.
- The message is read by a RECEIVE command.

When this occurs, control is transferred to a CICS read attention exit routine, if it has been generated into the system. This routine is a skeleton program that can be tailored by the system programmer to carry out actions such as the following:

- Perform data analysis or modification on a read attention.
- Return a common response to the terminal operator following a read attention.

- Return a response and request additional input that can be read into the initial input area or into a new area.
- Request new I/O without requiring a return to the task to request additional input.

When the read attention exit routine is completed, control is returned to the application program at the address specified in the HANDLE CONDITION RDATT command. The return is made whenever one of the following occurs:

- The exit routine issues no more requests for input.
- The exit routine issues a RECEIVE request and the operator terminates the input with a carriage return. (If the operator terminates the input with an attention, the exit routine is reentered and is free to issue another RECEIVE request.)

If a HANDLE CONDITION RDATT command is not included in the application program or read attention support has not been generated, the attention is treated as if the return key had been pressed.

Write break

If the terminal operator presses the attention key on the 2741 while a message is being received, it is recognized as a write break if:

- Write break support is generated into the system by the system programmer.
- A HANDLE CONDITION WRBRK command is active in the application program.

When this occurs, the remaining portion of the message is not sent to the terminal. The write is terminated as though it were successful, and a new-line character (X'15') is sent to cause a carrier return. Control is returned to the application program at the address specified for the WRBRK condition.

If a HANDLE CONDITION WRBRK command is not included in the application program or if write break support has not been generated, the attention is treated as an I/O error.

2770 data communication system

Support and command syntax for the 2770 is as for System/3. The 2770 recognizes a read interrupt and responds by transmitting the contents of the I/O buffer. After the contents of the buffer have been transmitted, the 2770 responds to the next read continue with an EOT. If the I/O buffer is empty, the 2770 transmits an EOT. CICS issues a read interrupt and read continue to relinquish use of the line and to enable the application program to write to the 2770.

Input from a 2770 consists of one or more logical records. CICS provides one logical record for each read request to the application program. The size of a logical record cannot exceed the size of the I/O buffer. If the input spans multiple buffers, multiple reads must be issued by the application program.

The 2265 component of the 2770 Data Communication System is controlled by data stream characters, not BTAM macro instructions; appropriate screen control characters should be included in the output area.

For 2770 input, data is deblocked to ETX, ETB, RS, and US characters. These characters are moved with the data to the input area but are not included in the data length; characters such as NL, CR, and LF are passed in the input area as data.

2780 data transmission terminal

Support and command syntax for the 2780 is as for System/3. The 2780 recognizes a read interrupt and responds by transmitting the contents of the I/O buffer. After the contents of the buffer have been transmitted, the 2780 responds to the next read continue with an EOT. If the I/O buffer is empty, the 2780 transmits an EOT. CICS issues a read interrupt and read continue to relinquish use of the line and to enable the application program to write to the 2780.

Input from a 2780 consists of one or more logical records. CICS provides one logical record for each read request to the application program. The size of a logical record cannot exceed the size of the I/O buffer. If the input spans multiple buffers, multiple reads must be issued by the application program.

Output to a 2780 requires that the application program contains an appropriate "escape sequence" for component selection associated with the output message. (For programming details, see the publication *Component Description: IBM 2780 Data Transmission Terminal.*)

For 2780 input, data is deblocked to ETX, ETB, RS, and US characters. These characters are moved with the data to the input area but are not included in the data length; characters such as NL, CR, and LF are passed in the input area as data.

2980 general banking terminal system

```
RECEIVE
{INTO(data-area)|SET(ptr-ref)}
LENGTH(data-area)
[MAXLENGTH[(data-value)]]
[NOTRUNCATE]
PASSBK

Conditions: LENGERR, NOPASSBKRD


SEND
FROM(data-area)
LENGTH(data-value)
[DEST(name)]
{PASSBK|CBUFF}

Condition: NOPASSBKWR
```

Passbook control

All input and output requests to the passbook area of a 2980 are dependent on the presence of a passbook. The PASSBK option is used to specify the passbook area. The conditions NOPASSBKRD and NOPASSBKWR will occur on input and output requests respectively when a passbook is not present. These conditions can be handled by a HANDLE CONDITION command and appropriate handling routines.

If the passbook is present on an input request, the application program generally writes back to the passbook area to update the passbook. If the NOPASSBKWR condition occurs, CICS allows immediate output to the terminal. In a routine for the NOPASSBKWR condition, the application program should send an error message to the journal area of the terminal to inform the 2980 operator of this error condition. To allow the operator to insert the required passbook, CICS causes the transaction to wait 23.5 seconds before continuing.

On regaining control from CICS after sending the error message, the application program can attempt again to update the passbook when it has ensured that the print element is positioned correctly in the passbook area. This is generally accomplished by issuing two carrier returns followed by the number of tabs required to move the print element to the correct position. (See "The DFH2980 structure" later in the section.)

If the NOPASSBKWR condition occurs during the second attempt to write to the passbook area, the application program can send another error message or take some alternative action (for example, place the terminal "out of service").

The presence of the Auditor Key on a 2980 Administrative Station Model 2 is controlled by the SEND PASSBK command and may be used in a manner similar to that described above.

Output control

The unit of transmission for a 2980 is called a segment. A segment is equivalent to the buffer size of the 2972 Control Unit. However, for the passbook and journal areas, CICS allows an application program to send messages that exceed the buffer size. For the passbook area, the maximum length of message is limited to one line of a passbook to avoid spacing (indexing) past the bottom of the passbook. For the journal area, the maximum length of message is specified in the LENGTH option of the SEND command.

For example, consider a 2972 buffer size of 48 characters and a 2980 Teller Station Model 4 passbook print area of 100 characters/line. The application program can send a message of 100 characters to this area; CICS segments the message to adjust to the buffer size. The application program must insert the passbook indexing character (X'25') as the **last** character written in one output request to the passbook area. This is done to control passbook indexing and thereby achieve positive control of passbook presence.

If a message contains embedded passbook index characters, and segmentation is necessary because of the length of the message, the output is terminated if the passbook spaces beyond the bottom of the passbook; the remaining segments are not printed.

Output to a common buffer

The SEND CBUFF command is used to transmit data to a common buffer. The data is translated to the character set of the receiving 2980 model. If more than one 2980 model type is connected to the 2972 Control Unit, the lengths are truncated if they exceed the buffer size.

The DFH2980 structure

The DFH2980 structure contains constants that may be used when writing only COBOL or PL/I application programs for the 2980. The structure is obtained by copying DFH2980 into the application program.

For COBOL, DFH2980 is copied into the working storage section; for PL/I, DFH2980 is included using a %INCLUDE statement.

The station identification is given in the field STATIONID, whose value must be determined by the ASSIGN command. To test whether a normal or alternate station is being used, the STATIONID field is compared with values predefined in DFH2980. The values are:

```
STATION-#-A or STATION-#-N    (COBOL)

STATION_#_A or STATION_#_N    (PL/I)
```

where # is an integer (0 through 9) and A and N signify alternate and normal stations. (The break symbol is "–" (minus) for COBOL, and "_" (underline) for PL/I.)

The teller identification on a 2980 Teller Station Model 4 is given in the one-byte character field TELLERID. An ASSIGN command must be used to determine the TELLERID value.

Tab characters (X'05') must be included in the application program. The number of tabs required to position the print element to the first position of a passbook area is given in the field NUMTAB. An ASSIGN command must be used to determine the NUMTAB value. The value of NUMTAB is specified by the system programmer and may be unique to each terminal.

Other tab characters are inserted as needed to control formatting.

Any of the DFH2980 values TAB-ZERO through TAB-NINE for COBOL and PL/I, may be compared with NUMTAB to determine the number of tab characters that need to be inserted in an output message to obtain correct positioning of the print element. The tab character is included in DFH2980 as TABCHAR.

Thirty special characters are defined in DFH2980. Twenty-three of these can be referred to by the name SPECCHAR-# or SPECCHAR_# (for American National Standard COBOL or PL/I) where #is an integer (0 through 22). The seven other characters are defined with names that imply their usage, for example, TABCHAR. For further information on these thirty characters, see Appendix B, "Translation tables for the 2980" on page 347.

Several other characters defined in DFH2980, such as HOLDPCF or TCTTEPCR, are intended for use in application programs using CICS macros and should not be required in application programs using CICS commands.

3270 information display system (BTAM or TCAM)

```
RECEIVE
{INTO(data-area)|SET(ptr-ref)}
LENGTH(data-area)
[MAXLENGTH[(data-value)]]
[NOTRUNCATE]
[ASIS]
[BUFFER]                     (not TCAM)

Condition: LENGERR


SEND
FROM(data-area)
LENGTH(data-value)
[DEST(name)]                (TCAM only)
[WAIT]
[STRFIELD|[[ERASE]
  [CTLCHAR(data-value)]]]


CONVERSE
FROM(data-area)
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
TOLENGTH(data-area)
[MAXLENGTH[(data-value)]]
[NOTRUNCATE]
[STRFIELD|[[ERASE]
  [CTLCHAR(data-value)]]]

Condition: LENGERR


ISSUE PRINT                  (not TCAM)


ISSUE COPY                   (not TCAM)
TERMID(name)
[CTLCHAR(data-value)]
[WAIT]

Condition: TERMIDERR


ISSUE ERASEAUP
[WAIT]


ISSUE RESET


ISSUE DISCONNECT
```

3270 logical unit

```
RECEIVE
{INTO(data-area)|SET(ptr-ref)}
LENGTH(data-area)
[MAXLENGTH[(data-value)]]
[ASIS]
[BUFFER]
[NOTRUNCATE]

Conditions: LENGERR, TERMERR


SEND
FROM(data-area)
LENGTH(data-value)
[WAIT]
[INVITE|LAST]
[STRFIELD|[[ERASE]
  [CTLCHAR(data-value)]]]
[DEFRESP]

Condition: TERMERR


CONVERSE
FROM(data-area)
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
[STRFIELD|[[ERASE]
  [CTLCHAR(data-value)]]]
TOLENGTH(data-area)
[MAXLENGTH[(data-value)]]
[DEFRESP]
[NOTRUNCATE]

Conditions: LENGERR, TERMERR


ISSUE PRINT

Condition: TERMERR


ISSUE COPY
TERMID(name)
[CTLCHAR(data-value)]
[WAIT]

Conditions: LENGERR, TERMERR


ISSUE ERASEAUP
[WAIT]

Condition: TERMERR


ISSUE DISCONNECT
```

3270 SCS printer logical unit

```
SEND
FROM(data-area)
LENGTH(data-value)
[DEST(name)]
[WAIT]
[INVITE|LAST]
[CNOTCOMPL|DEFRESP]
[DEFRESP]
[STRFIELD]

Condition: TERMERR


CONVERSE
FROM(data-area)
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
TOLENGTH(data-area)
[MAXLENGTH[(data-value)]]
[DEST(name)]
[DEFRESP]
[STRFIELD]
[NOTRUNCATE]

Condition: TERMERR


ISSUE DISCONNECT
```

The SCS printer logical unit accepts a character string as defined by SNA (Systems Network Architecture). Some devices connected under SNA can send a signal which can be detected by the HANDLE CONDITION SIGNAL command, which in turn can invoke an appropriate handling routine. If necessary, a WAIT SIGNAL command can be used to make the application program wait for the signal. The PA keys on a 3287 can be used in this way, or with a RECEIVE command.

3270-display logical unit (LUTYPE2) and 3270-printer logical unit (LUTYPE3)

```
RECEIVE
[INTO(data-area)|SET(ptr-ref)]
LENGTH(data-area)
[MAXLENGTH[(data-value)]]
[ASIS]
[BUFFER]
[NOTRUNCATE]

Conditions: LENGERR, TERMERR


SEND
FROM(data-area)
LENGTH(data-value)
[WAIT]
[INVITE|LAST]
[DEST(name)]
[STRFIELD|[[ERASE]
  [CTLCHAR(data-value)]]]
[DEFRESP]

Condition: TERMERR


CONVERSE
FROM(data-area)
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
[STRFIELD|[[ERASE]
  [CTLCHAR(data-value)]]]
TOLENGTH(data-area)
[MAXLENGTH[(data-value)]]
[DEST(name)]
[DEFRESP]
[NOTRUNCATE]

Condition: LENGERR, TERMERR


ISSUE PRINT

Condition: TERMERR


ISSUE ERASEAUP
[WAIT]

Condition: TERMERR


ISSUE DISCONNECT
```

3600 finance communication system (BTAM)

```
RECEIVE
[INTO(data-area)|SET(ptr-ref)]
LENGTH(data-area)
[MAXLENGTH[(data-value)]]
[NOTRUNCATE]

Condition: LENGERR


SEND
FROM(data-area)
LENGTH(data-value)
[DEST(name)]
[WAIT]
[ASIS]


CONVERSE
FROM(data-area)
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
TOLENGTH(data-area)
[MAXLENGTH[(data-value)]]
[NOTRUNCATE]
[DEST(name)]

Condition: LENGERR


ISSUE RESET


ISSUE DISCONNECT
```

Input

The unit of transmission from a 3601 Finance Communication Controller to CICS is a segment consisting of the start-of-text data link control character (STX), the one byte identification of the 3600 logical work station that issued the processor write, the data, and either an end-of-block (ETB) or an end-of-text (ETX) control character.

A logical work station sends a message either in one segment, in which case the segment ends with ETX, or in more than one segment, in which case only the last segment ends with ETX, all others ending with ETB.

The input area passed to the user-written application program consists of the data only. The one-byte field TCTTEDLM, which may be obtained by means of an ASSIGN DELIMITER command, contains flags describing the data-link control character (ETB, ETX, or IRS) that ended the segment. The application program can issue terminal control commands to read the data until it receives a segment ending with ETX. If blocked data is transmitted, it is received by CICS as blocks of segments. Only the first segment in a block starts with the STX control character, and all segments are separated by IRS characters. None of the segments contain ETB or ETX characters except the last, which has the ETX character.

For blocked input, the flags in TCTTEDLM only indicate end of segment, not end of message. The CICS application program still receives only the data, but user-defined conventions may be required to determine the end of the message.

The field TCTTEDLM also indicates the mode of the input, either transparent or nontransparent. Blocked input is nontransparent.

The terminal control program does not pass input containing a 'start of header' (SOH) data link control character to a user-written application program. If it receives an SOH it sets an indicator in TCTTEDLM, passes the input to the user exit in the terminal control program, and then discards it.

Output

When an application program issues a SEND command, the terminal control program determines, from the value specified in the BUFFER parameter of the DFHTCT TYPE=TERMINAL system macro, the number of segments to be built for the message. It sends the message to the 3600 logical unit either in one segment consisting of a start-of-text character (STX), the data, and an end-of-text character (ETX); or in more than one segment, in which case only the last ends with ETX, all others ending with ETB.

The host input buffer of the 3600 controller and the input segment of the receiving logical unit must be large enough to accommodate the data sent by CICS. However, space for the data link control characters need not be included. The 3600 application program reads the data from the host, by means of an LREAD, until it has received the entire message.

CICS system output messages begin with 'DFH' followed by a four-byte message number and the message text. These messages are sent in nontransparent mode. CICS user-written application programs should not send messages starting with "DFH" to the 3601.

Resend message

When a logical unit sends a message to the host and a short-on-storage condition exists or the input is unsolicited (the active task associated with the terminal has not issued a read), the terminal control program sends a "resend" message to the logical unit. The format of this message is DFH1033 RE-ENTER followed by X'15' (a 3600 new line character) followed by the first eight bytes of the text of the message being rejected. No message is sent to the destinations CSMT or CSTL.

The first eight bytes of data sent to CICS can be used by the 3600 application program to define a convention to associate responses received from CICS with transactions sent to the host, for example, sequence numbers could be used.

If a CICS user-written application program has already issued a SEND command when a resend situation occurs, the resend message is not sent to the 3601 until the user-written application program message has been sent. A 3600 logical unit cannot receive a resend message while receiving a segmented message.

Only one resend message at a time can be queued for a logical unit. If a second resend situation occurs before CICS has written the first, a resend message, containing the eight bytes of data that accompanied the second input transaction from the 3600 logical unit, is sent.

The resend message is sent in transparent mode if the input data from the 3601 to be retransmitted is received by CICS in transparent mode. Otherwise it is sent in nontransparent mode.

3600 pipeline logical unit

```
SEND
FROM(data-area)
LENGTH(data-value)
[WAIT]

Condition: TERMERR


ISSUE DISCONNECT
```

3600 (3601) logical unit

```
RECEIVE
{INTO(data-area)|SET(ptr-ref)}
LENGTH(data-area)
[MAXLENGTH[(data-value)]]
[NOTRUNCATE]
```

Conditions: EOC, EODS, INBFMH, LENGERR, SIGNAL, TERMERR

```
SEND
FROM(data-area)
LENGTH(data-value)
[LDC(name)|FMH]
[DEST(name)]
[WAIT]
[INVITE|LAST]
[CNOTCOMPL|DEFRESP]
```

Conditions: SIGNAL, TERMERR

```
CONVERSE
FROM(data-area)
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
TOLENGTH(data-area)
[LDC(name)|FMH]
[DEST(name)]
[DEFRESP]
[MAXLENGTH[(data-value)]]
[NOTRUNCATE]
```

Conditions: EOC, EODS, INBFMH, LENGERR, SIGNAL, TERMERR

```
WAIT SIGNAL
```

Condition: SIGNAL

```
ISSUE DISCONNECT
```

Condition: SIGNAL

A logical device code (LDC) is a code that can be included in an outbound FMH to specify the disposition of the data (for example, to which subsystem terminal it should be sent). Each code can be represented by a unique LDC mnemonic.

The installation can specify up to 256 two-character mnemonics for each TCTTE, and two or more TCTTEs can share a list of these mnemonics. Corresponding to each LDC mnemonic for each TCTTE is a numeric value (0 through 255).

A 3600 device and a logical page size are also associated with an LDC. "LDC" or "LDC value" is used in this publication in reference to the code specified by the user. "LDC mnemonic" refers to the two-character symbol that represents the LDC numeric value.

When the LDC option is specified in the SEND command, the numeric value associated with the mnemonic for the particular TCTTE, is inserted in the FMH. The numeric value associated with the LDC mnemonic is chosen by the installation, and is interpreted by the 3601 application program.

3600 (3614) logical unit

```
RECEIVE
{INTO(data-area)|SET(ptr-ref)}
LENGTH(data-area)
[MAXLENGTH[(data-value)]]
[NOTRUNCATE]
```

Conditions: LENGERR, TERMERR

```
SEND
FROM(data-area)
LENGTH(data-value)
[DEST(name)]
[WAIT]
[INVITE|LAST]
[CNOTCOMPL|DEFRESP]
```

Condition: TERMERR

```
CONVERSE
FROM(data-area)
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
TOLENGTH(data-area)
[DEFRESP(name)]
[DEST(name)]
[MAXLENGTH[(data-value)]]
[NOTRUNCATE]
```

Conditions: LENGERR, TERMERR

```
ISSUE DISCONNECT
```

The data stream and communication format used between a CICS application program and a 3614 is determined by the 3614. The application program is therefore device dependent when handling 3614 communications.

For further information about designing 3614 application programs for CICS, see the *CICS/OS/VS IBM 4700/3600/3630 Guide*.

3630 plant communication system

Support and command syntax as for the 3600 (3601) Logical Unit and the 3600 Pipeline Logical Unit as described earlier in this chapter for the 3600 Finance Communication System.

3650/3680 host command processor logical unit

```
RECEIVE
{INTO(data-area)|SET(ptr-ref)}
LENGTH(data-area)
[MAXLENGTH[(data-value)]]
[NOTRUNCATE]

Conditions: EOC, LENGERR, TERMERR


SEND
FROM(data-area)
LENGTH(data-value)
[WAIT]
[INVITE|LAST]
[CNOTCOMPL|DEFRESP]
[FMH]

Conditions: TERMERR


CONVERSE
FROM(data-area)
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
TOLENGTH(data-area)
[FMH]
[DEFRESP]
[MAXLENGTH[(data-value)]]
[NOTRUNCATE]

Conditions: LENGERR, TERMERR


ISSUE DISCONNECT
```

3650 host conversational (3270) logical unit

```
RECEIVE
{INTO(data-area)|SET(ptr-ref)}
LENGTH(data-area)
[MAXLENGTH[(data-value)]]
[NOTRUNCATE]

Conditions: EOC, LENGERR, TERMERR


SEND
FROM(data-area)
LENGTH(data-value)
[CTLCHAR(data-value)]
[WAIT]
[ERASE]
[INVITE|LAST]
[CNOTCOMPL|DEFRESP]
[FMH]

Condition: TERMERR


CONVERSE
FROM(data-area)
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
TOLENGTH(data-area)
[CTLCHAR(data-value)]
[ERASE]
[DEFRESP]
[FMH]
[MAXLENGTH[(data-value)]]
[NOTRUNCATE]

Conditions: LENGERR, TERMERR


ISSUE PRINT

Condition: TERMERR


ISSUE ERASEAUP
[WAIT]

Condition: TERMERR


ISSUE DISCONNECT
```

3650 host conversational (3653) logical unit

```
RECEIVE
{INTO(data-area)|SET(ptr-ref)}
LENGTH(data-area)
[MAXLENGTH[(data-value)]]
[NOTRUNCATE]

Conditions: EOC, LENGERR, TERMERR


SEND
FROM(data-area)
LENGTH(data-value)
[WAIT]
[INVITE|LAST]
[CNOTCOMPL|DEFRESP]

Condition: TERMERR


CONVERSE
FROM(data-area)
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
TOLENGTH(data-area)
[DEFRESP]
[MAXLENGTH[(data-value)]]
[NOTRUNCATE]

Conditions: EOC, LENGERR, TERMERR


ISSUE DISCONNECT
```

3650 interpreter logical unit

```
RECEIVE
{INTO(data-area)|SET(ptr-ref)}
LENGTH(data-area)
[MAXLENGTH[(data-value)]]
[NOTRUNCATE]

Conditions: EOC, EODS, INBFMH,
LENGERR, TERMERR


SEND
FROM(data-area)
LENGTH(data-value)
[WAIT]
[INVITE|LAST]
[DEFRESP]
[FMH]

Condition: TERMERR


CONVERSE
FROM(data-area)
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
TOLENGTH(data-area)
[DEFRESP]
[FMH]
[MAXLENGTH[(data-value)]]
[NOTRUNCATE]

Conditions: EOC, EODS, INBFMH,
LENGERR, TERMERR


ISSUE LOAD
PROGRAM(name)
[CONVERSE]

Conditions: NONVAL, NOSTART, TERMERR


ISSUE EODS

Condition: TERMERR


ISSUE DISCONNECT
```

The ISSUE LOAD command specifies the name of the 3650 application program that is to be loaded.

The ISSUE EODS command can be used to send an end-of-data-set function management header to the 3650.

3650 pipeline logical unit

Support and command syntax as for the 3600 Pipeline Logical Unit.

3650/3680 full function logical unit

Support and command syntax as for the 3790 Full Function Logical Unit.

3660 supermarket scanning system

Support and command syntax as for System/3.

3735 programmable buffered terminal

```
RECEIVE
{INTO(data-area)|SET(ptr-ref)}
LENGTH(data-area)
[MAXLENGTH[(data-value)]]
[NOTRUNCATE]

Conditions: EOF (not TCAM), LENGERR


SEND
FROM(data-area)
LENGTH(data-value)
[DEST(name)]
[WAIT]
[ASIS]


CONVERSE
FROM(data-area)
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
TOLENGTH(data-area)
[DEST(name)]
[MAXLENGTH[(data-value)]]
[NOTRUNCATE]

Conditions: EOF (not TCAM), LENGERR


ISSUE RESET


ISSUE DISCONNECT
```

The 3735 Programmable Buffered Terminal may be serviced by CICS in response to terminal-initiated input (Autoanswer), or as a result of an automatic (Autocall) or time-initiated transaction.

3735 transactions — autoanswer

The 3735 transaction is attached by CICS upon receipt of input from a 3735. Data is passed to the application program in 476-byte blocks; each block (one buffer) may contain several logical records. The final block may be shorter than 476 bytes; zero-length final blocks are not, however, passed to the application program. If the block contains several logical records, the application program must perform any necessary deblocking and gathering of partial logical records.

Input data from a 3735 should be spooled to an intermediate data set (for example, an intrapartition destination) to ensure that all data has been captured before deblocking and processing that data.

The application program must follow 3735 conventions and read to end-of-file before attempting to write FDPs (form description programs) or data to the 3735. For this reason, the application program must include a HANDLE CONDITION command for the EOF condition. When control passes to the EOF routine, FDPs or data may be written to the 3735, or, optionally, CICS requested to disconnect the line.

The 3735 may transmit the EOF condition immediately upon connection of the line, in which case, a HANDLE CONDITION command for the EOF condition must be issued before any other terminal control commands.

The application program must format all special message headers for output to the 3735 (for example, SELECTRIC, POWERDOWN). If FDPs are to be transmitted to a 3735 with ASCII transmission code, the ASIS option must be included in the SEND command for each block of FDP records.

An ISSUE DISCONNECT command must be issued when all output has been transmitted to the 3735. If the application program ends during batch write mode before this command is executed, CICS forces a 3735 'receive abort' condition and all data just transmitted is ignored.

3735 transactions — autocall or time-initiated

In automatic or time-initiated transactions, all considerations stated above apply when CICS dials a 3735, except that EOF cannot occur.

CICS connects the line and allows the first terminal control command to indicate the direction of data transfer. If this first command is SEND and the 3735 has data to send, the 3735 causes the line to be disconnected.

3740 data entry system

```
RECEIVE
{INTO(data-area)|SET(ptr-ref)}
LENGTH(data-area)
[MAXLENGTH[(data-value)]]
[NOTRUNCATE]

Conditions: EOF (except TCAM),
ENDINPT (except TCAM),
LENGERR


SEND
FROM(data-area)
LENGTH(data-value)
[DEST(name)]
[WAIT]
[ASIS]


CONVERSE
FROM(data-area)
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
TOLENGTH(data-area)
[DEST(name)]
[MAXLENGTH[(data-value)]]
[NOTRUNCATE]

Condition: LENGERR


ISSUE ENDFILE [ENDOUTPUT]


ISSUE ENDOUTPUT [ENDFILE]


ISSUE RESET


ISSUE DISCONNECT
```

In batch mode, many files are exchanged between the 3740 and CICS in a single transmission. The transmission of an input batch must be complete before an output transmission can be started.

On input, the EOF (end-of-file) condition is raised by CICS when a null block (indicating the end of a physical file) is received from the 3740. A HANDLE CONDITION EOF command should be included to specify that processing of the file is to continue. Eventually, the ENDINPUT condition is raised by CICS when all input has been received. No more RECEIVE commands will be executed and a HANDLE CONDITION ENDINPUT command should be included to specify that control is to be returned to CICS so that the 3740 can be set to receive data.

On output, the ISSUE ENDFILE and ISSUE ENDOUTPUT commands are used to indicate the end-of-file and end-of-output conditions, respectively, to the 3740. These two conditions may be specified in one command if required, for example: ISSUE ENDFILE ENDOUTPUT.

3767 interactive logical unit

```
RECEIVE
{INTO(data-area)|SET(ptr-ref)}
LENGTH(data-area)
[MAXLENGTH[(data-value)]]
[NOTRUNCATE]

Conditions: EOC, LENGERR, SIGNAL,
TERMERR


SEND
FROM(data-area)
LENGTH(data-value)
[DEST(name)]
[WAIT]
[INVITE|LAST]
[CNOTCOMPL|DEFRESP]

Conditions: SIGNAL, TERMERR


CONVERSE
FROM(data-area)
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
TOLENGTH(data-area)
[DEST(name)]
[DEFRESP]
[MAXLENGTH[(data-value)]]
[NOTRUNCATE]

Conditions: EOC, LENGERR, SIGNAL,
TERMERR


WAIT SIGNAL

Condition: SIGNAL


ISSUE DISCONNECT

Condition: SIGNAL
```

3770 batch logical unit

```
RECEIVE
{INTO(data-area)|SET(ptr-ref)}
LENGTH(data-area)
[MAXLENGTH[(data-value)]]
[NOTRUNCATE]

Conditions: EOC, EODS, INBFMH,
LENGERR, SIGNAL, TERMERR


SEND
FROM(data-area)
LENGTH(data-value)
[DEST(name)]
[WAIT]
[INVITE|LAST]
[CNOTCOMPL|DEFRESP]
[FMH]

Conditions: SIGNAL, TERMERR


CONVERSE
FROM(data-area)
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
TOLENGTH(data-area)
[DEST(name)]
[DEFRESP]
[FMH]
[MAXLENGTH[(data-value)]]
[NOTRUNCATE]

Conditions: EOC, EODS, INBFMH,
LENGERR, SIGNAL, TERMERR


WAIT SIGNAL

Condition: SIGNAL


ISSUE DISCONNECT

Condition: SIGNAL
```

3770 interactive logical unit

Support and command syntax for the 3770 interactive logical unit is as for the 3767 Interactive Logical Unit.

3770 full function logical unit

Support and command syntax for the 3770 full function logical unit is as for the 3790 Full Function Logical Unit.

3780 communications terminal

Support and command syntax for the 3780 communication terminal is as for the System/3.

3790 full function logical unit

```
RECEIVE
{INTO(data-area)|SET(ptr-ref)}
LENGTH(data-area)
[MAXLENGTH[(data-value)]]
[NOTRUNCATE]

Conditions: EOC, EODS, INBFMH,
LENGERR, SIGNAL, TERMERR


SEND
FROM(data-area)
LENGTH(data-value)
[DEST(name)]
[WAIT]
[INVITE|LAST]
[CNOTCOMPL|DEFRESP]
[FMH]

Conditions: SIGNAL, TERMERR


CONVERSE
FROM(data-area)
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
TOLENGTH(data-area)
[DEST(name)]
[FMH]
[DEFRESP]
[MAXLENGTH[(data-value)]]
[NOTRUNCATE]

Conditions: EOC, EODS, INBFMH,
LENGERR, SIGNAL, TERMERR


WAIT SIGNAL

Condition: SIGNAL


ISSUE DISCONNECT

Condition: SIGNAL
```

3790 inquiry logical unit

```
RECEIVE
{INTO(data-area)|SET(ptr-ref)}
LENGTH(data-area)
[MAXLENGTH[(data-value)]]
[NOTRUNCATE]

Conditions: EOC, EODS, INBFMH,
LENGERR, TERMERR


SEND
FROM(data-area)
LENGTH(data-value)
[DEST(name)]
[WAIT]
[INVITE|LAST]
[CNOTCOMPL|DEFRESP]
[FMH]

Condition: TERMERR


CONVERSE
FROM(data-area)
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
TOLENGTH(data-area)
[DEST(name)]
[FMH]
[DEFRESP]
[MAXLENGTH[(data-value)]]
[NOTRUNCATE]

Conditions: EOC, EODS, INBFMH,
LENGERR, TERMERR


ISSUE DISCONNECT
```

3790 SCS printer logical unit

```
SEND
FROM(data-area)
LENGTH(data-value)
[DEST(name)]
[WAIT]
[INVITE|LAST]
[CNOTCOMPL|DEFRESP]
[DEFRESP]

Condition: TERMERR


ISSUE DISCONNECT
```

3790 (3270-display) logical unit

```
RECEIVE
{INTO(data-area)|SET(ptr-ref)}
LENGTH(data-area)
[ASIS]
[BUFFER]
[MAXLENGTH[(data-value)]]
[NOTRUNCATE]

Conditions: LENGERR, TERMERR


SEND
FROM(data-area)
LENGTH(data-value)
[DEST(name)]
[CTLCHAR(data-value)]
[WAIT]
[ERASE]
[INVITE|LAST]
[DEFRESP]

Condition: TERMERR


CONVERSE
FROM(data-area)
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
TOLENGTH(data-area)
[DEST(name)]
[DEFRESP]
[CTLCHAR(data-value)]
[ERASE]
[MAXLENGTH[(data-value)]]
[NOTRUNCATE]

Conditions: LENGERR, TERMERR


ISSUE PRINT

Condition: TERMERR


ISSUE ERASEAUP
[WAIT]

Condition: TERMERR


ISSUE DISCONNECT
```

3790 (3270-printer) logical unit

```
SEND
FROM(data-area)
LENGTH(data-value)
[CTLCHAR(data-value)]
[WAIT]
[ERASE]
[INVITE|LAST]
[DEFRESP]

Condition: TERMERR


ISSUE PRINT

Condition: TERMERR


ISSUE ERASEAUP
[WAIT]

Condition: TERMERR


ISSUE DISCONNECT
```

4700 finance communication system

Support and command syntax for the 4700 Finance Communication System is as for the 3600 Finance Communication System.

7770 audio response unit

```
RECEIVE
{INTO(data-area)|SET(ptr-ref)}
LENGTH(data-area)
[MAXLENGTH[(data-value)]]
[NOTRUNCATE]

Condition: LENGERR


SEND
FROM(data-area)
LENGTH(data-value)
[DEST(name)]
[WAIT]


CONVERSE
FROM(data-area)
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
TOLENGTH(data-area)
[MAXLENGTH[(data-value)]]
[NOTRUNCATE]

Condition: LENGERR


ISSUE RESET


ISSUE DISCONNECT
```

CICS cannot distinguish between special codes (characters) entered at audio terminals (for example, the 2721 Portable Audio Terminal); however, an application program can make use of these codes. The special codes that can be entered from a 2721 are as follows:

Key	Code(hex)
CALL END	37
CNCL	18
#	3B or 7B
VERIFY	2D
RPT	3D
EXEC	26
F1	B1
F2	B2
F3	B3
F4	B4
F5	B5
00	A0
000	3B or B0
IDENT	11, 12, 13, or 14 plus two other characters

For further information concerning the 2721, see the publication *IBM 2721 Portable Audio Terminal Component Description.*

The special codes A0 and 3B (or B0) are also generated by the keys * and # respectively of a "Touch-Tone" telephone. (Touch-Tone is the trademark of the American Telephone and Telegraph Company.)

If the SET option has been specified in the associated command, codes 26, 37, and 3B (each of which causes a hardware interrupt) will immediately follow the data, but will not be included in the value set by the LENGTH option.

If the end-of-inquiry (EOI) Disable Feature (Feature No. 3540) is installed on the 7770 Model 3, the option of including either or both # and 000 as data is available.

If, after receiving at least one code from a terminal, no other codes have been received by the 7770 for a period of five seconds, the 7770 generates an EOI hardware interrupt that ends the operation.

Terminal control options

ASIS

for System/370, System/3, System/7, 2770, 2780, and 3740: indicates that output is to be sent in transparent mode (with no recognition of control characters and accepting any of the 256 possible combinations of eight bits as valid transmittable data).

For System/7: indicates that the data being written or read is not to be translated.

For 3735: prevents translation of the Form Description Program (FDP) records that are to be transmitted to a 3735 using ASCII code.

For 3270 and VTAM terminals: specifies a temporary override of the uppercase translation feature of CICS to allow 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, as terminal control will perform a read initial and use the terminal defaults to translate the data.

This option has no effect if the screen contains data prior to a transaction being initiated. This data will be read and translated in preparation for the next task and the first RECEIVE command in that task will retrieve the translated data.

ATTACHID(name)

specifies, for a BUILD ATTACH command, that the set of values specified is to be placed in an attach header control block identified by the specified name (maximum of eight characters).

specifies, for a SEND or CONVERSE command, that an attach header (created by a BUILD ATTACH command) is to precede, and be concatenated with, the user data supplied in the FROM option. 'Name' (maximum of eight characters) identifies the attach header control block to be used in the local task.

specifies, for an EXTRACT ATTACH command, that values are to be retrieved from an attach header control block. 'Name' (maximum of eight characters) identifies this control block to the local task. If the option is omitted, the attach header control block to be used is that associated with the facility named in the SESSION option.

BUFFER

specifies that the contents of the 3270 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 same order that they appear in the 3270 buffer.

CBUFF

specifies that data is to be written to a common buffer in a 2972 Control Unit. The WAIT option is implied.

CONFIRM

indicates that an application using a synchronization 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 will have EIBERR and EIBERRCD set. CICS will not return control to the sending application until the response is received.

CONVERSE

specifies that the 3650 application program will communicate with the host processor. If this option is not specified, the 3650 application program cannot communicate with the host processor.

CONVID(data-area)

specifies the symbolic identifier (maximum of four characters) of an LUTYPE6.2 conversation. This option specifies the alternate facility to be used. If this option is omitted, the principal facility for the task will be used.

CTLCHAR(data-value)

specifies a one-byte write control character (WCC) that controls a SEND command, or the copy control character (CCC) that controls an ISSUE COPY command, for a 3270. A COBOL user must specify a data area containing this character. If the option is omitted from a SEND command, all modified data tags are reset to zero and the keyboard is restored. If the option is omitted from an ISSUE COPY command, the contents of the entire buffer (including nulls) are copied.

DATASTR{(name)|(data-area)}

this corresponds to the data stream profile field, ATTDSP, in an LUTYPE6.1 attach FMH.

For communication between two CICS systems, no particular significance is attached by CICS to the data stream profile field in an attach FMH. For most CICS applications, the option may be omitted when a value of "user defined" will be assumed.

For communication between a CICS system and another subsystem, see the manuals for the subsystem for information on how to use the data stream profile field in an attach FMH.

When EIBATT is set during execution of a RECEIVE or CONVERSE command, the EXTRACT ATTACH command may be used to examine the data stream profile field received in the attach FMH.

The value is halfword binary; only the low-order byte is used. If this option is omitted, "user defined" is assumed. The bits in the binary value are used as follows:

0-7	reserved - must be set to zero
8-11	0000 - user defined 1111 - SCS data stream 1110 - 3270 data stream 1101 - structured field 1100 - logical record management
12-15	defined by the user if bits 8-11 are set to 0000; otherwise reserved (must be set to zero).

A value of "structured field" indicates that chains begin with four bytes of data that are used to interpret the following data; the four bytes consist of overall length (2 bytes), class identifier (1 byte), and subclass identifier (1 byte). A value of "logical record management" indicates that chains can be split into separate fields by the data receiver.

These values may be used for communication between a CICS system and another subsystem; for further details of structured fields and logical record management, see the manuals for the subsystem.

If the option is omitted from the BUILD ATTACH command, a value of "user defined" is assumed.

DEFRESP
indicates that a definite response is required when the output operation has been completed.

DEST(name)
specifies the four-byte symbolic name of the TCAM destination to which the message is to be sent. This option is meaningful only for terminals for which DEVICE=TCAM has been specified in the DFHTCT TYPE=SDSCI system macro.

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 will clear the screen ready for the new output data.

However, when switching from one screen size to another on a transaction basis, bear in mind that if ERASE is not specified in the first output command of the transaction, the screen size will be unchanged from its previous setting, that is, the previous transaction setting, or the default screen size if the CLEAR key has been pressed.

FMH
specifies that a function management header has been included in the data that is to be written. If the ATTACHID option is specified as well, the concatenated FMH flag will be set in the attach FMH.

FROM(data-area)
specifies the data that is to be written to the terminal or logical unit. For the ISSUE PASS command it contains the logon user data that is to be passed to the application named in the LUNAME option. This option may be omitted if ATTACHID is specified on an LUTYPE6.1 command, or if INVITE, CONFIRM, or LAST is specified on an LUTYPE6.2 SEND command.

FROMLENGTH(data-value)
see LENGTH(parameter). The FROMLENGTH option of the CONVERSE command is equivalent to the LENGTH option of a SEND command.

INTO(data-area)
specifies the receiving field for the data read from the terminal or 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.

IUTYPE{(name)|(data-area)}
this corresponds to the interchange unit field, ATTIU, in an LUTYPE6.1 attach FMH.

For communication between two CICS systems, no particular significance is attached by CICS to the interchange unit field in an attach FMH. For most CICS applications the option may be omitted, when a value of "multiple chain" will be assumed.

For communication between a CICS system and another subsystem, see the manuals for the subsystem for information on how to use the interchange unit field in an attach FMH.

When EIBATT is set during execution of a RECEIVE or CONVERSE command, the EXTRACT ATTACH command may be used to examine the interchange unit field received in the attach FMH.

The value is halfword binary; only the low-order 7 bits being used. The bits in the binary value are used as follows:

```
0-10   reserved - must be set to
                  zero
11     0 - not end of multichain
           interchange unit
       1 - end of multichain
           interchange unit
12,13  reserved - must be set to
                  zero
14,15  00 - multichain interchange
            unit
       01 - single chain
            interchange unit
       10 - reserved
       11 - reserved
```

If the option is omitted from the BUILD ATTACH command, values of "not end of multichain interchange unit" and "multiple chain" are assumed.

LAST
specifies that this is the last output operation for a transaction and therefore the end of a bracket.

LDC(name)
specifies the two-character mnemonic used to determine the appropriate logical device code (LDC) numeric value. The mnemonic represents an LDC entry in the DFHTCT TYPE=LDC macro.

LEAVEKB
specifies that the keyboard is to remain locked at the completion of the data transfer.

LENGTH(parameter)
specifies the length (as a halfword binary value) of the data transmitted by RECEIVE and SEND commands.

For a RECEIVE command with the INTO option, but without the MAXLENGTH option, the parameter must be a data area that specifies the maximum length that the program will accept. If the value specified is less than zero, zero is assumed. If the length of the data exceeds the value specified, but the NOTRUNCATE option is not specified, 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 the data.

For a RECEIVE command with the SET option, the parameter must be a data area. When the data has been received, the data area is set to the length of the data.

For a SEND command, the parameter must be a data value that is the length of the data that is to be written.

For an ISSUE PASS command, the parameter is a data value that is the length of the data specified in the FROM option.

LINEADDR(data-value)
specifies that the writing is to begin on a specific line of a 2260/2265 screen. The data value is a halfword binary value in the range 1 through 12 for a 2260, or 1 through 15 for a 2265.

LUNAME(data-area)
specifies the name of the VTAM application to which the terminal is to be passed.

MAXLENGTH(data-value)
specifies, as a halfword binary value, the maximum amount of data that CICS is to recover in response to a RECEIVE or CONVERSE command. If INTO is specified, MAXLENGTH will override the use of LENGTH and TOLENGTH as an input to CICS. If SET is specified, MAXLENGTH provides a means whereby the program can limit the amount of data 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 or TOLENGTH option is set to the original length of data.

If the length of data exceeds the value specified and the NOTRUNCATE option is present, CICS will retain the remaining data and use it to satisfy subsequent RECEIVE commands. The data area specified in the LENGTH or TOLENGTH option is set to the length of data returned.

If no operand is coded for MAXLENGTH, CICS will default a value in the same way that it currently defaults a value for the LENGTH option if this is omitted.

NETNAME(name)
specifies the eight-character name of the logical unit in the VTAM network.

NOQUEUE
specifies that the request to allocate a session or a system is not to be queued when a suitable session or system cannot be acquired immediately. The SESSBUSY or SYSBUSY condition will be raised and it will be handled as described on page 43.

NOSUSPEND
is an alternative keyword for NOQUEUE. It means the same.

NOTRUNCATE
specifies that when the data available exceeds the length requested in a RECEIVE or CONVERSE command, the remaining data is not to be discarded but is to be retained for retrieval by subsequent RECEIVE commands.

PASSBK
specifies that communication is with a passbook at a 2980. The WAIT option is implied.

PIPLENGTH(parameter)
for a CONNECT PROCESS command, parameter is a data value that specifies the total length of the list specified by PIPLIST. Its format is halfword binary.

For an EXTRACT PROCESS command, parameter is a data area into which is returned the total length of the PIP (process initialization parameter) list. Its format is halfword binary.

PIPLIST(parameter)
for a CONNECT PROCESS command, parameter is a data area containing the PIP data that is to be sent to the remote system. The PIP list consists of variable length records, each containing a single PIP.

For an EXTRACT PROCESS command, parameter is a pointer reference that is set to the address of a CICS-provided data area containing a PIP list. This list contains variable length records in the same format as the list in the CONNECT PROCESS command. A returned value of zero means that no PIP data has been received by CICS.

PROCESS{(name)|(data area)}
this corresponds to the process name, ATTDPN, in an LU6.1 attach FMH.

For communication between two CICS systems, a transaction running in one system can acquire a session to the second system and can identify the transaction to be attached; in the second system the identification is carried in the first chain of data sent across the session.

In general, the first four bytes of data will identify the transaction to be attached. However an attach FMH, identifying the transaction to be attached, may be built and sent; the PROCESS option on the BUILD ATTACH command is used to specify the transaction name. (The receiving CICS system will use just the first 4 bytes of the process name as a transaction name).

No significance is attached by CICS to process names in attach FMHs sent in chains of data other than the first.

For communication between a CICS system and another subsystem, see the manuals for the subsystem for information on how to use the process name field in an attach FMH.

When EIBATT is set during execution of a RECEIVE or CONVERSE command, the EXTRACT ATTACH command may be used to examine the process name received in the attach FMH.

PROCLENGTH(data-area)
specifies, on a CONNECT PROCESS command, the length (as a halfword binary value) of the process name specified by the PROCNAME option. On an EXTRACT PROCESS command, it specifies a halfword data area that is set by CICS to the length of the process name.

PROCNAME{(name)|(data area)}
on a CONNECT PROCESS command, it specifies the process (in CICS terms, the transaction) that is to be connected in the remote system.

On an EXTRACT PROCESS command, it specifies the data area into which the process name specified by the remote system which caused the task, is to be started. The data area must be 32 bytes long. The process name will be padded on the right with blanks if it is shorter that 32 bytes.

PROFILE(name)
specifies the name (maximum of eight characters) of a set of session processing options, held in the PCT, that are to be used during execution of terminal control commands for the session specified in the SYSID or SESSION options. If this option is omitted, a set of processing options, called DFHCICSA, will be selected.

PROGRAM(name)
specifies the name (maximum of eight characters) of the 3600 application program that is to be loaded.

PSEUDOBIN
specifies that the data being written or read is to be translated from System/7 pseudobinary representation to hexadecimal on a RECEIVE command or from hexadecimal to pseudobinary on a SEND command.

QUEUE{(name)|(data-area)}
this corresponds to the queue name, ATTDQN, in an attach FMH.

For communication between two CICS systems, no significance is attached by CICS to the queue name in an attach FMH.

For communication between a CICS system and another subsystem, see the manuals for the subsystem for information on how to use the queue name field in an attach FMH.

When EIBATT is set during execution of a RECEIVE or CONVERSE command, the EXTRACT ATTACH command may be used to examine the queue name received in the attach FMH.

RECFM{(name)|(data area)}
this corresponds to the deblocking algorithm field, ATTDBA, in an LU6.1 attach FMH.

For communication between two CICS systems, no particular significance is attached by CICS to the deblocking algorithm field in an attach FMH. For most CICS applications, the option may be omitted when a value of "chain of RUs" will be assumed.

For communication between a CICS system and another subsystem, see the manuals for the subsystem for information on how to use the deblocking algorithm field in an attach FMH.

When EIBATT is set during execution of a RECEIVE or CONVERSE command, the EXTRACT ATTACH command may be used to examine the deblocking algorithm field received in the attach FMH.

The value is halfword binary; only the low-order 8 bits being used. The bits in the binary value are used as follows:

```
0-7   reserved - must be set to
                 zero
8-15  X'00' - reserved
      X'01' - variable length
              variable blocked
      X'02' - reserved
      X'03' - reserved
      X'04' - chain of RUs
      X'05'
       to X'FF' - reserved
```

If the option is omitted from the BUILD ATTACH command, a value of "chain of RUs" is assumed.

RESOURCE{(name)|(data-area)}

this corresponds to the resource name, ATTPRN, in an LU6.1 attach FMH.

For communication between two CICS systems, no significance is attached by CICS to the resource name in an attach FMH.

For communication between a CICS system and another subsystem, see the manuals for the subsystem for information on how to use the resource name field in an attach FMH.

When EIBATT is set during execution of a RECEIVE or CONVERSE command, the EXTRACT ATTACH command may be used to examine the resource name received in the attach FMH.

RPROCESS{(name)|(data-area)}

this corresponds to the return process name, ATTRDPN, in an LU6.1 attach FMH.

For communication between two CICS systems, no significance is attached by CICS to the return process name in an attach FMH.

For communication between a CICS system and another subsystem, see the manuals for the subsystem for information on how to use the return process name field in an attach FMH.

When EIBATT is set during execution of a RECEIVE or CONVERSE command, the EXTRACT ATTACH command may be used to examine the return process name received in the attach FMH.

RRESOURCE{(name)|(data-area)}

this corresponds to the return resource name, ATTRPRN, in an LU6.1 attach FMH.

For communication between two CICS systems, no significance is attached by CICS to the return resource name in an attach FMH.

For communication between a CICS system and another subsystem, see the manuals for the subsystem for information on how to use the return resource name field in an attach FMH.

When EIBATT is set during execution of a RECEIVE or CONVERSE command, the EXTRACT ATTACH command may be used to examine the return resource name received in the attach FMH.

SESSION(name)

specifies the symbolic identifier (maximum of four characters) of a session TCTTE. This option specifies the alternate session to be used. If this option is omitted, the principal facility for the task will be used.

SET(ptr-ref)

specifies the pointer reference that is to be set to the address of the data read from the terminal or logical unit.

STRFIELD

specifies that the data area specified in the FROM option contains structured fields. If this option is specified, the contents of all structured fields must be handled by the application program. The CONVERSE command, rather than a SEND command, must be used if the data area contains a read partition structured field. (Structured fields are described in the *CICS/OS/VS IBM 3270 Data Stream Device Guide*.) CTLCHAR and ERASE are mutually exclusive with STRFIELD, and their use with STRFIELD will generate an error message.

SYNCLEVEL{(data-area|data-value)}

specifies, on a CONNECT PROCESS command, as a halfword binary value, the synchronization level for the current conversation. The possible values are: 0 none, 1 commit only, 2 all. On an EXTRACT PROCESS command, specifies a halfword data area that is set by CICS to the SYNCLEVEL value. For further information about synchronization levels see the *CICS/MVS Intercommunication Guide*.

SYSID{(name)|(data-area)}

specifies the name (maximum of four characters) of a system TCTSE. This option specifies that one of the sessions to the named system is to be allocated.

When used with the EXTRACT TCT command, this option specifies the variable to be set to the equivalent local name of the system.

TERMID{(name)|(data-area)}

specifies the name (up to four characters in length) of the terminal whose buffer is to be copied. The terminal must have been defined in the TCT.

When used with the EXTRACT TCT command this option specifies the variable to be set to the equivalent local name of the terminal.

TOLENGTH(data-area)

see LENGTH(parameter). The TOLENGTH option of the CONVERSE command is equivalent to the LENGTH option of a RECEIVE command.

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 once processing of the command has started. A subsequent input or output request (terminal control, BMS, or batch data interchange) to the terminal associated with the task will cause the application program to wait until the previous request has been completed.

Terminal control exceptional conditions

Some of the following exceptional conditions may occur in combination with others. CICS checks for these conditions in the following order: 1 EODS, 2 INBFMH, 3 EOC. If more than one of these conditions occurs, only the first one found to be present is passed to the application program.

However, EIBRCODE will be set to indicate all the conditions that have occurred.

CBIDERR
occurs if the named set of terminal-control processing options cannot be found.

Default action: terminate the task abnormally.

ENDINPT
occurs when an end-of-input indicator is received.

Default action: terminate the task abnormally.

EOC
occurs when a request/response unit (RU) is received with the end-of-chain indicator set. Field EIBEOC also contains this indicator.

Default action: ignore the condition.

EODS
occurs when an end-of-data-set indicator is received.

EOF
occurs when an end-of-file indicator is received.

Default action: terminate the task abnormally.

IGREQCD
occurs when an attempt is made to execute a SEND or CONVERSE command after a SIGNAL data-flow control command with an RCD (request change direction) code has been received from an LUTYPE4 logical unit.

Default action: terminate the task abnormally.

INBFMH
occurs if a request/response unit (RU) contains a function management header (FMH). Field EIBFMH contains this indicator and it should be used in preference to INBFMH. The IGNORE CONDITION command can be used to ignore the condition.

Default action: terminate the task abnormally.

INVREQ
occurs, for various commands, as follows:

- ALLOCATE – the LU specified is already allocated.
- FREE – the LU specified is in the wrong state.
- CONNECT PROCESS – SYNCLVL 2 has been requested, but cannot be supported on the session in use.
- EXTRACT ATTACH – invalid data.
- SEND – the CONFIRM option has been specified but LU6.2 conversation is not SYNCLVL 1.
- EXTRACT TCT – invalid NETNAME.
- EXTRACT PROCESS – invalid CONVID.

INVREQ also occurs if:

- An invalid command has been issued for the terminal or LU in use.
- An invalid command has been issued for the LU6.2 conversation type in use.

Default action: terminate the task abnormally.

LENGERR
occurs, for a RECEIVE or CONVERSE command, if data is discarded by CICS because its length exceeds the maximum the program will accept and the NOTRUNCATE option is not specified.

Occurs also if an out of range value is supplied in the LENGTH option on the SEND command, the FROMLENGTH option on the CONVERSE command, or the PROCLENGTH option on the CONNECT PROCESS command.

This condition will also occur if:

- The value specified in the PIPLENGTH option is less than zero.
- The value specified in the PIPLENGTH option exceeds the CICS implementation limit of 32,767.
- A PIP length element has a value less than 4.
- The sum of the length elements in the PIPLIST does not equal the value specified by PIPLENGTH.

Default action: terminate the task abnormally.

NONVAL
occurs if a 3650 application program name is invalid.

Default action: terminate the task abnormally.

NOPASSBKRD
occurs if no passbook is present on an input operation.

NOPASSBKWR
occurs if no passbook is present on an output operation.

NOSTART
the requested 3650 application program.

Default action: terminate the task abnormally.

NOTALLOC
occurs if the facility specified in the command is not owned by the application.

Default action: terminate the task abnormally.

RDATT
occurs if a RECEIVE command is terminated by the attention (ATTN) key rather than the return key.

Default action: ignore the condition.

SESSBUSY
occurs if the request for a session cannot be serviced immediately.

Default action: queue the request until a session is available.

SESSIONERR
occurs if the name specified in the SESSION option of the ALLOCATE command is not that of a session TCTTE, or if the session cannot be allocated because it is out of service.

Default action: terminate the task abnormally.

SIGNAL
control command is received from a logical unit or session. It is raised by execution of the next SEND, RECEIVE, or WAIT TERMINAL command that refers to the logical unit or session. It is raised also by execution of a WAIT SIGNAL command, in which case the data-flow control command has been received from the principal facility. EIBSIG will always be set when an inbound signal is received.

Default action: ignore the condition.

SYSBUSY
occurs if the request for a session cannot be serviced immediately.

Default action: queue the request until a session is available.

SYSIDERR
occurs if CICS is unable to provide the application program with a suitable session. This will occur if:

1. The name specified in the SYSID option is not recognized by CICS, or
2. The mode name derived from the PROFILE option is not one of the mode names defined for the LU6.2 system entry, or
3. All of the sessions in the group specified by SYSID and mode name are out of service, or if all sessions are out of service.

Default action: terminate the task abnormally.

TERMERR
occurs for a terminal related error, such as a session failure. This condition applies to VTAM-connected terminals only. Because of the asynchronous nature of this condition, the application program should check, using CONFIRM or SYNCPOINT, to make sure any errors still outstanding have been resolved before it relinquishes control.

If you want to handle this condition, you must first issue a FREE command to free the session. Failure to do this will result in an INVREQ condition, and an abend code ATCV if the condition is not handled.

Default action: terminate the task abnormally with abend code ATNI.

TERMIDERR
occurs if the specified terminal identifier cannot be found in the terminal control table (TCT).

Default action: terminate the task abnormally.

WRBRK
occurs if a SEND command is terminated by the attention key.

Default action: ignore the condition.

Chapter 3.4. Batch data interchange

The CICS batch data interchange program provides for communication between an application program and a named data set (or destination) that is part of a batch data interchange logical unit in an outboard controller, or with a selected medium on a batch logical unit or an LUTYPE4 logical unit. This medium indicates the required device such as a printer or console.

The term "outboard controller" is a generalized reference to a programmable subsystem, such as the IBM 3770 Data Communication System, the IBM 3790 Data Communication System, or the IBM 8100 System running DPCX, which uses SNA protocols. (Details of SNA protocols and the data sets that can be used are given in the *CICS/OS/VS IBM 3767/3770/6670 Guide* and *CICS/OS/VS IBM 3790/3730/8100 Guide*.)

Batch data interchange commands are provided to:

- Initiate transfer of a data set to the CICS application program (ISSUE QUERY).
- Read a record from a data set or read data from an input medium (ISSUE RECEIVE).
- Transmit data to a named data set or to a selected medium (ISSUE SEND).
- Add a record to a data set (ISSUE ADD).
- Update (replace) a record in a data set (ISSUE REPLACE).
- Delete a record from a data set (ISSUE ERASE).
- Terminate processing of a data set (ISSUE END).
- Terminate processing of a data set abnormally (ISSUE ABORT).
- Request the next record number in a data set (ISSUE NOTE).
- Wait for an operation to be completed (ISSUE WAIT).

Where the controller is an LUTYPE4 logical unit, only the ISSUE ABORT, ISSUE END, ISSUE RECEIVE, ISSUE SEND, and ISSUE WAIT commands can be used.

Where the data set is a DPCX/DXAM data set, only the ISSUE ADD, ISSUE ERASE, and ISSUE REPLACE commands can be used.

The HANDLE CONDITION command is used to deal with any exceptional conditions that occur during execution of a batch data interchange command. See "Chapter 1.5. Exceptional conditions" on page 43 for further information about exceptional conditions.

Destination selection and identification

All batch data interchange commands except ISSUE RECEIVE include options that specify the destination. This is either a named data set in a batch data interchange logical unit, or a selected medium in a batch logical unit or LUTYPE4 logical unit.

Selection by named data set: The DESTID and DESTIDLENG options must always be specified, to supply the data set name and its length (up to a maximum of eight characters). For destinations having diskettes, the VOLUME and VOLUMELENG options may be specified, to supply a volume name and its length (up to a maximum of six characters); the volume name identifies the diskette that contains the data set to be used in the operation. If the VOLUME option is not specified for a multidiskette destination, all diskettes are searched until the required data set is found.

Selection by medium: As an alternative to naming a data set as the destination, various media can be specified by means of the CONSOLE, PRINT, CARD, or WPMEDIA1-4 options. These media can be specified only in an ISSUE ABORT, ISSUE END, ISSUE SEND, or ISSUE WAIT command.

Definite-response

CICS uses terminal control commands to carry out the functions specified in batch data interchange commands. For those commands that cause terminal control output requests to be made, the DEFRESP option can be specified. This option has the same effect as the DEFRESP option of the SEND terminal control command; that is, to request a definite response from the outboard controller, irrespective of the specification of message integrity for the CICS task (by the system programmer). The DEFRESP option can be specified for the ISSUE ADD, ISSUE ERASE, ISSUE REPLACE, and ISSUE SEND commands.

Waiting for function completion

For those batch data interchange commands that cause terminal control output requests to be made, the NOWAIT option can be specified. This option has the effect of allowing CICS task processing to continue; unless the option is specified, task activity is suspended until the batch data interchange command is completed. The NOWAIT option can be specified only on the ISSUE ADD, ISSUE ERASE, ISSUE REPLACE, and ISSUE SEND commands.

After a batch data interchange command with the NOWAIT option has been issued, task activity can be suspended, by the ISSUE WAIT command, at a suitable point in the program to wait for the command to be completed.

Interrogate a data set (ISSUE QUERY)

```
ISSUE QUERY
DESTID(data-value)
[DESTIDLENG(data-value)]
[VOLUME(data-value)
  [VOLUMELENG(data-value)]]

Conditions: FUNCERR, SELNERR,
UNEXPIN
```

This command is used to request that a sequential data set in an outboard controller be transmitted to the host system. The application program should either follow this command with ISSUE RECEIVE commands to obtain the resulting inbound data, or terminate the transaction to allow CICS to start a new transaction to process the data.

Read a record from a data set (ISSUE RECEIVE)

```
ISSUE RECEIVE
{INTO(data-area)|SET(ptr-ref)}
[LENGTH(data-area)]

Conditions: DSSTAT, EODS, LENGERR,
UNEXPIN
```

This command is used to read a record from an outboard controller. The INTO option specifies the area into which the data is to be placed. The LENGTH option must include a data area that contains the maximum length of record that the program will accept. If the record length exceeds the specified maximum length, the record is truncated and the LENGERR condition occurs. After the retrieval operation, the data area specified in the LENGTH operand is set to the record length (before any truncation occurred).

Alternatively, a pointer reference can be specified in the SET option. CICS then acquires an area of sufficient size to hold the record and sets the pointer reference to the address of that area. After the retrieval operation, the data area specified in the LENGTH option is set to the record length.

The outboard controller might not send the data from the data set specified in the ISSUE QUERY command. The ASSIGN command must be used to obtain the value of DESTID, which identifies the data set that has actually been transmitted; also the value of DESTIDLENG, which is the length of the identifier in DESTID.

Add a record to a data set (ISSUE ADD)

```
ISSUE ADD
DESTID(data-value)
[DESTIDLENG(data-value)]
[VOLUME(data-value)
  [VOLUMELENG(data-value)]]
FROM(data-area)
[LENGTH(data-value)]
[NUMREC(data-value)]
[DEFRESP]
[NOWAIT]
[RIDFLD(data-area) RRN]

Conditions: FUNCERR, SELNERR,
UNEXPIN
```

This command is used to add records to a sequential or keyed direct data set in an outboard controller. The FROM option is used to specify the data to be written, and the LENGTH option specifies its length.

The RIDFLD option is only needed with this command when it applies to a DPCX/DXAM data set. In this case, it specifies the relative record number of the record to be added. When RIDFLD is used, NUMREC must be 1 (the default).

Update a record in a data set (ISSUE REPLACE)

```
ISSUE REPLACE
DESTID(data-value)
[DESTIDLENG(data-value)]
[VOLUME(data-value)
  [VOLUMELENG(data-value)]]
FROM(data-area)
[LENGTH(data-value)]
[NUMREC(data-value)]
RIDFLD(data-area)
[[KEYLENGTH(data-value)]
  [KEYNUMBER(data-value)]|RRN]
[DEFRESP]
[NOWAIT]

Conditions: FUNCERR, SELNERR,
UNEXPIN
```

This command is used to update (replace) a record in either a relative (addressed direct) or an indexed (keyed direct) data set in an outboard controller.

The FROM option is used to specify the data to be written to the data set and the LENGTH option specifies the length of the data.

The RIDFLD option specifies the relative record number of the first record to be replaced for a relative data set, or the embedded key in the data specified by the FROM option for an indexed data set.

For a relative data set, the RRN option must be specified, because the RIDFLD option contains a relative record number. In addition, the NUMREC option must specify the number of records to be replaced consecutively, starting with the one specified in RIDFLD.

For an indexed data set, the RIDFLD option specifies the key embedded in the data specified in the FROM option. In addition, the KEYLENGTH option must specify the length of the key. The NUMREC option cannot be specified because only one record is replaced.

For a DPCX/DXAM data set, KEYNUMBER specifies the number (1 through 8) of the index to be used to access the record to be updated.

Delete a record from a data set (ISSUE ERASE)

```
ISSUE ERASE
DESTID(data-value)
[DESTIDLENG(data-value)]
[VOLUME(data-value)
  [VOLUMELENG(data-value)]]
RIDFLD(data-area)
[[KEYLENGTH(data-value)]
  [KEYNUMBER(data-value)]|RRN]
[NUMREC(data-value)]
[DEFRESP]
[NOWAIT]

Conditions: FUNCERR, SELNERR,
UNEXPIN
```

This command is used to delete a record from a keyed direct data set in an outboard controller, or erase a record from a DPCX/DXAM relative record data set. The RIDFLD option specifies the key of the record to be deleted; the length of the key must be specified in the KEYLENGTH option.

For a DPCX/DXAM data set, NUMREC must be set to 1 (the default). KEYNUMBER specifies the index (1 through 8) used to find the record to be erased. Also, RIDFLD and RRN can be used to erase a record from a relative record data set. In this case, KEYNUMBER cannot be specified.

End processing of a data set (ISSUE END)

```
ISSUE END
[DESTID(data-value)
  [DESTIDLENG(data-value)]|
    [SUBADDR(data-value)]
      [CONSOLE|PRINT|CARD|
        WPMEDIA1|WPMEDIA2|
        WPMEDIA3|WPMEDIA4]]
[VOLUME(data-value)
  [VOLUMELENG(data-value)]]

Conditions: FUNCERR, SELNERR,
UNEXPIN
```

This command is used to end communication with a data set in an outboard controller or with the selected medium. The data set specified in the DESTID option, or the selected medium, is de-selected normally. The options CONSOLE, PRINT, CARD, WPMEDIA1-4 are alternatives to DESTID and DESTIDLENG.

End processing of a data set abnormally (ISSUE ABORT)

```
ISSUE ABORT
[DESTID(data-value)
  [DESTIDLENG(data-value)]|
    [SUBADDR(data-value)]
      [CONSOLE|PRINT|CARD|
        WPMEDIA1|WPMEDIA2|
        WPMEDIA3|WPMEDIA4]]
[VOLUME(data-value)
  [VOLUMELENG(data-value)]]

Conditions: FUNCERR, SELNERR,
UNEXPIN
```

This command is used to end communication with a data set in an outboard controller, or with the selected medium, abnormally. The data set specified in the DESTID option is deselected abnormally. The options CONSOLE, PRINT, CARD, WPMEDIA1-4 are alternatives to DESTID and DESTIDLENG.

Send data to an output device (ISSUE SEND)

```
ISSUE SEND
[DESTID(data-value)
  [DESTIDLENG(data-value)]|
    [SUBADDR(data-value)]
      [CONSOLE|PRINT|CARD|
        WPMEDIA1|WPMEDIA2|
        WPMEDIA3|WPMEDIA4]]
[VOLUME(data-value)
  [VOLUMELENG(data-value)]]
[LENGTH(data-value)]
FROM(data-area)
[NOWAIT]
[DEFRESP]

Conditions: FUNCERR, IGREQCD,
SELNERR, UNEXPIN
```

This command is used to send data to a named data set in an outboard controller, or to a selected medium in a batch logical unit or an LUTYPE4 logical unit. The options CONSOLE, PRINT, CARD, WPMEDIA1-4 are alternatives to DESTID and DESTIDLENG.

Request next record number (ISSUE NOTE)

```
ISSUE NOTE
DESTID(data-value)
[DESTIDLENG(data-value)]
[VOLUME(data-value)
  [VOLUMELENG(data-value)]]
RRN
RIDFLD(data-area)

Conditions: FUNCERR, SELNERR,
UNEXPIN
```

This command is used to find the relative record number of the next record in an addressed direct data set. The number is returned in the data area specified in the RIDFLD option. The RRN option must be specified, because a relative record number is involved.

Wait for an operation to be completed (ISSUE WAIT)

```
ISSUE WAIT
[DESTID(data-value)
  [DESTIDLENG(data-value)]|
    [SUBADDR(data-value)]
      [CONSOLE|PRINT|CARD|
        WPMEDIA1|WPMEDIA2|
        WPMEDIA3|WPMEDIA4]]
[VOLUME(data-value)
  [VOLUMELENG(data-value)]]

Conditions: FUNCERR, SELNERR,
UNEXPIN
```

This command is used to cause task activity to be suspended until the previous batch data interchange command is completed. This command is meaningful only when it follows an ISSUE ADD, ISSUE ERASE, ISSUE REPLACE, or ISSUE SEND command. The options CONSOLE, PRINT, CARD, WPMEDIA1-4 are alternatives to DESTID and DESTIDLENG.

Batch data interchange options

CARD
specifies that the output medium is a card reader/punch device. This option is not valid with DESTID and DESTIDLENG.

CONSOLE
specifies that the output medium is that provided for messages to the operator. This option is not valid with DESTID and DESTIDLENG.

DEFRESP
specifies that all terminal control commands issued as a result of the batch data interchange command will request a definite response from the outboard batch program, irrespective of the specification of message integrity for the CICS task (by the system programmer).

DESTID(data-value)
specifies the name of the data set in the outboard destination. The data value must be a character string of up to eight characters. This option is not valid with CONSOLE, CARD, PRINT, or WPMEDIA1-4.

DESTIDLENG(data-value)
specifies the length of the name specified in the DESTID option as a halfword binary value. This option is not valid with CONSOLE, CARD, PRINT, or WPMEDIA1-4.

FROM(data-area)
specifies the data that is to be written to the data set.

INTO(data-area)
specifies the receiving field for the data read from the data set. The INTO option implies move-mode access.

KEYLENGTH(data-value)
specifies the length of the key specified in the RIDFLD option as a halfword binary value.

KEYNUMBER(data-value)
specifies the number, as a halfword binary value, of the index to be used to locate the record. There can be eight indexes (1 through 8). The default is 1. If the number is invalid, the FUNCERR condition will be raised. This option applies only to DPCX/DXAM and is mutually exclusive with RRN.

LENGTH(parameter)
specifies a halfword binary value to be used with ISSUE ADD, ISSUE RECEIVE, ISSUE REPLACE, and ISSUE SEND commands.

For an ISSUE ADD, ISSUE REPLACE, or ISSUE SEND command, the parameter must be a data value that is the length of the data that is to be written.

For an ISSUE RECEIVE command with the INTO option, the parameter 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.

For an ISSUE RECEIVE command with the SET option, the parameter must be a data area. On completion of the retrieval operation, the data area is set to the length of the data.

NOWAIT
specifies that the CICS task will continue processing without waiting for the batch data interchange command to complete. If this option is not specified, the task activity will be suspended until the command is completed.

NUMREC(data-value)
for a relative data set, specifies as a halfword binary value the number of logical records to be added, replaced, or deleted. Records are replaced sequentially starting with the one identified by the RIDFLD option.

For an indexed data set, NUMREC cannot be specified because only one record is replaced.

PRINT
specifies that the output is to the print medium.

RIDFLD(data-area)
specifies the record identification field for use with ISSUE REPLACE and ISSUE ERASE commands; it also specifies a data area in which the relative record number of the next record is returned in an ISSUE NOTE command.

For ISSUE REPLACE, ISSUE ADD, or ISSUE ERASE commands for a relative data set, the RIDFLD option must specify a fullword binary integer being the relative record number (starting from zero) of the record. The RRN option is also required.

For ISSUE REPLACE and ISSUE ERASE commands for an indexed data set, the RIDFLD option specifies the key which is embedded in the data specified by the FROM option. The KEYLENGTH option is also required.

RRN
specifies that the record identification field specified in the RIDFLD option contains a relative record number. If the option is not specified, RIDFLD is assumed to specify a key.

SET(ptr-ref)
specifies the pointer reference that is to be set to the address location of the data read from the data set. The SET option implies locate-mode access.

SUBADDR(data-value)
specifies the medium subaddress as a decimal value (in the range 0 through 15) which allows media of the same type, for example, 'printer 1' or 'printer 2', to be defined. Value 15 means a medium of any type. The default is 00.

VOLUME(data-value)
specifies the name of a diskette in an outboard destination that contains the data set specified in the DESTID option. The data value must be a character string of up to six characters.

VOLUMELENG(data-value)
specifies the length of the name specified in the VOLUME option as a halfword binary value.

WPMEDIA1 through WPMEDIA4
specifies that for each specific LUTYPE4 device, a word processing medium is defined to relate to a specific input/output device.

Batch data interchange exceptional conditions

DSSTAT
occurs when the destination status changes in one of the following ways:

- The data stream is aborted.
- The data stream is suspended.

Default action: terminate the task abnormally.

EODS
occurs when the end of the data set is encountered.

Default action: terminate the task abnormally.

IGREQCD
occurs when an attempt is made to execute an ISSUE SEND command after a SIGNAL RCD data-flow control code has been received from an LUTYPE4 logical unit.

Default action: terminate the task abnormally.

FUNCERR
occurs when an error occurs during execution of the command. Destination selection is unaffected and other commands for the same destination may be successful.

Default action: terminate the task abnormally.

LENGERR
occurs if the length of the retrieved data is greater than the value specified by the LENGTH option for a move-mode ISSUE RECEIVE command.

Default action: terminate the task abnormally.

SELNERR
occurs when an error occurs during destination selection. The destination is not selected and other commands for the same destination are unlikely to be successful.

Default action: terminate the task abnormally.

UNEXPIN
occurs when some unexpected or unrecognized information is received from the outboard controller.

Default action: terminate the task abnormally.

Part 4. Control operations

Chapter 4.1. Introduction to control operations 273

Chapter 4.2. Interval control 275

Chapter 4.3. Task control 285

Chapter 4.4. Program control 289

Chapter 4.5. Storage control 299

Chapter 4.6. Transient data control 301

Chapter 4.7. Temporary storage control 305

Chapter 4.1. Introduction to control operations

This part of the manual collects together several groups of operations that are not specifically database or data communication operations, but that control the execution of tasks within a CICS system. These groups of operations are as follows:

- Interval control – comprising functions whose execution is dependent on time.
- Task control – comprising functions to temporarily relinquish control or to synchronize resource usage.
- Program control – comprising functions affecting the flow of control between application programs.
- Storage control – comprising functions to obtain and release areas of main storage.
- Transient data control – comprising functions for the transfer of data between CICS tasks and between the CICS region and other regions.
- Temporary storage control – comprising functions for the temporary storage of data.

Each group of operations is described in a separate chapter within this part.

Chapter 4.2. Interval control

The CICS interval control program, in conjunction with a time-of-day clock maintained by CICS, provides functions that can be performed at the correct time; such functions are called **time-controlled** functions. The time of day is obtained from the operating system at intervals whose frequency, and thus the accuracy of the time-of-day clock, depends on the task mix and the frequency of task switching operations.

Using interval control commands you can:

- Request the current date and time of day (ASKTIME)
- Select the format of date and time (FORMATTIME)
- Delay the processing of a task (DELAY)
- Request notification when specified time has expired (POST)
- Wait for an event to occur (WAIT EVENT)
- Start a task and store data for the task (START)
- Retrieve data stored (by a START command) for a task (RETRIEVE)
- Cancel the effect of previous interval control commands (CANCEL).

Exceptional conditions that occur during execution of an interval control command are handled as described in "Chapter 1.5. Exceptional conditions" on page 43.

Expiration times: The time at which a time-controlled function is to be started is called the **expiration time**. You can specify expiration times absolutely, as a time of day, or as an interval that is to elapse before the function is to be performed.

An interval is measured relative to the current time and so the expiration time will always be after the current time (assuming a nonzero interval is specified). An absolute time is measured relative to midnight prior to the current time and may therefore be prior to the current time.

CICS treats as expired a request for an absolute time that is equal to the current time or that precedes the current time by up to 6 hours. If you specified an absolute time, and it precedes the current time by more than 6 hours, CICS adds 24 hours, that is, the requested function is performed at the time you specified but on the next day.

Examples of the START command specifying absolute time-of-day requests, are as follows:

```
EXEC CICS START TIME(123000)
```

This command, issued at 1000 hours on Monday, will expire at 1230 hours on the same Monday.

```
EXEC CICS START TIME(090000)
```

This command, issued at 1000 hours on Monday, will expire immediately because the specified time is within the preceding 6 hours.

```
EXEC CICS START TIME(020000)
```

This command, issued at 1000 hours on Monday, will expire at 0200 hours on Tuesday because the specified time is more than 6 hours before the current time.

```
EXEC CICS START TIME(330000)
```

This command, issued at 1000 hours on Monday, will expire at 0900 hours on Tuesday.

Because each end of an intersystem link may be in a different time zone, you must use the INTERVAL form of expiration time when the transaction to be started is in a remote system.

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 DELAY, POST, or START command. You can specify your own request identifier by means of the REQID option; if you do not, CICS assigns (for POST and START only) a unique request identifier and places it in field EIBREQID in the EXEC interface block (EIB). Specify a request identifier if you want the request to be canceled at some later time by a CANCEL command.

Request current date and time of day (ASKTIME)

```
ASKTIME
[ABSTIME(data-area)]
```

You use this command to update the date and CICS time-of-day clock, and the fields EIBDATE and EIBTIME in the EIB. These two fields contain initially the date and time when the task started. The command returns the current time in the form of the number of milliseconds since 0000 hours on January 1, 1900. See Appendix A, "EXEC interface block" on page 339 for details of the EIB.

The following example shows you how the ABSTIME option works:

```
EXEC CICS ASKTIME ABSTIME(utime)
```

After execution, 'utime' might contain the value 002694057952138 in milliseconds.

Select the format of date and time (FORMATTIME)

```
FORMATTIME
ABSTIME(data-value)
[YYDDD(data-area)]
[YYMMDD(data-area)]
[YYDDMM(data-area)]
[DDMMYY(data-area)]
[MMDDYY(data-area)]
[DATE(data-area)]
[DATEFORM(data-area)]
[DATESEP[(data-area)]]
[DAYCOUNT(data-area)]
[DAYOFWEEK(data-area)]
[DAYOFMONTH(data-area)]
[MONTHOFYEAR(data-area)]
[YEAR(data-area)]
[TIME(data-area)
  [TIMESEP[(data-area)]]]
```

You use this command to transform the absolute date and/or time into any of a variety of formats, as described in the list of options at the end of the chapter.

The following example shows the effect of some of the options of the command:

```
EXEC CICS FORMATTIME ABSTIME(utime)
          DATESEP('-') DDMMYY(date)
          TIME(time) TIMESEP
```

After execution, 'date' would contain 15-05-85, and 'time' would contain 08:12:32.

Delay processing of a task (DELAY)

```
DELAY
[INTERVAL(hhmmss)[1]|TIME(hhmmss)]
[REQID(name)]

Conditions: EXPIRED, INVREQ
```

[1] INTERVAL(0) is the default

You use this command to suspend 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.

The following example shows you how to suspend the processing of a task for 5 minutes:

```
EXEC CICS DELAY
     INTERVAL(500)
     REQID('GXLBZQMR')
```

The following example shows you how to suspend the processing of a task until 1245 hours:

```
EXEC CICS DELAY
     TIME(124500)
     REQID('UNIQCODE')
```

Request notification when specified time has expired (POST)

```
POST
[INTERVAL(hhmmss)[1]|TIME(hhmmss)]
SET(ptr-ref)
[REQID(name)]

Conditions: EXPIRED, INVREQ
```

[1] INTERVAL(0) is the default

You use this command to request notification that a specified time has expired. 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. This area is available for the duration of the task issuing the POST command.

When the time you specified 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 of 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, described in "Chapter 4.3. Task control" on page 285.
- By suspending task activity by 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 — WAIT EVENT 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 is unpredictable.

  However, other tasks can CANCEL the event if they have access to the REQID associated with the POST command. (See CANCEL command and description of 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 previously issued POST command by the task.

The following example shows you 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 you 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)
```

Wait for an event to occur (WAIT EVENT)

```
WAIT EVENT
ECADDR(ptr-value)

Condition: INVREQ
```

You use this command to synchronize 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 the preceding section. The WAIT EVENT command provides a method of directly relinquishing control to some other task until the event being waited on is completed.

You must specify, in the ECADDR option, a pointer value giving the address of an event control area, which must conform to the format and standard posting conventions for an event control block (ECB); it will normally be the timer event control area created by a POST command. For a program executing in 31-bit mode on MVS/XA, the INVREQ condition will be raised if the specified event control area address is above the 16-megabyte line.

The following example shows you how to suspend processing of a task until the specified event control area is posted:

```
EXEC CICS WAIT EVENT ECADDR(PVALUE)
```

Start a task (START)

You use the START command to start a task, on a local or remote system, at a specified time. The starting task may pass data to the started task and may also specify a terminal to be used by the started task as its principal facility. The TRANSID, TERMID, and FROM options specify the transaction to be executed, the terminal to be used, and the data to be used, respectively.

CEDF is an exception to the START command – it is not valid as a TRANSID name, so do not attempt to start CEDF in this way.

You can specify the FMH option if the FROM option is specified. The FMH option indicates that the data, to be passed to the started task, contains function management headers.

The syntax of the command is as follows:

```
START
[INTERVAL(hhmmss)[1]|TIME(hhmmss)]
TRANSID(name)
[REQID(name)]
[FROM(data-area)
  LENGTH(data-value)[FMH]]
[TERMID(name)]
[SYSID(name)]
[RTRANSID(name)]
[RTERMID(name)]
[QUEUE(name)]
[NOCHECK]
[PROTECT]

Conditions: INVREQ, IOERR,
ISCINVREQ, NOTAUTH, SYSIDERR,
TERMIDERR, TRANSIDERR

[1] INTERVAL(0) is the default
```

Further data may be passed to the started task in the RTRANSID, RTERMID, and QUEUE options. For example, 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 data is to be passed, it will be queued using the request identifier specified in the REQID option, if one is provided. This identifier should be recoverable (in temporary storage terms) if the PROTECT option is also specified, or nonrecoverable if PROTECT is not specified, otherwise unpredictable results can occur. Such problems cannot | occur if REQID is not used. An IOERR will occur if a | START operation uses a REQID(name) that already exists.

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 will be returned, whereas those for tasks to be started on a remote system will not be returned. 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.

START commands are queued by means of the LOCALQ operand of the DFHPCT TYPE=REMOTE system macro as described in the *CICS/MVS Resource Definition (Macro)* manual, or by means of the LOCALQ operand of the TRANSACTION definition as described in the *CICS/MVS Resource Definition (Online)* manual.

One or more constraints have to be satisfied before the transaction to be executed can be started, as follows:

1. The specified interval must have elapsed or the specified expiration time must have been reached (see "Expiration times" on page 275). The INTERVAL option 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.
2. If the TERMID option is specified, the named terminal must be available.
3. If the PROTECT option is specified, the starting task must have taken a successful sync point. This option, coupled to extensions to system tables, reduces the exposure to lost or duplicated data caused by failure of a starting task.
4. 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. This is done by the DATASTR and RECFM operands of the DFHTCT TYPE=SYSTEM system macro. For CICS-CICS, these are always the default values. For CICS-IMS, care should be taken to specify the correct values.

Execution of a START command naming a transaction in the local system will supersede any outstanding POST command executed by the starting task.

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 example shows how to start a specified task, not associated with a terminal, in one hour:

```
EXEC CICS START
     TRANSID('TRNL')
     INTERVAL(10000)
     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 prior to terminal availability.

The following example shows how to request initiation of a task associated with a terminal. Because no request identifier is specified in this example, CICS assigns one and returns it to the application program in the EIBREQID field in the EXEC interface block.

```
EXEC CICS START
     TRANSID('TRN1')
     TIME(185000)
     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 through execution of a RETRIEVE command as described later in the chapter.

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 will ultimately cause the new task to be started and allow it to retrieve the data specified on the command. The new task will also be 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 will be started and will be able to retrieve the outstanding data. If this second new task fails to retrieve the outstanding data, a third task will be started, and so on, up to a maximum of 5 times, after which, the data will be lost.

The following example shows 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)
```

Retrieve data stored for a task (RETRIEVE)

```
RETRIEVE
[INTO(data-area)|SET(ptr-ref)]
[LENGTH(data-area)]
[RTRANSID(data-area)]
[RTERMID(data-area)]
[QUEUE(data-area)]
[WAIT]

Conditions: ENDDATA, ENVDEFERR,
INVREQ, INVTSREQ, IOERR, LENGERR,
NOTAUTH, NOTFND
```

You use this command to retrieve data stored by expired START commands. It is the only method available for accessing such data.

The INTO option specifies the area into which the data is to be placed.

You must specify, in the LENGTH option, a data area that contains the maximum length of record that the application program will accept. If the record length exceeds the specified maximum, it is truncated and the LENGERR condition occurs. After the retrieval operation, the data area specified in the LENGTH option is set to the record length (before any truncation occurred).

Alternatively, a pointer reference can be specified in the SET option. CICS then acquires an area large enough to hold the record and sets the pointer reference to the address of that area. After the retrieval operation, the data area specified in the LENGTH option is set to the record length.

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 upon execution of the RETRIEVE command, or upon 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 the START command that started the task; it does so by issuing consecutive RETRIEVE commands. Expired data records are presented to the task upon 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 exceptional 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.

The WAIT option specifies that, if all expired data records have already been retrieved, the task is suspended until further expired data records become available. The transaction that issues a WAIT option must be running on a terminal and, to pass an expired data record, you need to issue a START command from some other transaction that explicitly states the terminal id as well as the transaction id.

The ENDDATA exceptional condition will be raised:

- If no data is available after the deadlock time interval (as specified in the DTIMOUT operand of the DFHPCT TYPE=ENTRY system macro).
- If CICS enters shutdown and the transaction is still suspended. An attempt to reissue the RETRIEVE command with the WAIT option after this event (that is, system shutdown) will cause an abend with a code of AICB.

If the retrieved data contains FMHs, as specified by the FMH option on the associated START command, field EIBFMH in the EIB will be set to X'FF'. If no FMH is present, EIBFMH will be set to X'00'.

If an input/output error occurs during a retrieval operation, the IOERR exceptional condition occurs. The operation can be retried by reissuing the RETRIEVE command.

The following example shows how to retrieve data stored by a START command for the task, and store it in the user provided data area called DATAFLD.

```
EXEC CICS RETRIEVE
     INTO(DATAFLD)
     LENGTH(LENG)
```

The following example shows how to request retrieval of a data record stored for a task into a data area provided by CICS; the pointer reference (PREF) specified by the SET option is set to the address of the storage area reserved for the data record.

```
EXEC CICS RETRIEVE
     SET(PREF)
     LENGTH(LENG)
```

Cancel interval control requests (CANCEL)

```
CANCEL
[REQID(name)
  [TRANSID(name)][SYSID(name)]]

Conditions: INVREQ, ISCINVREQ,
NOTAUTH, NOTFND, SYSIDERR
```

You use this command to cancel 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, will determine 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 prior to its expiration, 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 should 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 prior to expiration of the original command.

Interval control options

ABSTIME(parameter)
For the ASKTIME command, "parameter" specifies the user data area into which the time, in milliseconds since 0000 hours on January 1, 1900, is to be stored.

For the FORMATTIME command, "parameter" specifies the data value, in milliseconds since 0000 hours on January 1, 1900, which is to be converted to an alternative format.

The format of the parameter is:

```
ASM:    PL8
COBOL:  PIC S9(15) COMP-3
PL/I:   FIXED DEC(15)
```

DATE(data-area)
specifies the variable that is to receive the date in the format specified in the SIT DATFORM operand. The returned value is in 8 character format, and a separator is present or not depending on the DATESEP option. You normally use this option only when a date is needed for output purposes. Where a date is needed for analysis, you must request the date in explicit form, for example, MMDDYY.

DATEFORM(data-area)
specifies the format of the installation defined date. CICS returns YYMMDD, DDMMYY, or MMDDYY (6 characters) according to the DATFORM operand of the DFHSIT system macro.

DATESEP(data-value)
specifies as a single character field the character to be inserted as the separator between the year and the month, between the day and the month, or between the year and the day if form YYDDD 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 in 'data-area' the number of days since January 1, 1900 (day 0) as a 31-bit binary number. You will find this useful if you need to compare the current date with a previous date that has, for example, been stored in a data set.

DAYOFWEEK(data-area)
returns in 'data-area' the relative day number of the week as a 31-bit binary number. Sunday = 0, Saturday = 6. This number can be converted to a textual form of day in any language.

DAYOFMONTH(data-area)
returns in "data-area" the relative day number of the month as a 31-bit binary number.

DDMMYY(data-area)
specifies the user field (8 characters) in which CICS is to return the date, for example, 21/10/84.

ECADDR(ptr-value)
specifies the address of the timer event control area that must be posted before task activity can be resumed.

FMH
specifies that the user data to be passed to the started task contains function management headers.

FROM(data-area)
specifies the data that is to be stored for a task that is to be started at some future time.

INTERVAL(hhmmss)
specifies the expiration time for an interval control function as an interval of time that is to elapse from the time at which the interval control command is issued. The time specified is added to the current clock time by CICS when the command is executed to calculate the expiration time.

This option is used in DELAY commands (to specify the time for which the task should be suspended), POST commands (to specify when the posting of the timer event control area should occur), and START commands (to specify when a new task should be started).

'hhmmss' can be replaced by a decimal constant; or, for ASM, by a reference to a field defined as PL4; for COBOL, by a data name of the form PIC S9(7) COMP-3; or for PL/I, by an expression that can be converted to FIXED DEC(7,0). The value must be of the form 0HHMMSS+ where 'HH' represents hours from 00 through 99, 'MM' represents minutes from 00 through 59, and 'SS' represents seconds from 00 through 59.

INTO(data-area)
specifies the user data area into which retrieved data is to be written. If this option is specified, move-mode access is implied.

LENGTH(parameter)
specifies a halfword binary value to be used with START and RETRIEVE commands.

For a START command, the parameter must be a data value that is the length of the data that is to be stored for the new task.

For a RETRIEVE command with the INTO option, the parameter 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.

For a RETRIEVE command with the SET option, the parameter must be a data area. On completion of the retrieval operation, the data area is set to the length of the data.

MMDDYY(data-area)
specifies the 8 character user field in which CICS is to return the date, for example, 07/10/84.

MONTHOFYEAR(data-area)
data-area is set to the relative month number of the year as a 31-bit binary number. January=1, December=12. You can convert this number, in your application program, to the name of the month in any language.

NOCHECK
specifies that, for a remote system, CICS should optimize the execution of the START command to improve performance by providing less error checking and slightly less function. For more information, see the section on improving the performance of intersystem START requests in the *CICS/MVS Intercommunication Guide*.

PROTECT
specifies that, in addition to the constraints described earlier in the chapter, the new task will not be started until the starting task has taken a sync point. If the starting task abends before the sync point is taken, the request to start the new task is canceled. If the REQID option is specified as well, 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 synchronization point (syncpoint). For more information on the PROTECT option with remote transactions, see the *CICS/MVS Intercommunication Guide*.

QUEUE{(name)|(data area)}
when used in a START command, "name" specifies the name of the queue that may be used by the transaction specified also in the START command. The name can be up to 8 characters in length.

When used in a RETRIEVE command, "data area" specifies the name of the queue that may be accessed by the transaction issuing the RETRIEVE command. The data area must be 8 characters in length.

REQID(name)
specifies a unique name (up to 8 characters) to identify a command. This name will be 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 START command will be processed. The START command will be processed on the system identified by the SYSID option or on the local system if SYSID is omitted.

If specified in a START command that also specifies FROM, the data in the FROM option is stored in temporary storage using the name specified in REQID as the name of the temporary storage queue.

This option can be used in a DELAY, POST, or START command when another task is to be provided with the capability of canceling an unexpired command; and in CANCEL commands, except those canceling a POST command issued by the same task (for which, the REQID option is ignored if it is specified).

If this option is omitted from a POST command, CICS generates a unique request identifier in the EIBREQID field of the EXEC interface block. This applies also to a START command unless the NOCHECK option is specified, in which case field EIBREQID is set to blanks and cannot be used subsequently to cancel the START command.

RTERMID{(name)|(data area)}

When used in a START command, “name” specifies a value, for example a terminal name, that may be retrieved when the transaction, specified in the TRANSID option in the START command, is started. The name can be up to 4 characters in length.

When used in a RETRIEVE command, “data area” specifies an area which may be used in the TERMID option of a START command that may be executed subsequently. The data area must be 4 characters in length.

RTRANSID{(name)|(data area)}

When used in a START command, “name” specifies a value, for example a transaction name, that may be retrieved when the transaction, specified in the TRANSID option in the START command, is started. The name can be up to 4 characters in length.

When used in a RETRIEVE command, “data area” specifies an area which may be used in the TRANSID option of a START command that may be executed subsequently. The data area must be 4 characters in length.

SET(ptr-ref)

When used with a POST command, SET specifies the 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'.

When used with a RETRIEVE command, SET specifies the pointer reference to be set to the address of the retrieved data. If this option is specified, locate-mode access is implied.

SYSID(name) remote systems only

specifies the name of the system whose resources are to be used for intercommunication facilities. The name can be up to 4 characters in length. The START command will be executed on the specified system.

TERMID(name)

specifies the symbolic identifier of the terminal associated with a transaction to be started as a result of a START command. This option is required when the transaction to be started must communicate with a terminal; it should be omitted otherwise. The name must be alphanumeric, up to 4 characters in length, and must have been defined in the terminal control table (TCT) by the system programmer.

The terminal identifier must be defined as either a local or a remote terminal in the TCT on the system in which the START command is executed, ***when the start of the transaction takes effect.***

TIME(parameter)

The parameter ‘hhmmss’ specifies the expiration time for an interval control function. See the section ‘Expiration Times’ earlier in the chapter.

This option is used in DELAY commands (to specify the time for which the task should be suspended), POST commands (to specify when the posting of the timer event control area should occur), and START commands (to specify when a new task should be started).

‘hhmmss’ can be replaced by a decimal constant; or, for ASM, by a reference to a field defined as PL4; for COBOL, by a data name of the form PIC S9(7) COMP-3; or for PL/I, by an expression that can be converted to FIXED DEC(7,0). The value must be of the form 0HHMMSS+ where “HH” represents hours from 00 through 99, “MM” represents minutes from 00 through 59, and “SS” represents seconds from 00 through 59.

When used in a FORMATTIME command, “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 determined 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 6 bytes are returned in an 8 character field. If you omit the ‘data-value’, a colon : is used as a separator.

TRANSID(name)

specifies the symbolic identifier of the transaction to be executed by a task started as the result of a START command, or to be canceled by a CANCEL command. The name can be up to 4 characters in length and must have been defined in the program control table (PCT) by the system programmer.

If SYSID is specified, the transaction is assumed to be on a remote system irrespective of whether or not the name is defined in the PCT. Otherwise the entry in the PCT will be used to determine if the transaction is on a local or remote system.

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. The ENDDATA condition will be raised only if CICS is shut down before any expired data records become available.

YEAR(data-area)
specifies the full number of the year as a 31-bit binary number, for example, 1984, 2001.

YYDDD(data-area)
specifies the user field (6 characters) in which CICS is to return the date, for example, 84/301.

YYDDMM(data-area)
specifies the user field (8 characters) in which CICS is to return the date, for example, 84/30/10.

YYMMDD(data-area)
specifies the user field (8 characters) in which CICS is to return the date, for example, 84/10/21.

Interval control exceptional conditions

ENDDATA
occurs if any of the following situations exists:

- No more data is stored for a 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 which did not specify the FROM option.
- The RETRIEVE command is issued by a task that is not started by a START command.

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.

EXPIRED
occurs if the time specified in a POST or DELAY command has already expired when the command is issued.

Default action: ignore the condition.

INVREQ
occurs if an invalid type of interval control command is received for processing by CICS, or if the ECB resides above the 16-megabyte line.

Default action: terminate the task abnormally.

INVTSREQ
occurs if there is no support for a temporary storage read request issued by CICS during execution of a RETRIEVE command. This situation can occur when a dummy temporary storage program is included in the system by the system programmer in place of a functional temporary storage program.

Default action: terminate the task abnormally.

IOERR
occurs if an input/output error occurs during a RETRIEVE or START operation. The operation can be retried by reissuing the RETRIEVE command.

This condition also occurs if a START operation:

- Attempts to write to temporary storage and the temporary storage data set is full.
- Uses a REQID(name) that already exists.

Default action: terminate the task abnormally.

ISCINVREQ
occurs when the remote system indicates a failure which does not correspond to a known condition.

Default action: terminate the task abnormally.

LENGERR
occurs in move-mode retrieval if the length specified is less than the actual length of the stored data.

Default action: terminate the task abnormally.

NOTAUTH
occurs when a resource security check has failed. Use of SYSID will always raise the NOTAUTH condition when resource security level checking is in effect (RSLC=YES in the PCT). The reasons for the failure are the same as for abend code AEY7, as described in the *CICS/MVS Messages and Codes* manual.

Default action: terminate the task abnormally.

NOTFND
occurs if any of the following situations exists:

- The request identifier specified in a CANCEL command fails to match an unexpired interval control command.
- A RETRIEVE command is issued but some prior task retrieved the data stored under the request identifier directly through temporary storage requests and then released the data.
- The request identifier associated with the START command is not unique; when a RETRIEVE command is issued, CICS cannot find the data.

Default action: terminate the task abnormally.

SYSIDERR

occurs when the SYSID option specifies either a name which is not defined in the intersystem table or a system to which the link is closed.

Default action: terminate the task abnormally.

TERMIDERR

occurs in either of the following cases:

- The terminal identifier in a START command cannot be found in the terminal control table.
- A START command has defined TERMID to have the same value as an LUTYPE62 conversation identifier (the CONVID returned by a previous ALLOCATE command).

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.

Chapter 4.3. Task control

The CICS task control program provides functions that synchronize task activity, or that control the use of resources.

CICS processes tasks according to priorities assigned by the system programmer. Control of the processor 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 user-written application programs.

Task control commands are provided to:

- Suspend a task (SUSPEND).
- Schedule the use of a resource by a task (ENQ and DEQ).

A task can issue the SUSPEND command to relinquish control and allow tasks higher on the active chain to proceed. This facility can be used to prevent processor-intensive tasks from monopolizing the processor. As soon as no other task higher on the active chain is waiting to be processed, control is returned to the issuing task; that is, the task remains dispatchable.

Scheduling the use of a resource by a task is sometimes useful in order to protect the resource from concurrent use by more than one task, that is, to make the resource serially reusable. Each task that is to use the resource issues an ENQ (enqueue) command. The first task to do so has the use of the resource immediately, but subsequent ENQ commands for the resource, issued by other tasks, result in those tasks being suspended until the resource is available. Each task using the resource should issue a DEQ (dequeue) command when it has finished with it. The resource then becomes available and the next task to have issued an ENQ command is resumed and given use of the resource. The other tasks obtain the resource in turn, in the order in which they enqueued upon it.

The exceptional condition, ENQBUSY, that can occur during execution of a task control command is handled as described in "Chapter 1.5. Exceptional conditions" on page 43.

Suspend a task (SUSPEND)

```
SUSPEND
```

This command is used to relinquish control to a task of higher dispatching priority. Control is returned to the task issuing the command as soon as no other task of a higher priority is ready to be processed.

Schedule use of a resource by a task (ENQ and DEQ)

```
ENQ
RESOURCE(data-area)
[LENGTH(data-value)]
[NOSUSPEND]

Conditions: ENQBUSY, LENGERR
```

```
DEQ
RESOURCE(data-area)
[LENGTH(data-value)]

Conditions: LENGERR
```

The ENQ and DEQ commands can be used to enqueue upon and dequeue from a resource that is to be protected from concurrent use by more than one task.

The ENQ command 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.

The ENQBUSY condition allows a conditional ENQ to be used. If a resource is not available when enqueued, the ENQBUSY condition is raised. The execution of a HANDLE CONDITION ENQBUSY command will return control to the task at the ENQBUSY label, without waiting for the resource to become available.

The DEQ command causes a resource currently enqueued upon by the task to be released for use by other tasks. If a task enqueues upon a resource but does not dequeue from it, CICS automatically releases the resource during sync point processing or when the task is terminated.

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 upon must be identified by one of the following methods:

- Specifying a data area that is the resource.
- Specifying a data variable that contains a unique character-string argument (for example, an employee name) that represents the resource. The character string may be up to 255 bytes in length. The length of the string must be supplied in the LENGTH option.

When issuing the DEQ command, the resource to be dequeued from must be identified by the method used when enqueuing upon the resource. If no enqueue has been issued for the resource, the dequeue will be ignored.

The following examples show how to enqueue upon a resource using the two methods shown above. Substituting "DEQ" for "ENQ" in these examples illustrates the ways in which a resource can be released.

```
EXEC CICS ENQ
     RESOURCE(RESNAME)

        or

EXEC CICS ENQ
     RESOURCE(SOCSECNO)
     LENGTH(9)
```

Task control options

LENGTH(data-value)
specifies that the resource to be enqueued upon (or dequeued from) is a data variable of length given by the data value. The data value is a halfword binary value in the range 1 through 255. 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. This option is required if the resource is specified as a character string; it should not be specified otherwise.

NOSUSPEND
specifies that application program suspension for the ENQBUSY condition is to be inhibited. This condition will be handled as described in "Chapter 1.5. Exceptional conditions" on page 43.

RESOURCE(data-area)
specifies either the resource to be enqueued upon (or dequeued from) or a data variable that contains a character string (for example an employee name) that represents the resource. In the latter case, the length of the string must be specified by the LENGTH option.

Task control exceptional condition

ENQBUSY
occurs when an ENQ command specifies a resource that is unavailable.

Default action: wait for the resource to become available.

LENGERR
occurs when the LENGTH option is outside the range 1 through 255.

Default action: terminate the task abnormally.

Controlling access sequence

Using the ENQ/DEQ mechanism alone, you cannot ensure that two or more tasks issuing ENQ and DEQ commands will access a resource in a given sequence relative to each other. If you wish two or more tasks to access a .esource in a specific order, use one or more EXEC CICS WAIT commands in conjunction with one or more hand-posted ECBs.

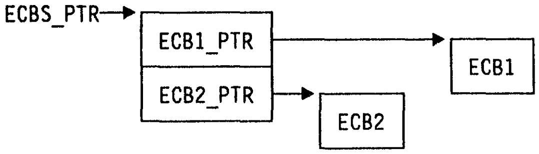
To hand-post an ECB, a CICS task sets a four-byte field to either the cleared state of binary zeroes, or the posted state of X'40008000'. The task can use an EXEC CICS START command to start another task and pass the address of the ECB. The started task receives the address through an EXEC CICS RETRIEVE command.

Either task can set the ECB or wait on it. Use the ECB to control the sequence in which the tasks access resources. Two tasks can share more than one ECB if necessary. You can extend this technique to the control of as many tasks as you wish.

Notes:

1. Only one task can wait on an ECB at one time.
2. In an MVS/XA environment, always create ECBs at addresses below the 16MB line.

If two tasks need to access a resource once each in a specified order, one ECB is sufficient. In the more realistic case, where repeated alternate access is required, two ECBs are necessary. Figure 28 on page 287 gives an example of the logic required. The example uses two ECBs, ECB1 and ECB2, addressed by the pointers ECB1_PTR and ECB2_PTR, which are held in an eight-byte area addressed ECBS_PTR.



These tasks will go on exchanging data through the T/S queue for ever. In practice some code would be included to close the process down in an orderly way.

```
      TASK A                               TASK B
      ------                               ------

Delete T/S queue
Clear ECB1 (set to X'00000000')
Clear ECB2
EXEC CICS START TASK B and pass
the pointer to the list of ECB
pointers, ECBS_PTR.                 EXEC CICS RETRIEVE the
                                    pointer ECBS_PTR.

START OF LOOP                       START OF LOOP
  EXEC CICS WAIT EVENT                Write to T/S queue
           ECADDR(ECB1_PTR)           Post ECB1 (set to X'40008000')
  Clear ECB1                          EXEC CICS WAIT EVENT
  Read T/S queue                               ECADDR(ECB2_PTR)
  < act on data from queue >          Clear ECB2
  Delete T/S queue                    Read T/S queue
  Write to T/S queue                  < act on data from queue >
  Post ECB2                           Delete T/S queue
GO TO START OF LOOP                 GO TO START OF LOOP
```

Figure 28. Example of task control

Chapter 4.4. Program control

The CICS program control program governs the flow of control between application programs in a CICS system. The name of an application program referred to in a program control command must have been defined as a program to CICS.

Program control commands are provided to:

- Link one user-written application program to another, anticipating subsequent return to the requesting program (LINK). The COMMAREA option allows data to be passed to the requested application program.
- Transfer control from one user-written application program to another, with no return to the requesting program (XCTL). The COMMAREA option allows data to be passed to the requested application program.
- Return control from one user-written application program to another or to CICS (RETURN). The COMMAREA option allows data to be passed to a newly-initiated transaction.
- Load a designated application program, table, or map into main storage and return control to the requesting program (LOAD).
- Delete a previously loaded application program, table, or map from main storage (RELEASE).

Exceptional conditions that occur during execution of a program control command are handled as described in "Chapter 1.5. Exceptional conditions" on page 43.

The HANDLE ABEND command can be used to deal with abnormal terminations. See "Chapter 5.2. Abnormal termination recovery" on page 313 for further information about this command.

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 anticipating return (LINK)

```
LINK
PROGRAM(name)
[COMMAREA(data-area)
  [LENGTH(data-value)]]

Condition: NOTAUTH, PGMIDERR,
LENGERR
```

This 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 will be loaded. When the RETURN command is executed in the linked-to program, control is returned to the program initiating the linkage at the next sequential executable instruction.

The following example shows how to request a link to an application program called PROG1:

```
EXEC CICS LINK PROGRAM('PROG1')
```

The COMMAREA option can be used to pass data to the linked-to program. For further details, see "Passing data to other programs" on page 292, the description of COMMAREA on page 296, and the description of TRANSID on page 296.

The LENGTH option specifies the length of the data being passed. The LENGTH value being passed must not be greater than the length of the data area specified in the COMMAREA option. If it is, the results are unpredictable and may result in the LENGERR condition being set.

The linked-to program operates independently of the program that issues the LINK command with regard to handling exceptional conditions, attention identifiers, and abends. For example, the effects of HANDLE commands in the linking program are not inherited by the linked-to program, but the original HANDLE commands are restored on return to the linking program. Figure 29 on page 290 illustrates the concept of logical levels.

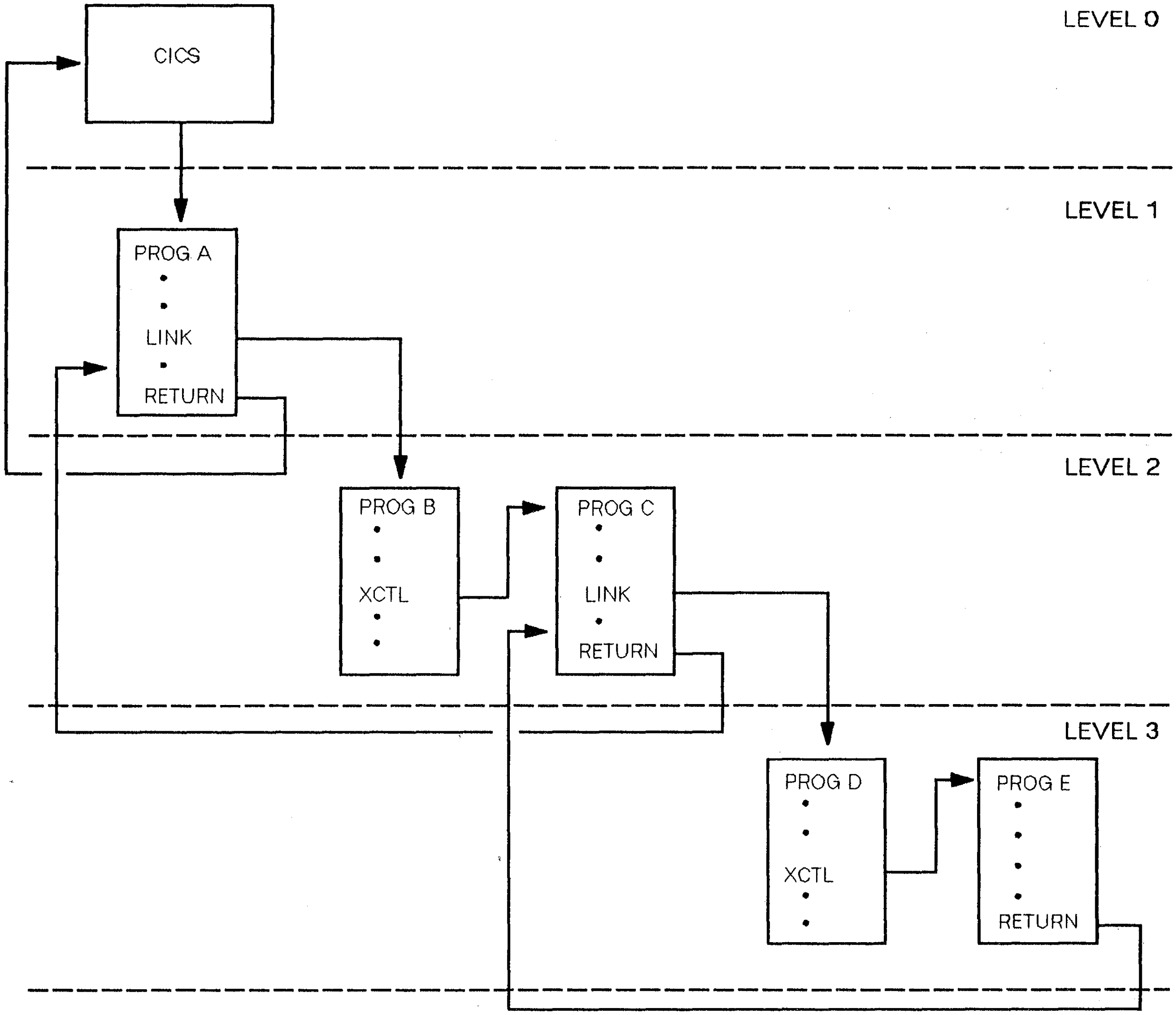


Figure 29. Application program logical levels

Transfer program control (XCTL)

```
XCTL
PROGRAM(name)
[COMMAREA(data-area)
  [LENGTH(data-value)]]

Condition: NOTAUTH, PGMIDERR, LENGERR
```

This command is used to transfer control from one application program to another at the same logical level. The program from which control is transferred is released.

If the program to which control is transferred is not already in main storage, it will be loaded.

The following example shows how to request a transfer of control to an application program called PROG2:

```
EXEC CICS XCTL PROGRAM('PROG2')
```

The COMMAREA option can be used to pass data to the invoked program. For further details, see "Passing data to other programs" on page 292.

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. If it is, the results are unpredictable and may result in the LENGERR condition being set.

Return program control (RETURN)

```
RETURN
[TRANSID(name)
  [COMMAREA(data-area)
    [LENGTH(data-value)]]]

Condition: INVREQ, LENGERR
```

This command is used to return control from an application program either to an application program at the next higher logical level or to CICS.

When the RETURN command is issued in a lower-level program (that is, a linked-to program), control will be passed back to the level that is 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 of the next transaction to be associated with that terminal. The transaction identifier will come into play only after the highest logical level has relinquished control to CICS using the RETURN command, and after input has been received from the terminal. Any input entered from the terminal, other than an attention key, will be interpreted wholly as data. In addition, the COMMAREA option can be used to pass data to the new task that will be started. For further details, see "Passing data to other programs" on page 292.

No resource security checking occurs on the RETURN TRANSID command. However, transaction security checking is still available when CICS attaches the returned transaction.

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. If it is, the results are unpredictable and may result in the LENGERR condition being set. The COMMAREA and LENGTH options can be used only when the RETURN command is returning control to CICS; the INVREQ condition will occur otherwise. If the COMMAREA option specifies a zero value as the address of the data area, INVREQ is returned.

Load a program (LOAD)

```
LOAD
PROGRAM(name)
[SET(ptr-ref)]
[LENGTH(data-area)|
  FLENGTH(data-area)]
[ENTRY(ptr-ref)]
[HOLD]

Condition: NOTAUTH, PGMIDERR
```

This command is used to fetch application programs, tables, or maps from the library where they reside and load them into main storage. This facility is used to load an application program that will be used repeatedly, thereby reducing system overhead through a single load, to load a table to which control is not to be passed, or to load a map to be used in a mapping operation. (See "Chapter 3.2-1. Introduction to basic mapping support" on page 135 for further details about maps.)

CICS sets the pointer reference specified in the SET option to the address of the loaded program, table, or map; if the LENGTH or FLENGTH option is specified, the data area provided will be set to the length involved. (See also the RELOAD operand of the DFHPPT TYPE=ENTRY macro as described in the *CICS/MVS Resource Definition (Macro)* manual, or the RELOAD attribute for the PPT as described in the *CICS/MVS Resource Definition (Online)* manual.)

If the HOLD option is specified, the loaded program, table, or map remains in main storage until a RELEASE command is issued; if HOLD is not specified, the program, table, or map remains in main storage until a RELEASE command is issued or until the task that issued the LOAD command is terminated normally or abnormally.

The following example shows how to load a user-prepared table called TB1:

```
EXEC CICS LOAD PROGRAM('TB1') SET(PTR)
```

Delete a loaded program (RELEASE)

```
RELEASE
PROGRAM(name)

Condition: NOTAUTH, PGMIDERR
```

This command is used to delete from main storage a program, table, or map previously loaded by a LOAD command. If the HOLD option is specified in the LOAD command, the loaded program is deleted only in response to a RELEASE command. If the HOLD option is not specified, the loaded program can be deleted by a

RELEASE, or it will be deleted automatically when the task that issued the LOAD is terminated.

The following example shows how to delete an application program, called PROG4, loaded in response to a LOAD command:

```
EXEC CICS RELEASE PROGRAM('PROG4')
```

Passing data to other programs

You can pass data to another program when control is passed to that other program by means of a program control command.

The COMMAREA option of the LINK and 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 the receiving program you must give this data area the name DFHCOMMAREA.

In a similar manner, the COMMAREA option of the RETURN command specifies the name of a communication area in which data is passed to the transaction identified in the TRANSID option. See the description of COMMAREA on page 296. (The TRANSID option specifies a transaction that will be initiated when input is received from the terminal associated with the task.) See the description of TRANSID on page 296. The length of the communication area is specified in the LENGTH option; PL/I programs need not specify the length.

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 an assembler language program, the data area should be a DSECT. The register used to address this DSECT must be loaded from DFHEICAP, which is in the DFHEISTG DSECT.

In a COBOL program, if a program passes a COMMAREA as part of a LINK, XCTL, or RETURN command, the data area can be in either working storage or the linkage section. A program receiving a COMMAREA should have the data specified in the linkage section. This applies whether the program is the receiving program during a LINK or XCTL command where a COMMAREA is passed, or the initial program where a COMMAREA and TRANSID have been specified on the RETURN command of a previously called task.

In a PL/I program, the data area can have any name, but it must be declared as a based variable, based on the parameter passed to the program. The pointer to this based variable should be declared explicitly as a pointer rather than contextually by its appearance in the declaration for the area. This will prevent the generation of a PL/I error message. No ALLOCATE statement can be executed within the receiving program for any variable based on this pointer. This pointer must not be updated by the application program.

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. It must not be longer than the length of the communication area being passed, because the results in this situation are unpredictable.

The invoked program can determine the length of any communication area that has been passed to it by accessing the EIBCALEN field in the EIB of the task. If no communication area has been passed, the value of EIBCALEN will be zero; otherwise, EIBCALEN will always contain the value specified in the LENGTH option of the LINK, XCTL, or RETURN command, regardless of the size of the data area in the invoked program.

When a communication area is passed by means of a 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 by means of an 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, if program A issues a LINK command to program B which, in turn, issues an 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.

A communication area can be passed by means of a RETURN command issued at the highest logical level when control returns to CICS; in this case, a copy of the communication area is made, and addressability to the copy is passed to the first program of the next transaction.

The invoked program can access field EIBFN in the EIB to determine which type of command invoked the program. The field must be tested before CICS commands are issued. If a LINK or XCTL invoked the program, the appropriate code will be found in the field; if RETURN is used, no CICS commands will have been issued in the task, and the field will contain zeros.

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); see "Chapter 1.6. Access to system information" on page 51 for details. Another way is to store the data in temporary storage; see "Chapter 4.7. Temporary storage control" on page 305 for details.

MVS/XA mixed addressing mode transactions:

Transactions that use either of the two types of addressing mode (AMODE=24 or AMODE=31) can be mixed only on MVS/XA, which supports the 31-bit addressing mode.

CICS supports the use of the LINK, XCTL, and RETURN commands between programs with different addressing modes and between programs with the same addressing mode.

The following restrictions apply to programs passing data by means of a communication area named by the COMMAREA option.

- Addresses passed within a communication area to an AMODE=31 program must be 31 bits long. Do not use 3-byte addresses with flag data packed into the top byte, unless the called program is specifically designed to ignore the top byte.
- Addresses passed as data to an AMODE=24 program must be below the 16-megabyte line if they are to be interpreted correctly by the called program.

These restrictions apply to the address of the communication area itself, as well as to addresses within it. However, a communication area above the 16-megabyte line can be passed to an AMODE=24 subprogram. CICS copies the communication area into an area below the 16-megabyte line for processing. It copies it back again when control returns to the linking program.

CICS does not validate any data addresses passed within a communication area between programs with different addressing modes.

Examples of passing data

The following examples, in assembler language, COBOL, and PL/I, show how the LINK command causes data to be passed to the program being linked to; the XCTL command is coded in a similar way.

ASM example – LINK

```
DFHEISTG DSECT     Invoking program
COMREG   DS 0CL20
FIELD    DS CL3
         .
         .
PROG1    CSECT
         .
         .
         MVC FIELD,=C'ABC'
         EXEC CICS LINK
         PROGRAM('PROG2')
         COMMAREA(COMREG) LENGTH(3)
         .
         .
         END

COMREG   DSECT      Invoked program
FIELD    DS CL3
         .
         .
PROG2    CSECT
         .
         .
         L COMPTR,DFHEICAP
         USING COMREG,COMPTR
         CLC FIELD,=C'ABC'
         .
         .
         END
```

COBOL example – LINK

```
                   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' ....
```

PL/I example – LINK

```
                   Invoking program
PROG1: PROC OPTIONS(MAIN);
DCL 1 COM_REGION AUTOMATIC,
      2 FIELD CHAR(3),
         .
         .
FIELD='ABC';
EXEC CICS LINK PROGRAM('PROG2')
     COMMAREA(COM_REGION) LENGTH(3);
END;


PROG2:              Invoked program
PROC(COMM_REG_PTR) OPTIONS(MAIN);
DCL COMM_REG_PTR PTR;
DCL 1 COM_REGION BASED(COMM_REG_
      PTR),
      2 FIELD CHAR(3),
         .
         .
IF EIBCALEN>0 THEN DO;
    IF FIELD='ABC' THEN ...
         .
         .
    END;
         .
         .
END;
```

The following examples, in assembler language, COBOL, and PL/I, show how the RETURN command is used to pass data to a new transaction.

ASM example — RETURN

```
DFHEISTG DSECT     Invoking program
TERMSTG  DS 0CL20
FIELD    DS CL3
DATAFLD  DS CL17
         .
         .
PROG1    CSECT
         .
         .
         MVC FIELD,=C'ABC'
         EXEC CICS RETURN
         TRANSID('TRN2')
         COMMAREA(TERMSTG)
         .
         .
         END

TERMSTG  DSECT      Invoked program
FIELD    DS CL3
DATAFLD  DS CL17
         .
         .
PROG2    CSECT
         .
         .
         CLC EIBCALEN,=H'0'
         BNH LABEL2
         L COMPTR,DFHEICAP
         USING TERMSTG,COMPTR
         CLC FIELD,=C'XYZ'
         BNE LABEL1
         MVC FIELD,=C'ABC'
LABEL1   DS 0H
         .
         .
LABEL2   DS 0H
         .
         .
         END
```

COBOL example — RETURN

```
                   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 'ABC' 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.
```

PL/I example — RETURN

```
                   Invoking program
PROG1: PROC OPTIONS(MAIN);
DCL 1 TERM_STORAGE,
      2 FIELD CHAR(3),
        .
        .
FIELD='XYZ';
EXEC CICS RETURN TRNID('TRN2')
     COMMAREA(TERM_STORAGE);
END;


PROG2:              Invoked program
PROC(TERM_STG_PTR) OPTIONS(MAIN);
DCL TERM_STG_PTR PTR;
DCL 1 TERM_STORAGE
       BASED(TERM_STG_PTR),
      2 FIELD CHAR(3),
        .
        .
IF EIBCALEN>0 THEN DO;
    IF FIELD='XYZ' THEN FIELD='ABC';
    END;
EXEC CICS RETURN;
END;
```

Program control options

COMMAREA(data-area)
specifies a communication area that is to be made available to the invoked program. For LINK commands, a pointer to the data area is passed; for XCTL commands, a pointer to the data area is passed or a copy of it (see "Passing data to other programs" on page 292) and for RETURN commands, because the data area is freed before the next program is invoked, a copy of the data area is created and a pointer to the copy is passed.

The communication area specified on a RETURN command will be passed to the next command level program that runs at the terminal. If the terminal is in TRANSCEIVE status, the next command level program is not guaranteed to be part of the transaction specified by TRANSID. It could be part of a transaction started by automatic task initiation (ATI). To ensure that the communication area is passed to the correct program, the terminal must not be in TRANSCEIVE status.

ENTRY(ptr-ref)
specifies the pointer reference that is to be set to the address of the entry point in the program, table, or map that has been loaded.

FLENGTH(data-area)
specifies a fullword binary area to be used with the LOAD command. On completion of the load operation, the data area is set to the length of the loaded program, table, or map. FLENGTH and LENGTH are mutually exclusive.

HOLD
specifies that the loaded program, table, or map is not to be deleted (if still resident) when the task issuing the LOAD command is terminated; deletion is to occur only in response to a RELEASE command, from this task or from another task.

LENGTH(parameter)
specifies a halfword binary value to be used with LINK, XCTL, RETURN, and LOAD commands.

For a LINK, XCTL, or RETURN command, the parameter must be a data value that is the length in bytes of the communication area. If a negative value is supplied, zero is assumed. The maximum length that you can specify is 32763 bytes. If you want to specify anything greater, use FLENGTH. The LENGTH operand is mutually exclusive with the FLENGTH operand, which can be coded only on a LOAD command.

For a LOAD command, the parameter must be a data area. On completion of the LOAD operation, the data area is set to the length of the loaded program, table, or map.

PROGRAM(name)
specifies the identifier of the program to which control is to be passed unconditionally (for a LINK or XCTL command); or the identifier of a program, table, or map to be loaded (for a LOAD command) or deleted (for a RELEASE command). The specified name can consist of up to eight alphanumeric characters and must have been defined as a program to CICS.

SET(ptr-ref)
specifies the pointer reference that is to be set to the address at which a program, table, or map is loaded.

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 consist of up to four characters and must have been defined as a transaction to CICS.

If the terminal is in TRANSCEIVE status, 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 will assume that the transaction started by ATI performs the same function and will erase the memory of the "name" specified in the TRANSID option.

Program control exceptional conditions

INVREQ

occurs if any of the following situations exists:

- A RETURN command with the COMMAREA option is issued in a program that is not at the highest logical level.
- A RETURN command with the COMMAREA option is issued but the supplied address is zero.
- A RETURN command with the TRANSID option is issued in a task that is not associated with a terminal.

LENGERR

occurs if the length specified 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.

Default action: terminate the task abnormally.

NOTAUTH

occurs when a resource security check has failed. The reasons for the failure are the same as for abend code AEY7, as described in the *CICS/MVS Messages and Codes* manual.

Default action: terminate the task abnormally.

PGMIDERR

occurs if a program, table, or map cannot be found in the PPT, in the library, or is disabled. PGMIDERR also occurs on MVS/XA if an application program executing in 24-bit mode issues a LOAD command for a program residing above the 16-megabyte line.

Default action: terminate the task abnormally.

Chapter 4.5. Storage control

The CICS storage control program controls requests for main storage to provide intermediate work areas and any other main storage not provided automatically by CICS but needed to process a transaction. You can initialize the acquired main storage to any bit configuration; for example, zeros or EBCDIC blanks.

Storage control commands are provided to:

- Get and initialize main storage (GETMAIN).
- Release main storage (FREEMAIN).

CICS releases all main storage associated with a task when the task is terminated normally or abnormally. This includes any storage acquired, and not subsequently released, by your application program.

If there is insufficient main storage to satisfy a GETMAIN command, the NOSTG exceptional condition will occur.

Exceptional conditions that occur during execution of a storage control command are handled as described in "Chapter 1.5. Exceptional conditions" on page 43.

Obtain and initialize main storage (GETMAIN)

This command is used to obtain a specified amount of main storage and, optionally, to initialize that storage to a specified bit configuration.

The storage obtained cannot be used as a terminal input/output area (TIOA) for subsequent terminal control input/output operations. Storage violations may occur if this is contravened.

The pointer reference specified in the SET option is set to the address of the acquired storage. The acquired storage is doubleword-aligned.

```
GETMAIN
SET(ptr-ref)
{LENGTH(data-value)|
  FLENGTH(data-value)}
[INITIMG(data-value)]
[NOSUSPEND]
[SHARED]

Conditions: LENGERR, NOSTG
```

Storage should be released when no longer needed; it will then be available to other tasks. Storage not released by the task is released by CICS when the task is terminated, unless you code the SHARED option in the GETMAIN command.

The LENGTH and FLENGTH options are mutually exclusive. On MVS/XA, if an application program executing in 31-bit mode uses the FLENGTH option, the storage may be allocated from above the 16-megabyte line. If an application program is executing in 24-bit mode or if the LENGTH option is used, the storage will always be allocated from below the 16-megabyte line.

The following example shows how to obtain a 1024-byte area of main storage. Declare BLANK in your program as a character representing a space.

```
EXEC CICS GETMAIN SET(PTR)
     LENGTH(1024) INITIMG(BLANK)
```

Release main storage (FREEMAIN)

```
FREEMAIN
DATA(data area)
```

This command is used to release main storage previously acquired by a GETMAIN command. If the task itself does not release the acquired storage, it is released by CICS when the task is terminated. The following example shows how to release main storage:

```
EXEC CICS FREEMAIN DATA(RECORD)
```

Some examples of GETMAIN and FREEMAIN are as follows:

Assembler

```
AREA     DS   CL100
:
EXEC CICS GETMAIN SET(9) LENGTH(100)
USING AREA,9
EXEC CICS FREEMAIN DATA(AREA)
```

PL/I

```
DCL AREA_PTR     POINTER,
    AREA         CHAR(100 BASED(ARERA_PTR);
:
EXEC CICS GETMAIN SET(AREA_PTR) LENGTH(100);
.
EXEC CICS FREEMAIN(AREA);
```

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).
SET ADDRESS OF AREA TO AREA-POINTER.
⋮
EXEC CICS FREEMAIN(AREA).
```

Storage control options

DATA(data area)
releases the main storage located at the data area.

This storage must have been acquired by a previous GETMAIN command (except in the case of BMS pages – see page 190).

You specify the *data area* (that is, the address of the storage) that was acquired by GETMAIN, not the *pointer reference* that was set to that address (see "Chapter 1.2. Command format and argument values" on page 5 for a description of argument values). So, in assembler, it must be a relocatable expression that is a data reference; in COBOL, it must be a data name; and in PL/I, it must be a data reference.

The length of storage released will be the length obtained by the GETMAIN and not necessarily the length of the data area.

FLENGTH(data-value)
specifies the length of main storage required as a fullword binary value. The maximum length that you can specify is 65,504 bytes, except for an application program executing in 31-bit mode on MVS/XA when the maximum length is 1,073,741,824 bytes (that is, 1 gigabyte).

FLENGTH and LENGTH are mutually exclusive.

The following table shows which requests are eligible to be satisfied by storage from above the 16-megabyte line (ANY), which requests are forced below the line (BELOW), and which result in a length error (LENGERR). Requests that are described as ANY will be satisfied by storage below the line if MVS decides that there is insufficient storage above.

FLENGTH value	24-bit mode	31-bit mode
1 - 4095	BELOW	BELOW
4096 - 65504	BELOW	ANY
65505 - 1024M	LENGERR	ANY
>1024M	LENGERR	LENGERR

INITIMG(data-value)
specifies the one-byte hexadecimal initialization value for the acquired main storage. In COBOL programs, a data value cannot be used. Instead the option should name a one byte data area containing the value.

LENGTH(data-value)
specifies the length of main storage required as a halfword binary value. The maximum length that you can specify is 32767 bytes. If you want to specify anything greater, use FLENGTH.

LENGTH and FLENGTH are mutually exclusive.

NOSUSPEND
specifies that application program suspension for the NOSTG condition is to be inhibited. This condition will be handled as described on page 43.

SET(ptr-ref)
specifies the pointer reference to be set to the address of the acquired main storage. The pointer reference addresses the user data, and not the CICS control information that precedes the acquired main storage.

SHARED
specifies that CICS will *not* release the storage at the end of the task. This enables task-to-task communication. When storage acquired with this option is no longer needed, release it with a FREEMAIN command.

Storage control exceptional conditions

LENGERR
occurs if the value specified for FLENGTH exceeds the maximum length.
Default action: terminate the task abnormally.

NOSTG
occurs if the requested main storage cannot be obtained.
Default action: suspend task activity until the required main storage can be provided.

Chapter 4.6. 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 region, and extrapartition if the data is directed to a destination that is external to the CICS region. The destinations must be defined in the destination control table (the DCT) by the system programmer when the CICS system is generated.

Transient data control commands are provided to:

- Write data to a transient data queue (WRITEQ TD).
- Read data from a transient data queue (READQ TD).
- Delete an intrapartition transient data queue (DELETEQ TD).

If TD is omitted, the command is assumed to be for temporary storage (see "Chapter 4.7. Temporary storage control" on page 305).

Exceptional conditions that occur during execution of a transient data control command are handled as described in "Chapter 1.5. Exceptional conditions" on page 43.

Intrapartition destinations: Intrapartition destinations are queues of 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 called intrapartition data; it must consist of variable-length records. Intrapartition destinations can be associated with either a terminal or an output data set. Intrapartition data may ultimately be transmitted upon request to the destination terminal or retrieved sequentially from the output data set.

Typical uses of intrapartition data include message switching, broadcasting, database access, and 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), and data collection (for example, for batched input from 2780 Data Transmission Terminals).

The storage associated with an intrapartition queue can be reused. The system programmer can specify, for each symbolic destination, whether or not storage tracks are to be reused as the data on them is read. If the storage is specified to be nonreusable, an intrapartition queue continues to grow, irrespective of whether the data has been read, until a DELETEQ TD command is issued when the whole of an intrapartition queue is deleted and the storage associated with it is released.

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

Data directed to or from an external destination is called extrapartition data and consists of sequential records that are fixed-length or variable-length, blocked or unblocked. The record format for an extrapartition destination must be defined in the DCT by the system programmer. (See the *CICS/MVS Resource Definition (Macro)* manual for details.)

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/MVS Resource Definition (Macro)* manual.

Automatic task initiation (ATI): For intrapartition destinations, CICS provides the option of automatic task initiation. A basis for automatic task initiation is established by the system programmer by specifying a nonzero trigger level for a particular intrapartition destination in the DCT. (See the description of the DFHDCT TYPE=INTRA macro in the *CICS/MVS Resource Definition (Macro)* manual.) When the number of entries (created by 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 READQ TD commands to deplete the queue.

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

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

Write data to transient data queue (WRITEQ TD)

```
WRITEQ TD
QUEUE(name)
FROM(data-area)
[LENGTH(data-value)]
[SYSID(name)]

Conditions: DISABLED, IOERR,
ISCINVREQ, LENGERR, NOSPACE,
NOTAUTH, NOTOPEN, QIDERR, SYSIDERR
```

This command is used to write transient data to a predefined symbolic destination. The destination (queue) is identified in the QUEUE option.

The FROM option specifies the data to be written to the queue, and the LENGTH option specifies the record length. The LENGTH option need not be specified for extrapartition queues of fixed-length records if the length is known and a data area of the correct size is available. If SYSID is specified, LENGTH must be specified as well.

The following example shows how to write data to a predefined symbolic destination; in this case, the control system message log (CSML):

```
EXEC CICS WRITEQ TD
     QUEUE('CSML')
     FROM(MESSAGE)
     LENGTH(LENG)
```

Read data from transient data queue (READQ TD)

```
READQ TD
QUEUE(name)
{INTO(data-area)|SET(ptr-ref)}
[LENGTH(data-area)]
[SYSID(name)]
[NOSUSPEND]

Conditions: DISABLED, IOERR,
ISCINVREQ, LENGERR, NOTAUTH,
NOTOPEN, QBUSY, QIDERR, QZERO,
SYSIDERR
```

This command is used to read transient data from a predefined symbolic source. The source (queue) is identified in the QUEUE option.

Reading a record from an intrapartition transient data queue defined as reusable is destructive; it can only be read once.

The INTO option specifies the area into which the data is to be placed. The LENGTH option must specify a data area that contains the maximum length of record that the program will accept. If the record exceeds this value, it is truncated and the LENGERR condition occurs. After the retrieval operation, the data area specified in the LENGTH option is set to the record length (before any truncation occurred). The LENGTH option need not be specified for extrapartition queues of fixed-length records if the length is known and a data area of the correct size is available. If SYSID is specified, LENGTH must be specified as well.

Alternatively, a pointer reference can be specified in the SET option. CICS then 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. After the retrieval operation, the data area specified in the LENGTH option is set to the record length.

If automatic task initiation is being used (see "Automatic task initiation (ATI)" on page 301), the HANDLE CONDITION QZERO command should be included to ensure that termination of an automatically initiated task only occurs when the queue is empty.

The following example shows how to read a record from an intrapartition data set (queue), which in this case is the control system message log (CSML), into a data area specified in the request:

```
EXEC CICS READQ TD
     QUEUE('CSML')
     INTO(DATA)
     LENGTH(LENG)
```

The following example shows how to read a record from an extrapartition data set (queue) having fixed-length records 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. It is assumed that the record length is known.

```
EXEC CICS READQ TD
     QUEUE(EX1)
     SET(PREF)
```

Delete an intrapartition transient data queue (DELETEQ TD)

```
DELETEQ TD
QUEUE(name)
[SYSID(name)]

Conditions: DISABLED, ISCINVREQ,
NOTAUTH, QIDERR, SYSIDERR
```

This command is used to delete all of the transient data associated with a particular intrapartition destination (queue). All storage associated with the destination is released (deallocated).

This command must be used to release the storage associated with a destination specified as nonreusable in the DCT. Otherwise, the storage remains allocated to the destination; the data and the amount of storage associated with the destination continue to grow whenever a WRITEQ TD command refers to the destination.

Transient data control options

FROM(data-area)
specifies the data that is to be written to the transient data queue.

INTO(data-area)
specifies the user data area into which the data read from the transient data queue is to be placed. If this option is specified, move-mode access is implied.

LENGTH(parameter)
specifies a halfword binary value to be used with WRITEQ TD and READQ TD commands.

For a WRITEQ TD command, the parameter must be a data value that is the length of the data that is to be written.

For a READQ TD command with the INTO option, the parameter 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.

For a READQ TD command with the SET option, the parameter must be a data area. On completion of the retrieval operation, the data area is set to the length of the data.

NOSUSPEND
specifies that application program suspension for the QBUSY condition is to be inhibited. This condition will be handled as described on page 43.

QUEUE(name)
specifies the symbolic name of the queue to be written to, read from, or deleted. The name must be alphanumeric, up to four characters in length, and must have been defined in the DCT by the system programmer.

When used with the READQ TD command, the name used should not be that of the system spool file otherwise unpredictable results or an abnormal termination will occur.

If SYSID is specified, the data set is assumed to be on a remote system irrespective of whether or not the name is defined in the DCT. Otherwise the entry in the DCT will be used to determine if the data set is on a local or remote system.

SET(ptr-ref)
specifies a pointer reference that is to be set to the address of the data read from the queue. If this option is specified, locate-mode access is implied.

SYSID(name) remote systems only
specifies the name of the system whose resources are to be used for intercommunication facilities. The name can be up to four characters in length.

Transient data control exceptional conditions

DISABLED
occurs if a transient data command is function shipped to a CICS/ESA* system and the queue has been disabled on that system.

Default action: terminate the task abnormally.

IOERR
occurs when an input/output error occurs and the data record in error is skipped. Also occurs for an extrapartition destination if the data length does not

* IBM Trademark. For a list of trademarks see page iii.

match the size specified in the RECSIZE operand of the DFHDCT TYPE=SDSCI system macro.

Also occurs for an intrapartition destination if the data length exceeds the maximum permissible length for an intrapartition data set.

This condition occurs so long as the queue can be read; a QZERO condition occurs when the queue cannot be read, in which case a restart may be attempted.

Default action: terminate the task abnormally.

ISCINVREQ
occurs when the remote system indicates a failure which does not correspond to a known condition.

Default action: terminate the task abnormally.

LENGERR
occurs in any of the following situations:

- The LENGTH option is not coded for an input (without the SET option) or output operation involving variable-length records.
- The length specified on output is greater than the maximum record size specified for the queue in the DCT.
- The record read from a queue is longer than the length specified for the input area; the record is truncated and the data area supplied in the LENGTH option is set to the actual record size.
- An incorrect length is specified for an input or output operation that involves fixed-length records.
- The LENGTH option is not coded for an input operation (without the SET option) from, or an output operation to, a destination other than disk, involving fixed-length records.

Default action: terminate the task abnormally.

NOSPACE
occurs if no more space exists on the intrapartition 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 when a resource security check has failed. Use of SYSID will always raise the NOTAUTH condition when resource security level checking is in effect (RSLC=YES in the PCT). The reasons for the failure are the same as for abend code AEY7, as described in the *CICS/MVS Messages and Codes* manual.

Default action: terminate the task abnormally.

NOTOPEN
occurs if the destination is closed.

Default action: terminate the task abnormally.

QBUSY
occurs if a READQ TD command attempts to access a record in an intrapartition queue that is being written to or is being deleted by another task. This condition applies only to input; output requests are always queued until the intrapartition queue is no longer busy.

Default action: the task issuing the READQ TD command waits until the queue is no longer being used for output.

However, the NOSUSPEND option on page 303 overrides this default action.

QIDERR
occurs if the symbolic destination to be used with a transient data control command cannot be found.

Default action: terminate the task abnormally.

QZERO
occurs when the destination (queue) accessed by a READQ TD command is empty, or the end of the transient data queue has been reached.

Default action: terminate the task abnormally.

SYSIDERR
occurs when the SYSID option specifies either a name which is not defined in the intersystem table, or a system to which the link is closed.

Default action: terminate the task abnormally.

Chapter 4.7. 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 control commands are provided to:

- Write data to a temporary storage queue (WRITEQ TS).
- Update data in a temporary storage queue (WRITEQ TS REWRITE).
- Read data from a temporary storage queue (READQ TS).
- Delete a temporary storage queue (DELETEQ TS).

If TS is omitted, the command is assumed to be for temporary storage, not for transient data which has similar commands.

Exceptional conditions that occur during execution of a temporary storage control command are handled as described in "Chapter 1.5. Exceptional conditions" on page 43.

Temporary storage queues: Temporary storage queues are identified by symbolic names of up to eight characters assigned by the originating task. Temporary data can be retrieved by the originating task or by any other task using the symbolic name assigned to it. Specific items (logical records) within a queue are referred to by relative position numbers. 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.

Temporary storage queues remain intact until they are deleted by the originating task or by any other task; prior to deletion, they can be accessed any number of times. Even after the originating task is terminated, temporary data 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. Generally, main storage should be used if the data is needed for short periods of time; auxiliary storage should be used if the data is to be kept for long periods of time. Data stored in auxiliary storage is retained after CICS termination and can be recovered in a subsequent restart, but data in main storage cannot be recovered. 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).

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 READQ TS command with the ITEM(1) option; failure to do 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 provides facilities for efficient handling of sequential data sets.

Some uses of temporary storage queues follow:

- Terminal paging. A task could retrieve a large master record from a direct-access data set, 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 data set. Assume 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.

Write data to a temporary storage queue (WRITEQ TS)

```
WRITEQ TS
QUEUE(name)
FROM(data-area)
[LENGTH(data-value)]
[ITEM(data-area)[REWRITE]]
[SYSID(name)]
[MAIN|AUXILIARY]
[NOSUSPEND]

Conditions: INVREQ, IOERR,
ISCINVREQ, ITEMERR, LENGERR,
NOSPACE, NOTAUTH, QIDERR, SYSIDERR
```

This command is used to store temporary data (records) in a temporary storage queue in main or auxiliary storage.

If you use the MAIN option on this command to write data to a temporary storage queue on a remote system, the data is stored in main storage provided that the remote system is accessed by the CICS multiregion operation (MRO) facility and that the remote system is at the same release level as the requesting system. If these conditions are not met, the data is stored in auxiliary storage.

The queue is identified in the QUEUE option. The FROM and LENGTH options are used to specify the record that is to be written to the queue, and its length.

If the ITEM option is specified, CICS assigns an item number to the record in the queue, and sets the data area supplied in that option to the item number. If the record starts a new queue, the item number assigned is 1; subsequent item numbers follow on sequentially.

The REWRITE option specifies that records are to be updated, in which case the ITEM option must also be specified to identify the item (record) that is to be replaced by the data identified in the FROM option. If the specified queue exists, but the specified item cannot be found, the ITEMERR condition occurs. If the specified queue does not exist, the QIDERR condition occurs.

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)
     ITEM(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
```

Read data from temporary storage queue (READQ TS)

```
READQ TS
QUEUE(name)
{INTO(data-area)|SET(ptr-ref)}
[LENGTH(data-area)]
[NUMITEMS(data-area)]
[ITEM(data-area)|NEXT]
[SYSID(name)]

Conditions: INVREQ, IOERR,
ISCINVREQ, ITEMERR, LENGERR,
NOTAUTH, QIDERR, SYSIDERR
```

This command is used to retrieve data from a temporary storage queue in main or auxiliary storage. The queue is identified in the QUEUE option.

The INTO option specifies the area into which the data is to be placed. The LENGTH option must specify a data area that contains the maximum length of record that the program will accept. If the record length exceeds the specified maximum length, it is truncated and the LENGERR condition occurs. After the retrieval operation, the data area specified in the LENGTH option is set to the record length (before any truncation occurred).

Alternatively, a pointer reference can be specified in the SET option. CICS then 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 is executed. After the retrieval operation, the data area specified in the LENGTH option is set to the record length.

The ITEM and NEXT options are used to specify which record (item) within a queue is to be read. If the ITEM option is specified, the record with the specified item number is retrieved. If the NEXT option is in effect (either explicitly or by default), the next record after the last record to be retrieved (**by any task**) is retrieved. Therefore, if different tasks are to access the same queue and each task is to start at the beginning of the queue, the ITEM option must be used.

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)
```

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
```

Delete temporary storage queue (DELETEQ TS)

```
DELETEQ TS
QUEUE(name)
[SYSID(name)]

Conditions: ISCINVREQ, NOTAUTH,
QIDERR, SYSIDERR
```

This command is used to delete all the temporary data associated with a temporary storage queue. All storage associated with the queue is freed.

Temporary data should be deleted at the earliest possible time to avoid using excessive amounts of storage.

Temporary storage control options

AUXILIARY
specifies that the temporary storage queue is on a direct-access storage device in auxiliary storage.

FROM(data-area)
specifies the data that is to be written to temporary storage.

INTO(data-area)
specifies the data area into which the data is to be written. The data area may be any variable, array, or structure. If this option is specified, move-mode access is implied.

ITEM(parameter)
specifies a halfword binary value to be used with WRITEQ TS and READQ TS commands.

When used with a WRITEQ TS command in which the REWRITE option is not specified, 'parameter' must be a data area that is to be set to the item (record) number assigned to this record in the queue. If the REWRITE option is specified, the data area specifies the item in the queue that is to be replaced.

When used with a READQ TS command, 'parameter' specifies the item number of the logical record to be retrieved from the queue. The parameter must be a data value that is to be taken as the relative number of the logical record to be retrieved. This number may be the number of any item that has been written to the temporary storage queue.

LENGTH(parameter)
specifies the length (as a halfword binary value) of the data to be used with WRITEQ TS and READQ TS commands.

For a WRITEQ TS command, the parameter must be a data value that is the length of the data that is to be written.

For a READQ TS command with the INTO option, the parameter 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.

For a READQ TS command with the SET option, the parameter must be a data area. On completion of the retrieval operation, the data area is set to the length of the data.

MAIN
specifies that the temporary storage queue is in main storage.

NEXT
specifies that the next sequential logical record following the last record to be retrieved (by any task) is to be retrieved.

NOSUSPEND
specifies that application program suspension for the NOSPACE condition is to be inhibited. This condition will be handled as described on page 43.

NUMITEMS
specifies a halfword binary field into which CICS stores a number indicating how many items there are in the queue.

QUEUE(name)
specifies the symbolic name of the queue to be written to, read from, or deleted. 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, up to eight characters in

length, and unique within the CICS system, and must not be solely binary zeros. Do not use the characters X'DF' as a queue name prefix; these characters are reserved for CICS use. Also, using any of the following in queue name prefixes can have unpredictable consequences:

- Embedded blanks
- Hexadecimal X'AA' through X'AF'
- Hexadecimal X'BA' through X'BF'
- Hexadecimal X'CA' through X'CF'
- Hexadecimal X'DA' through X'DF'
- Hexadecimal X'EA' through X'EF'
- Hexadecimal X'FA' through X'FF'

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 but the data is not stored.

SET(ptr-ref)
specifies the pointer reference that is to be set to the address of the retrieved data. If this option is specified, locate-mode access is implied.

SYSID(name) (remote systems only)
specifies the name of the system whose resources are to be used for intercommunication facilities. The name can be up to four characters in length.

Temporary storage control exceptional conditions

INVREQ
occurs when a WRITEQ TS command specifies a queue:

- That is locked and awaiting ISC session recovery (see the *CICS/MVS Installation Guide* for details.), **or**
- With a symbolic name that is all binary zeros.

This condition also occurs for a READQ TS or DELETEQ TS command when the record to be retrieved has been created by a DFHTS TYPE=PUT macro.

Default action: terminate the task abnormally.

IOERR
occurs when there is an unrecoverable input/output error.

Default action: terminate the task abnormally.

ISCINVREQ
occurs when the remote system indicates a failure which does not correspond to a known condition.

Default action: terminate the task abnormally.

ITEMERR
occurs when the item number specified or implied by a READQ TS command, or a WRITEQ TS command with the REWRITE option, is invalid (that is, outside the range of entry numbers assigned for the queue).

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 for move-mode input operations, **or**
- the length of the stored data specified by the WRITEQ TS command is zero or negative.

Default action: terminate the task abnormally.

NOSPACE
occurs when insufficient space is available in the temporary storage data set 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 when a resource security check has failed. Use of SYSID will always raise the NOTAUTH condition when resource security level checking is in effect (RSLC=YES in the PCT). The reasons for the failure are the same as for abend code AEY7, as described in the *CICS/MVS Messages and Codes* manual.

Default action: terminate the task abnormally.

QIDERR
occurs when the queue specified by a READQ TS command, or by a WRITEQ TS command with the REWRITE option cannot be found, either in main storage or in auxiliary storage.

Default action: terminate the task abnormally.

SYSIDERR
occurs when the SYSID option specifies either a name which is not defined in the intersystem table, or a system to which the link is closed.

Default action: terminate the task abnormally.

Part 5. Recovery and debugging

Chapter 5.1. Introduction to recovery and debugging 311

Chapter 5.2. Abnormal termination recovery 313

Chapter 5.3. Trace control 317

Chapter 5.4. Dump control 323

Chapter 5.5. Journal control 327

Chapter 5.6. Recovery (sync points) 331

Chapter 5.1. Introduction to recovery and debugging

CICS application programs are executed in an interactive environment. As a result, the operating system, CICS itself, and the application programs must be responsive to many factors. Because the network on which the CICS system is based consists of a variety of terminals and subsystems from which requests for services are received at random, the relationships between application programs and data set activity differ from one moment to the next.

CICS provides the following aids to the testing, monitoring, and debugging of application programs:

- Execution (Command Level) Diagnostic Facility (EDF). Allows commands to be displayed in source form on a screen, both before and after execution so that they can be checked and altered if necessary. This facility is described in "Chapter 1.7. Execution (command level) diagnostic facility" on page 57.
- Sequential terminal support. Enables sequential devices, such as card readers and disk units, to simulate online interactive terminals or subsystems of a CICS network so that early testing can be carried out.
- Abnormal termination recovery. The HANDLE ABEND command can be used to deal with abnormal termination conditions, and the ABEND command can be used to cause a task to be terminated abnormally.
- Trace facility. A trace table containing entries that reflect the execution of various CICS commands, and entries generated by application programs, can be written to main storage and, optionally, to an auxiliary storage device.
- Dump facility. Specified areas of main storage can be dumped onto a sequential data set, either tape or disk, for subsequent off-line formatting and printing using a CICS utility program.
- Journals. Facilities are provided for creating entries in "journals". A journal is a set of special-purpose sequential data sets, which are used for statistical or monitoring purposes; for example, the system log is a journal.
- Recovery. When a task is abnormally terminated, CICS can restore certain resources to their original state so that a transaction can be resubmitted for restart with no further action by the operator. The SYNCPOINT command can be used to subdivide a program so that only the uncompleted part of a transaction need be resubmitted.

Sequential terminal support, for which no special CICS commands are required, is described below. The other facilities, and the commands that enable the application programmer to make use of them, are discussed in the other chapters of this part.

Sequential terminal support

Even at the simplest level of program testing, the programmer should take the following into consideration. It is inefficient and error-prone to test a program from a terminal if all test data must be keyed into the system from that terminal for each test case. The programmer cannot easily retain a backlog of proven test data and quickly test programs through the key-driven terminal as changes are made.

CICS allows the application programmer to test a program without the use of a telecommunication device. It is possible for the system programmer to specify through the terminal control table (TCT) that sequential devices be used as terminals. These sequential devices may be card readers, line printers, disk units, or magnetic tape units. In fact, the terminal control table can include combinations of sequential devices such as: card reader and line printer (CRLP), one or more disk or tape data sets as input, one or more disk or tape data sets as output. A TCT that contains references to these sequential terminals can also define other true telecommunications terminals in the system.

The input data submitted from a sequential device must be prepared in the form in which it would come from a telecommunication device. The input data must start with a transaction identification code of up to four characters, unless the transaction identification is predefined in the TCT. If there is more data, and the transaction identification code has less than four characters, a system-defined transaction code delimiter or a blank must precede the extra data. If a sequential device is being used as a terminal, an end-of-data indicator (a 0-2-8 punched card code (X'E0') or the equivalent as specified when the CICS system is generated) must follow the input message or the system-defined data termination character. The input is processed sequentially and must be unblocked. The sequential access method (SAM) is used to read and write the necessary inputs and outputs. The operating system utilities can be used to create the input data sets and print the output data sets.

Using this approach, it is possible to prepare a stream of transaction test cases to do the basic testing of a program module. As the testing progresses, the user can generate additional transaction streams to validate the multiprogramming capabilities of the programs or to allow transaction test cases to be run concurrently.

For operational convenience, it is usually appropriate to place a terminating transaction at the end of each input stream. For tests that use a single input stream, the transaction can be CEMT SHUTDOWN with appropriate responses following the initial message to respond to the CEMT queries about the mode of shutdown. In a batch-only testing environment, this enables CICS to be terminated in an orderly manner without operator intervention.

Where more than one sequential input stream is used, only one should include the CEMT PERFORM SHUT transaction. Others can be terminated with CSSF GOODNIGHT.

At some point in testing, it is necessary to use telecommunication devices to ensure that the transaction formats are satisfactory, that the terminal operational approach is satisfactory, and that the transactions can be processed on the terminal. The terminal control table can be altered to contain more and different devices as the testing requirements change.

When testing has proved that transactions can be processed concurrently and the necessary data sets (actual or duplicate) for online operation have been created, the user begins testing in a controlled environment with the telecommunication devices. In this controlled environment, the transaction test cases should represent all functions of the eventual system, but on a smaller, measurable scale. For example, a company whose information system will work with 15 district offices may select one district office for the controlled test. During the controlled test, all transactions, data set activity, and output activity from the system should be monitored closely.

Requests for input or output from a sequential terminal are expressed by means of terminal control commands in the normal way. In response to a RECEIVE command, where the terminal has been described in the terminal control table as a CRLP, DISK, or TAPE terminal, data is read from the input data set until any one of the following situations occurs:

- An end-of-data indicator is detected in the input stream. (The indicator must be defined by the user when the CICS system is generated.)
- Sufficient input has been read to fill the input area associated with the line used for transmission. If an end-of-data indicator is not detected before the input area is filled, all further data preceding an end-of-data indicator is bypassed and treated as a system error, which is passed to the user-installation terminal error program (DFHTEP).
- End-of-file (EOF) is detected. The input operation is considered complete. Any subsequent RECEIVE command is treated as a system error, which is passed to the user-installation terminal error program (DFHTEP) with a response code of 4.

In response to a SEND command for a CRLP terminal, lines are written in print format as follows:

- If there is no new-line (X'15') character within the number of characters contained in one print line of the specified line size (as defined by the system programmer in the LPLEN option of the DFHTCT TYPE=TERMINAL macro), the output is written in fixed-length lines of the size specified.
- If new-line characters are encountered, a new line is begun for each one.

Writing of output continues until the end of the user data is reached. For more information about terminal control commands, see "Chapter 3.3. Terminal control" on page 221.

Chapter 5.2. Abnormal termination recovery

During abnormal termination of a task, a program-level abend exit facility is provided in CICS so that you can include an exit routine of your own that can be executed when required. An example of a function performed by such a routine is the "cleanup" of a program that has started but not completed normally.

The **HANDLE ABEND** command activates or reactivates an abend exit within your application program; you can also use this command to cancel a previously activated exit.

The **ABEND** command terminates a task abnormally, and causes an active exit routine to be executed; you can also use this command to request a dump.

A 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 exit, but only one exit at each logical level can be active. (Logical levels are explained in "Chapter 4.4. Program control" on page 289.)

When a task is abnormally terminated, CICS searches for an active 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 exit found, if any, is given control. This procedure is shown in Figure 30 on page 314, which also shows how subsequent abend exit processing is determined by the user-written exit routine.

To prevent recursive abends in an exit routine, CICS deactivates an exit upon entry to the exit routine. If a retry of the operation is attempted, the application programmer can branch to a point in the program that was in control at the time of the abend and issue a HANDLE ABEND RESET command to reactivate the exit. This command can also be used to reactivate an exit (at the logical level of the issuing program) that was canceled previously as described above.

See the section dealing with creating a program abend exit in the *CICS/MVS Customization Guide* for additional information about exit routines, and the *CICS/MVS Messages and Codes* manual for a list of the transaction abend codes generated for abnormal terminations initiated by CICS.

Handle an abnormal termination exit (HANDLE ABEND)

```
HANDLE ABEND
[PROGRAM(name)|
  LABEL(label)|CANCEL|RESET]

Condition: PGMIDERR (PROGRAM only)
```

This command is used to activate, cancel, or reactivate an exit for abnormal termination processing. You can suspend the command by means of the PUSH and POP commands as described in "Chapter 1.5. Exceptional conditions" on page 43.

When activating an exit, you must use the PROGRAM option to specify the name of a program to receive control, or (except for PL/I programs) the LABEL option to specify a program label to which control will branch, when an abnormal termination condition occurs. A HANDLE ABEND PROGRAM or HANDLE ABEND LABEL command overrides any previous such request in any application program at the same logical level.

A HANDLE ABEND is not effective while a transaction is being processed by a task-related user exit interface module (DFHERM) – for example, a EXEC DLI command or a DB2 transaction. An abend in this case will cause a dump to be taken. If you do not want this to happen, consult your system programmer about adding a global user exit that can suppress the dump.

If intersystem communication 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 will not be returned to CICS to clean up the control blocks. Results will be unpredictable if the command is retried.

A HANDLE ABEND command with the CANCEL option will cancel a previously established exit at the logical level of the application program in control.

A HANDLE ABEND command with the RESET option will reactivate an abnormal termination exit that was canceled by a HANDLE ABEND CANCEL command or by CICS. This command would usually be issued in an abnormal termination exit routine.

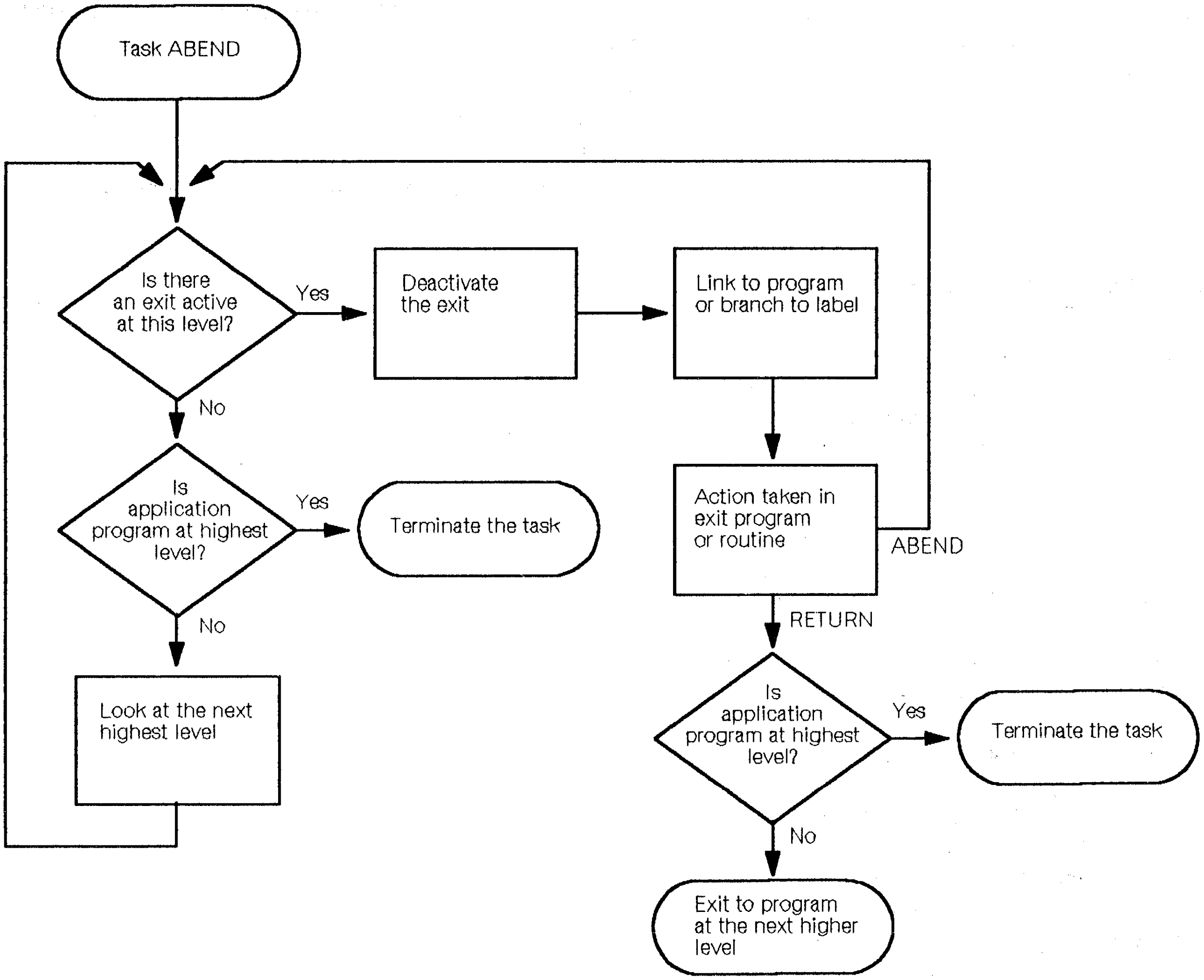

Task ABEND
Is there an exit active at this level?
Yes
Deactivate the exit
Link to program or branch to label
No
Is application program at highest level?
Yes
Terminate the task
Action taken in exit program or routine
ABEND
No
RETURN
Look at the next highest level
Is application program at highest level?
Yes
Terminate the task
No
Exit to program at the next higher level

Figure 30. ABEND exit processing

When the label specified in a HANDLE ABEND LABEL command receives control, the registers are set as follows:

```
ASM:    R15   - Abend label.
        R0-14 - Contents at the time
        of last CICS service request.

COBOL: Control returns to the HANDLE
        ABEND command with the
        registers restored; COBOL GO
        TO statement is then
        executed.
```

On MVS/XA, the addressing mode will be set to the addressing mode in which the HANDLE ABEND command has been issued.

The following example shows how to establish a program as an exit:

```
EXEC CICS HANDLE ABEND
     PROGRAM('EXITPGM')
```

Terminate task abnormally (ABEND)

```
ABEND
[ABCODE(name)]
[CANCEL]
```

This command is used to request that a task be terminated abnormally.

The main storage associated with the terminated task is released; optionally, a dump of this storage can be obtained first by using the ABCODE option to specify a four-character abnormal termination code, which CICS will place in the dump to identify it.

If the CANCEL option is specified, all abnormal termination exits, if any, established by HANDLE ABEND commands at any level in the task are canceled before the task is terminated. If the PL/I STAE execution-time option has been specified, an abnormal termination exit will have been established by PL/I. This exit is revoked by the CANCEL option. (See the *PL/I Optimizing Compiler Programmer's Guide* for further information.)

The following example shows how to terminate a task abnormally:

```
EXEC CICS ABEND ABCODE('BCDE')
```

Abnormal termination recovery options

ABCODE(name)
specifies that main storage related to the task that is being terminated is to be dumped and provides a name to identify the dump. The specified name may consist of up to four characters.

CANCEL
specifies that exits established by HANDLE ABEND and ABEND commands are to be canceled; in effect they are ignored. A HANDLE ABEND CANCEL command cancels a previously established exit at the logical level of the application program in control. An ABEND CANCEL command cancels all exits at any level in the task (and terminates the task abnormally).

LABEL(label)
specifies the program label to which control will branch if abnormal termination occurs. This option cannot be used for PL/I application programs.

PROGRAM(name)
specifies the name of the program to which control is to be passed if the task is terminated abnormally. The name can consist of up to eight alphanumeric characters and must have been defined in the processing program table (PPT).

RESET
specifies that an exit canceled by a HANDLE ABEND CANCEL command, or by CICS, is to be reactivated.

Abnormal termination recovery exceptional condition

PGMIDERR
occurs if a program cannot be found in the PPT or is disabled.

Default action: terminate the task abnormally.

Chapter 5.3. Trace control

The CICS trace control program is a debugging and monitoring aid for application programmers and IBM field engineers. This facility makes use of a trace table, which resides in main storage, and which consists of entries produced in response to trace control commands. CICS auxiliary trace allows you to write trace records on a sequential device for later analysis.

Using trace control commands, you can:

- Specify user trace entry points or user event monitoring points (ENTER)
- Switch the CICS trace facility on or off (TRACE ON and TRACE OFF).

Trace entry points

The points at which trace entries are produced during CICS operation are of two types: 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 only system trace entry points that need concern the command level application programmer are for the EXEC interface program. These produce entries in the trace table whenever a CICS command is executed. 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 the application program. Between them, these two trace entries allow the flow of control through an application program to be traced, and a check to be made on which exceptional conditions, if any, occurred during its execution. (The ABEND, RETURN, TRACE ON, TRACE OFF, and XCTL commands produce single entries only.)

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 ENTER command is executed. Each trace entry request, which can be given a unique identifier, causes eight bytes of data to be placed in the trace table.

Event monitoring points

A user event monitoring point (EMP) can be defined in an application program by means of the MONITOR option of the ENTER command. At a user EMP, information can be added to the user fields in accounting and performance class monitoring data records. The classes of data records to be eligible for the addition of user information are specified by the ACCOUNT and PERFORM options. The actual user information to be recorded is defined in the monitoring control table. The user information recorded, in conjunction with similar data recorded automatically by the system, can be used as input to offline analysis and reporting programs. More information on the use of monitoring is given in the *CICS/MVS Customization Guide*.

Trace facility control

The CICS trace facility is controlled by a number of trace flags; the flags are stored within CICS and the TRACE ON and TRACE OFF commands are used to turn them on or off.

There is a master system trace flag, which must be on before any system trace entries are produced, and a separate system flag for each type of system trace entry. The master system trace flag can be turned on or off independently of individual system trace flags; thus the system trace pattern of activity can be left intact but controlled as a single unit. When the master system trace flag and one or more system trace flags are on, the relevant system trace entries are produced for all active tasks, and tasks that become active subsequently, until the flags are turned off again.

The TRACE command can be used to control the system trace flags for other parts of CICS, should it be necessary to debug a program down to the level of the CICS macro instructions issued by the EXEC interface program; for further details, see “Control the CICS trace facility (TRACE ON, TRACE OFF)” on page 320.

There is a master user trace flag, and an individual user trace flag for each task. If the master user trace flag is on, requested user trace entries are produced for all active tasks, and tasks that become active subsequently, until the flag is turned off again. Each individual user trace flag controls user trace entries only for the task that turns the flag on or off.

The master terminal operator can turn the whole CICS trace facility on or off by entering suitable commands; all flags are turned on or off together when this method is used.

Trace table format

The CICS trace table is located in main storage; you can gain access to it by investigating a CICS dump. How to get a CICS dump is described in “Chapter 5.4. Dump control” on page 323.

The trace table consists of a trace header and a variable number of fixed-length entries produced by the trace control commands.

The format of the trace header is:

Trace header

Bytes	Contents
0-3	Address of last-used entry.
4-7	Address of first entry in the table.
8-11	Address of last entry in the table.
12-31	Reserved.

Each entry in the trace table is 32 bytes in length and is aligned on a 32-byte boundary. The trace table area is of a fixed size specified by the system programmer, and entries are placed in the table in a wraparound manner; that is, when the table is full, the next entry is placed at the head of the table, overwriting the original entry.

The general format of a trace table entry in storage is as follows. A different layout is used in the interpreted display of trace table entries in a CICS dump and in auxiliary trace output. See the *CICS/MVS Problem Determination Guide* for further information.

Trace entry - general format

Bytes	Contents
0	Trace identifier (binary).
1,2	TCA type of request (binary).
3,4	Reserved.
5-7	TCA identifier (packed dec).
8-11	Data field A.
12-15	Data field B.
16-23	Resource name (character).
24-27	Register 14.
28-31	Time stamp.

The formats of the EXEC interface program trace entry on issuance of a command, on completion of a command, and of a user trace, are shown in the following tables.

In these tables, the numbers in parentheses are the bit positions of the associated byte.

Trace entry on issuance of command

Bytes	Contents
0	X'E1' trace identifier.
1	Not used.
2(0-3)	X'0', identifying the first entry for the command.
2(4-7)	X'4', identifies entry as a system entry.
3,4	Not used.
5-7	User task sequence number (packed decimal).
8-11	ASM: address of the dynamic storage described by DFHEISTG DSECT. COBOL: address of the working storage section. PL/I: address of the dynamic storage area (DSA).
12,13	Not used.
14,15	Code identifying the CICS command. (See EIBFN in Appendix A.)
16-23	Not used.
24-27	Return point in application program.
28-31	Time stamp.

Trace entry on completion of command

Bytes	Contents
0	X'E1' trace identifier.
1	EIBGDI - "go to depending on index" field (COBOL only).
2(0-3)	X'F', identifying the second entry for the command.
2(4-7)	X'4', identifies entry as a system entry.
3,4	Not used.
5-7	User task sequence number (packed decimal).
8-13	Code identifying the condition raised during execution of the command. (See EIBRCODE in Appendix A.) Zero means no condition raised.
14,15	Code identifying the command (same as bytes 14 and 15 in the entry on issuance of the command, above).
16-23	Not used.
24-27	Return point in application program; if code in bytes 8-13 is nonzero, these bytes contain address of label given in HANDLE CONDITION command associated with response.
28-31	Time stamp.

User trace entry

Bytes	Contents
0	Trace identifier; the binary value specified in the TRACEID option of the ENTER command.
1	Not used.
2(0-3)	Not used.
2(4-7)	X'2', identifies entry as a user entry.
3,4	Not used.
5-7	User task sequence number (packed decimal).
8-15	Data field supplied in the FROM option of the ENTER command.
16-23	Name supplied in the RESOURCE option of the ENTER command.
24-27	Return point in application program.
28-31	Time stamp.

For system trace entries only, if consecutive, identical entries for the trace table are generated, the first entry has the form of a standard entry, but subsequent identical entries are replaced by a single special entry, immediately following the first entry. The trace identifier of this special entry (in byte 0) is X'FD'; bytes 24 through 27 contain a packed decimal number that shows how many repeated entries have been replaced by this single entry.

Trace table entries with the trace identifiers X'FE' or X'FF' indicate the turning on or turning off, respectively, of the trace facility. Details of these and other CICS trace entries are given in the *CICS/MVS Problem Determination Guide*.

CICS auxiliary trace facility

All trace entries that are written to the trace table will also be written to the auxiliary trace data sets, if the auxiliary trace facility is active. Auxiliary trace entries are recorded only when main storage trace is also active. Whereas the entries written to the trace table wrap around, the auxiliary trace data sets contains all of the trace table entries that have been made.

The CICS trace utility program (DFHTUP) can be used to process and print selected trace entries from the data set (for example, all the EXEC interface program trace entries). The printout shows the trace entries in the same format as that used to display the main storage trace table in a CICS formatted dump.

You will find more information on using the auxiliary trace facility in the *CICS/MVS Operations Guide*.

User trace entry point and event monitoring point (ENTER)

```
ENTER
TRACEID(data-value)
[FROM(data-area)]
[RESOURCE(name)]
[ENTRYNAME(name)]
[ACCOUNT]
[MONITOR]
[PERFORM]

Condition: INVREQ
```

This command is used to specify a point within an application program at which a user trace table entry is to be produced (if the trace facility has been turned on for this type of entry).

This command is used also to define a user event monitoring point (specify MONITOR). The classes of monitoring data for which user information is to be collected at this user event monitoring point can be specified by the ACCOUNT and PERFORM options.

A trace identifier in the range 0 through 199 must be provided in the TRACEID option; this will appear in the first byte of the trace table entry that is produced. Optionally, 8 bytes of data can be supplied in the FROM option; this data will appear in bytes 8 through 15 of the trace table entry.

Additionally, an 8-character name may be supplied in the RESOURCE option; this name will appear in the resource field (bytes 16 through 23) of the trace table entry.

For a user event monitoring point, the TRACEID specified should match the identification number of a TYPE=EMP entry in the monitoring control table that defines the user information to be collected. If no such entry exists, the ENTER command will have no effect. This provides a way of coding optional recording points which are activated by the use of an appropriate monitoring control table.

If both the ACCOUNT and PERFORM options are specified in the application program, the corresponding entry in the monitoring control table can specify recording of either accounting or performance data, or both. If only one option is specified at the user EMP, only that class of recording is possible. Thus greater flexibility is obtained by specifying both options for the user EMP and controlling run-time activity by a suitably coded monitoring control table.

The following example shows how to specify that a user trace table entry should be produced:

```
EXEC CICS ENTER TRACEID(123)
                FROM(MSG)
```

Control the CICS trace facility (TRACE ON, TRACE OFF)

```
TRACE
{ON|OFF}
[SYSTEM]
[EI]
[USER]
[SINGLE]
```

This command is used to control the CICS trace facility by turning on and off the various trace flags. (See "Trace facility control" on page 317 for details of trace flags.)

A TRACE ON or TRACE OFF command without options controls the entire CICS trace facility but leaves the established pattern of trace activity undisturbed.

The SYSTEM option controls the master system trace flag, which must be on before any system trace table entries are produced. The EI option controls the EXEC-interface-program system trace flag. The USER option controls the master user trace flag, and the SINGLE option controls the user trace flag for the task.

The following example shows how to turn on the master system and EXEC interface program system trace flags to start tracing of CICS commands:

```
EXEC CICS TRACE ON SYSTEM EI
```

Macro-level trace facilities

If debugging at the macro level is necessary, an additional option, ALL, can be used, specifying that the entire CICS trace facility is to be controlled by the TRACE ON and TRACE OFF commands. It has the same effect as a master terminal trace control instruction and affects all master, system, and user trace flags.

The following options can only be used in conjunction with the SYSTEM option but no system trace entries will be produced unless the master system trace flag is on. Each option specifies that the system trace entries produced by the associated program are controlled by the TRACE ON and TRACE OFF commands. The options can be specified in any combination and in any order.

Option	CICS Program
BF	Built-in Function
BM	Basic Mapping Support
DC	Dump Control
DI	Batch Data Interchange
FC	File Control
IC	Interval Control
IS	ISC
JC	Journal Control
KC	Task Control
PC	Program Control
SC	Storage Control
SP	Sync Point
TC	Terminal Control
TD	Transient Data
TS	Temporary Storage
UE	User Exit Interface

Trace control options

ACCOUNT
specifies, for a user event monitoring point, that user information is to be collected in the accounting class monitoring data records.

EI specifies that tracing of CICS commands through the EXEC interface program is affected by the TRACE ON or TRACE OFF command.

ENTRYNAME(name)
specifies a qualifier (up to 8 characters in length) for a user event monitoring point. If this option is omitted, a default entry name of USER will be assumed.

FROM(data-area)
specifies an 8-byte data area whose contents are to be entered into the data field of the trace table entry. When used for monitoring, the data area is regarded as two successive fullword fields. These correspond, in order, to the areas DATA1 and DATA2, the required contents of which depend on the option specified in the DFHMCT TYPE=EMP system macro. If the FROM option is omitted, two fullwords of binary zeros are passed as the values of DATA1 and DATA2.

MONITOR
specifies that a user event monitoring point, rather than a trace entry point, is to be recorded.

PERFORM
specifies, for a user event monitoring point, that user information is to be collected in the performance class monitoring data records.

RESOURCE(name)
specifies an 8-character name which is to be entered into the resource field of the trace table entry.

SINGLE
specifies that the TRACE ON or TRACE OFF command applies to user entries of the single task issuing the request for the duration of the task. This option is only effective if user tracing is already active, or if you specify the 'user' option along with this one.

SYSTEM
specifies that all trace entries made from within CICS are affected by the TRACE ON or TRACE OFF command.

This option controls the master system trace flag but does not change the status of individual system trace flags; the established pattern of system trace activity remains intact but is controlled as a single unit. (This characteristic is useful when macro-level trace facilities are in use, as described earlier in the chapter.)

TRACEID(data-value)
specifies the trace identifier for a user trace table entry as a halfword binary value in the range 0 through 199. When used for monitoring, the data value is the user-event monitoring point identifier as specified in the DFHMCT TYPE=EMP system macro.

USER
specifies that all user entries for all current transactions are affected by the TRACE ON or TRACE OFF command.

Trace control exceptional conditions

INVREQ
occurs when TRACEID is greater than 199.

Chapter 5.4. Dump control

The CICS dump control program allows you to specify areas of main storage to be dumped, by means of the DUMP command, onto a sequential data set, which can be either on tape or on disk. The contents of the data set can be formatted subsequently and printed offline (or while the dump data set is closed) using the CICS dump utility program (DFHDUP).

Only one dump control command is processed at a time. If you issue additional dump control commands while a dump is in progress, activity within the tasks associated with those commands is suspended until the dump is completed. Remaining dump commands are processed in the order in which they are made. The use of the DUMP command will cause certain fields (for example, EIBFN and EIBRCODE) in the EIB and the TCA to be overwritten.

Options of the DUMP command allow you to dump the following areas of main storage in various combinations:

- Task-related storage areas: selected main storage areas related to the requesting task. You would normally use a dump of these areas to test and debug your application program. (CICS automatically provides this service if the related task is terminated abnormally.)
- CICS control tables:
  - File control table (FCT)
  - Destination control table (DCT)
  - Program control table (PCT)
  - Processing program table (PPT)
  - System initialization table (SIT)
  - Terminal control table (TCT).

  A dump of these tables is typically the first dump taken in a test in which the base of the test must be established; subsequent dumps are usually of the task-related storage type.
- Task-related storage areas and CICS control tables (a complete dump): a complete dump is sometimes appropriate during execution of a task, but do not use this facility excessively. CICS control tables are primarily static areas; you will find that specifying one CICS control tables dump and a number of task related storage dumps is generally more efficient than specifying a comparable number of complete dumps.

Program storage will not be dumped for programs defined in the PPT as RELOAD=YES.

You will also get a list of the CICS nucleus modules and active PPT programs, indexed by address, at the end of the printed dump.

The dump produced by the DUMP command displays the registers belonging to DFHEDC at the point of invocation of the dump control program at the macro level, not the registers belonging to the application at the time that the DUMP command is issued.

Dump main storage (DUMP)

```
DUMP
DUMPCODE(name)
[FROM(data-area)
  LENGTH(data-value)
    |FLENGTH(data-value)]
[COMPLETE]
[TASK]
[STORAGE]
[PROGRAM]
[TERMINAL]
[TABLES]
[DCT]
[FCT]
[PCT]
[PPT]
[SIT]
[TCT]
```

This command is used to dump any or all of the main storage areas related to a task, any or all of the CICS tables (FCT, DCT, PCT, PPT, SIT, TCT), or all of these together.

The following example shows how to request a dump of the entire task-related storage areas, the terminal control table, and a specified data area:

```
EXEC CICS DUMP
     TASK
     TCT
     FROM(AREA1)
     LENGTH(200)
     DUMPCODE('DUM1')
```

Dump control options

You can specify the dump control options in any combination; only one copy of each area or table will be dumped, even if you have specified it more than once.

If you do not specify any options, that is you specify simply EXEC CICS DUMP, the areas dumped will be the same as those dumped when you specify the TASK option, except that the DL/I control blocks will not be dumped.

COMPLETE
dumps all main storage areas related to a task, all of the CICS tables, and the DL/I control blocks.

DCT
dumps the destination control table.

DUMPCODE(name)
specifies a name (up to four characters) that identifies the dump.

FCT
dumps the file control table.

FLENGTH(data-value)
specifies, as a fullword binary value, the length of the storage area (specified in the FROM option) that is to be dumped. The maximum length that you can specify is 16,777,215 bytes.

FLENGTH and LENGTH are mutually exclusive.

FROM(data-area)
dumps the specified data area which must be a valid area, that is, storage allocated by the operating system within the CICS region. In addition, the following areas are dumped:

- Task control area (TCA) and, if applicable, the transaction work area (TWA).
- Common system area (CSA), including the user's portion of the CSA (CWA).
- Trace table (only when the CICS trace facility is active).
- Contents of general-purpose registers upon entry to dump control from the requesting task.
- Either the terminal control table terminal entry (TCTTE) or the destination control table entry associated with the requesting task.

Whenever the TCTTE is dumped, the terminal control table user area (if any) and the message control blocks (if any) associated with the TCTTE are dumped. The latter are used by basic mapping support.

LENGTH
specifies the length (halfword binary) of the data area specified in the FROM option.

LENGTH and FLENGTH are mutually exclusive.

PCT
dumps the program control table.

PPT
dumps the processing program table.

PROGRAM
specifies that program storage areas associated with this task are to be dumped, as follows:

- Task control area (TCA) and, if applicable, the transaction work area (TWA).
- Common system area (CSA), including the user's portion of the CSA (CWA).
- Trace table (only when the CICS trace facility is active).
- All program storage areas containing user-written application program(s) active on behalf of the requesting task.
- Register save areas (RSAs) indicated by the RSA chain off the TCA.
- Contents of general-purpose registers upon entry to dump control from the requesting task.
- Either the terminal control table terminal entry (TCTTE) or the destination control table entry associated with the requesting task.

Whenever the TCTTE is dumped, the terminal control table user area (if any) and the message control blocks (if any) associated with the TCTTE are dumped.

SIT
dumps the system initialization table.

STORAGE
specifies that storage areas associated with this task are to be dumped, as follows:

- Task control area (TCA) and, if applicable, the transaction work area (TWA).
- Common system area (CSA), including the user's portion of the CSA (CWA).
- Trace table (only when the CICS trace facility is active).
- Contents of general-purpose registers upon entry to dump control from the requesting task.
- All transaction storage areas chained off the TCA storage accounting field.
- Either the terminal control table terminal entry (TCTTE) or the destination control table entry associated with the requesting task.

Whenever the TCTTE is dumped, the terminal control table user area (if any) and the message control blocks (if any) associated with the TCTTE are dumped.

TABLES
dumps the DCT, FCT, PCT, PPT, SIT, and the TCT.

TASK
specifies that storage areas associated with this task are to be dumped, as follows:

- Task control area (TCA) and, if applicable, the transaction work area (TWA).
- Common system area (CSA), including the user's portion of the CSA (CWA).
- Trace table (only when the CICS trace facility is active).

- All program storage areas containing user-written application programs active on behalf of the requesting task.
- Contents of general-purpose registers upon entry to dump control from the requesting task.
- All transaction storage areas chained off the TCA storage accounting field.
- Either the terminal control table terminal entry (TCTTE) or the destination control table entry associated with the requesting task.
- Register save areas (RSAs) indicated by the RSA chain off the TCA.
- All terminal input/output areas (TIOAs) chained off the terminal control table terminal entry (TCTTE) for the terminal associated with the requesting task.
- DL/I control blocks.

Whenever the TCTTE is dumped, the terminal control table user area (if any) and the message control blocks (if any) associated with the TCTTE are dumped.

TCT
dumps the terminal control table.

TERMINAL
specifies that storage areas associated with the terminal are to be dumped, as follows:

- Task control area (TCA) and, if applicable, the transaction work area (TWA).
- Common system area (CSA), including the user's portion of the CSA (CWA).
- Trace table (only when the CICS trace facility is active).
- All terminal input/output areas (TIOAs) chained off the terminal control table terminal entry (TCTTE) for the terminal associated with the requesting task.
- Contents of general-purpose registers upon entry to dump control from the requesting task.
- Either the terminal control table terminal entry (TCTTE) or the destination control table entry associated with the requesting task.

Whenever the TCTTE is dumped, the terminal control table user area (if any) and the message control blocks (if any) associated with the TCTTE are dumped. The latter are used by basic mapping support.

Dump control exceptional conditions

There are no dump control exceptional conditions.

Chapter 5.5. Journal control

CICS provides facilities for creating and managing **journals** during CICS execution. A journal is a set of special-purpose sequential data sets. Journals may contain any and all data the user needs to facilitate subsequent reconstruction of events or data changes. For example, a journal might act as an audit trail, a change-file of database updates and additions, or a record of transactions passing through the system (often called a **log**). Each journal can be written from any task.

Only the CICS facilities dealing with the creation of journal records (journal output) using journal control commands are dealt with here; the *CICS/MVS Resource Definition (Macro)* manual contains information about reading journal data sets (journal input), which involves the use of CICS journal control macros.

Journal control commands are provided to allow the application programmer to:

- Create a journal record (JOURNAL).
- Synchronize with (wait for completion of) journal output (WAIT JOURNAL).

Exceptional conditions that occur during execution of a journal control command are handled as described in "Chapter 1.5. Exceptional conditions" on page 43.

Journal records: Data may be directed to any journal specified in the journal control table (JCT), which defines the journals available during a particular CICS execution. The JCT may define one or more journals on tape or direct access storage. Each journal is identified by a number known as the journal identifier. This number may range from 2 through 99; the value 1 is reserved for a journal known as the system log.

When a journal record is built, the data is moved to the journal buffer area. All buffer space and other work areas needed for journal operations are acquired and managed by CICS. The user task supplies only the data to be written to the journal.

Journal records are built into blocks compatible with standard variable-blocked format. CICS uses the sequential access method of the host operating system to write the blocks to auxiliary storage.

Each journal record begins with a standard length field (LLbb), 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 only which journal to use, what user data to specify, and what user-identifier to supply.

Journal output synchronization: When a journal record is created by issuing the JOURNAL command with the WAIT option, the requesting task can wait until the output has been completed. By specifying that this should happen, the application programmer ensures that the journal record is written on the external storage device associated with the journal before processing continues; the task is said to be **synchronized** with the output operation.

The application programmer can also request asynchronous journal output. This causes a journal record to be created in the journal buffer area and, optionally, initiates the data output operation from the buffer to the external device, but allows the requesting task to retain control and thus to continue with other processing. The task may check and wait for output completion (that is, synchronize) at some later time by issuing the WAIT JOURNAL command.

The basic process of building journal records in the buffer space of a given journal continues until one of the following events occurs:

- A request specifying the STARTIO option is made (from any task) for output of a journal record.
- A request is rejected because of insufficient journal buffer space.
- The available buffer space is reduced below an amount that is specified by the system programmer.
- One second elapses after the last occasion on which any task issued an implied or explicit wait on records in this journal buffer.

When any one of these occurs, all journal records present in the buffer, including any deferred output resulting from asynchronous requests, are written to auxiliary storage as one block.

The advantages that may be gained by deferring journal output are:

- Transactions may get better response times by waiting less.
- The load of physical I/O requests on the host system may be reduced.
- Journal data sets may contain fewer but larger blocks and so better utilize auxiliary storage devices.

However, these advantages are achievable only at the cost of more buffer space and greater programming complexity. It is necessary to plan and program to control synchronizing with journal output. Additional decisions that depend on the data content of the journal record and how it is to be used must be made in the application program. In any case, the full benefit of deferring journal output is obtained only when the load on the journal is high.

The STARTIO option is used with JOURNAL output requests to specify that the journal output operation is to be initiated immediately. For asynchronous output requests, control returns directly to the requesting program. The STARTIO option should not be used unnecessarily because, if every journal request used STARTIO, no improvement over synchronous output requests, in terms of reducing the number of physical I/O operations and increasing the average block size, would be possible.

If the journal buffer space available at the time of the request is not sufficient to contain the journal record, the NOJBUFSP exceptional condition occurs. If no HANDLE CONDITION request is active for this condition, the requesting task loses control, the contents of the current buffer are written out, and the journal record is built in the resulting freed buffer space before control returns to the requesting task.

If the requesting task is not willing to lose control (for example, if some housekeeping must be performed before other tasks get control), a HANDLE CONDITION command should be issued. If the NOJBUFSP condition occurs, no journal record is built for the request, and control is returned directly to the requesting program at the location provided in the HANDLE CONDITION request. The requesting program can perform any housekeeping needed before reissuing the journal output request.

Create a journal record (JOURNAL)

```
JOURNAL
JFILEID(data-value)
JTYPEID(data-value)
FROM(data-area)
[LENGTH(data-value)]
[REQID(data-area)]
[PREFIX(data-value)
  PFXLENG(data-value)]]
[STARTIO]
[WAIT]
[NOSUSPEND]

Conditions: IOERR, JIDERR, LENGERR,
NOJBUFSP, NOTAUTH, NOTOPEN
```

This command is used to create a journal record. The request can be for synchronous or asynchronous output; definitions of these terms, and detailed information regarding the synchronization of journal output, are contained in "Journal output synchronization" on page 327. The following options must be specified.

- JFILEID specifies the journal to receive the data. (JFILEID(1) specifies the system log.)
- JTYPEID specifies a two-character identifier for the journal record.
- FROM specifies the user data to be included in the journal record.
- LENGTH specifies the length of the user data. This length does not include anything reserved for CICS.

The following are optional:

- PREFIX specifies the user prefix data for the journal record.
- PFXLENG specifies the length of the prefix data.

To request synchronous journal output the WAIT option must be specified. For asynchronous output, (WAIT option not specified), the REQID option can be included to provide an identifier for the journal record; the identifier can be used later in a WAIT JOURNAL command to synchronize the task with the creation of the journal record.

The STARTIO option can be included in a synchronous or asynchronous request to specify that the journal output operation should start immediately. STARTIO reduces absolute waiting time at the expense of general system performance and input/output load.

The following example shows how to request synchronous journal output and wait for the output operation to be completed:

```
EXEC CICS JOURNAL
     JFILEID(2)
     JTYPEID('XX')
     FROM(KEYDATA)
     LENGTH(8)
     PREFIX(PROGNAME)
     PFXLENG(6)
     WAIT
```

In this example, because STARTIO is not specified, the task will wait until the journal buffer is full or until output is initiated by a STARTIO request in another task. CICS limits the wait to one second.

The following example shows how to request deferred (asynchronous) journal output:

```
EXEC CICS JOURNAL
     FROM(COMDATA)
     LENGTH(10)
     JFILEID(1)
     JTYPEID('SD')
     REQID(ENTRYID)
```

Synchronize with journal output (WAIT JOURNAL)

```
WAIT JOURNAL
JFILEID(data-value)
[REQID(data-value)]
[STARTIO]

Conditions: INVREQ, IOERR, JIDERR,
NOTAUTH, NOTOPEN
```

This command is used to synchronize 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.

The JFILEID option specifies the journal identifier, and the REQID option optionally specifies a particular journal record. If the REQID option is not specified, the task is synchronized with the output of the the last record created for the journal specified in the JFILEID option.

The journal records in the journal buffer area may already be written out to auxiliary storage, or the journal record output operation may be in progress. If the output operation has already been completed, control returns immediately to the requesting task; if not, the requesting task waits until the operation has been completed. If STARTIO is specified, output is initiated immediately.

If the requesting program has made a succession of successful asynchronous output requests to the same journal, it is necessary to synchronize on only the last of these requests to ensure that all of the journal records have reached auxiliary storage. This may be done either by issuing a stand-alone WAIT JOURNAL command, or by making the last output command itself synchronous (by specifying the WAIT option in the JOURNAL command).

The following example shows how to request synchronization with the output of a journal record:

```
EXEC CICS WAIT JOURNAL
     JFILEID(4)
     REQID(ENTRYID)
```

Journal control options

FROM(data-area)
specifies the user data to be built into the journal record.

JFILEID(data-value)
specifies a halfword numeric value in the range 1 through 99 to be taken as the journal identifier. The value 1 specifies that the system log data set is the journal for this operation.

JTYPEID(data-value)
specifies a two-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 1 and the maximum value is (buffer size – 72) minus PFXLENG.

NOSUSPEND
specifies that application program suspension for the NOJBUFSP condition is to be inhibited. This condition will be handled as described on page 43.

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 maximum value is (buffer size – 72) minus LENGTH.

PREFIX(data-value)
specifies the user prefix data to be included in the journal record. A data area must be provided in COBOL programs.

REQID(parameter)
specifies a fullword binary variable. For a JOURNAL command, the REQID option specifies that asynchronous output is required; the parameter must be a data area. CICS sets the variable to a number that depends upon the position in the data set of the record being created.

When used with a WAIT JOURNAL command, the REQID option specifies a variable set to a number that refers to the journal record that has been created but possibly not yet written out; the parameter is a data value.

STARTIO
specifies that output of the journal record is to be initiated immediately. If WAIT is specified for a journal with a low utilization, STARTIO should be specified also to prevent the requesting task waiting for the journal buffer to be filled. Very high utilization ensures that the buffer is flushed quickly, so that STARTIO is unnecessary.

WAIT
specifies that synchronous journal output is required. The journal record is written out; the requesting task waits until the record has been written.

Journal control exceptional conditions

INVREQ

occurs if a WAIT JOURNAL command is issued before any JOURNAL command has been issued in the same task.

Default action: terminate the task abnormally.

IOERR

occurs if the physical output of a journal record was not accomplished because of an unrecoverable 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 data set, as specified in the JCT entry for the data set, 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 buffer space allocated by the system programmer is not sufficient to contain a journal record.

Default action: write out the contents of the current buffer; suspend task activity until the JOURNAL command is satisfied.

NOTAUTH

occurs if a resource security check has failed. The reasons for the failure are the same as for abend code AEY7, as described in the *CICS/MVS Messages and Codes* manual.

Default action: terminate the task abnormally.

NOTOPEN

occurs if the journal command cannot be satisfied because the specified journal was never opened, and is not available.

Default action: terminate the task abnormally.

Chapter 5.6. Recovery (sync points)

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 certain 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 dynamic transaction backout program. If the CICS system fails, backout is performed as part of the emergency restart process. The *CICS/MVS Facilities and Planning Guide* and the *CICS/MVS Customization Guide* describe these facilities, which in general have no effect on the coding of application programs.

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 SYNCPOINT command to split the program into logically separate sections termed **logical units of work (LUWs)**; the end of an LUW is called a synchronization point (**sync point**).

In addition to those defined with the SYNCPOINT command, sync points also occur at the end of a task and at each DL/I termination or checkpoint (CHKP) call. For the purposes of backout, each of these sync points is treated as though it marked the end of a task. If failure occurs after a sync point but before the task has been completed, only changes made since the sync point are backed out.

LUWs should be entirely logically independent, not merely with regard to protected resources, but also with regard to execution flow. Typically, an LUW would comprise a complete conversational operation bounded by SEND and RECEIVE commands. A browse is another example of an LUW. An ENDBR command must therefore precede the sync point.

In addition to a DL/I termination call being considered to be a sync point, the execution of a SYNCPOINT command will cause CICS to issue a DL/I termination call. If a DL/I PSB is required in a subsequent LUW, it must be rescheduled by means of a PCB call.

A BMS logical message started but not completed when a SYNCPOINT command is executed is forced to completion by an implied SEND PAGE command. However, do not rely on this, because a logical message whose first page is incomplete will be lost. You must also code an explicit SEND PAGE command before the SYNCPOINT command or before termination of the transaction.

Consult the system programmer if sync points are to be issued in a transaction that is eligible for transaction restart.

Establish a sync point (SYNCPOINT)

```
SYNCPOINT [ROLLBACK]

Condition: ROLLEDBACK
```

This command is used to divide a task (usually a long running one) into smaller LUWs. Each SYNCPOINT command causes a sync point to be established to mark the completion of an LUW.

Sync point option

ROLLBACK

specifies that all changes to recoverable resources made by the task since its last sync point are to be backed out.

This option can be used, for example, to tidy up in a HANDLE ABEND routine, or to revoke database changes after the application program finds unrecoverable errors in its input data.

If the LUW updates remote recoverable resources using an MRO or LU6.2 session, the ROLLBACK option is propagated to the backend transaction.

When a distributed transaction processing conversation is in use, the remote application program will have the EIB fields EIBSYNRB, EIBERR, and EIBERRCD set. For the conversation to continue, the remote application program should execute a SYNCPOINT ROLLBACK command.

When the mirror transaction is involved in the LUW using an MRO or LU6.2 session, the mirror will honor the rollback request, revoke changes, and then terminate normally.

This option is not supported across LU6.1 VTAM sessions to the mirror or backend transactions. In these cases, the frontend transactions could be abended to cause the backend transactions to back out. This option must not be used if remote recoverable resources have been updated in the same LUW (because those resources will not be backed out) unless an LU6.2 or MRO session is in use. To back out remote recoverable resources, the transaction could be abended.

When a distributed transaction processing conversation is in use, the remote application program will have the EIB fields EIBSYNRB, EIBERR, and EIBERRCD set. For the conversation to continue, the remote application program should execute a SYNCPOINT ROLLBACK command.

This option can be used, for example, to tidy up in a HANDLE ABEND routine, or to revoke database changes after the application program finds unrecoverable errors in its input data.

This option cannot be used for transactions involved in function shipping or transaction routing on an LU6.1 session.

Sync point exceptional condition

ROLLEDBACK
occurs when a SYNCPOINT command is driven into rollback by a remote system that is unable to commit the sync point. All changes made to recoverable resources in the current LUW will have been backed out.

Default action: terminate the task abnormally.

Part 6. The CICS built-in function command

Chapter 6.1. The field edit built-in function (BIF DEEDIT) command

The built-in function, DEEDIT, is provided by means of the BIF DEEDIT command. BFP=YES must be specified in the DFHSIT system macro for this built-in function to work.

```
BIF DEEDIT
FIELD(data-area)
[LENGTH(data-value)]
```

This command specifies that alphabetic and special characters are to be removed from an EBCDIC data field, the remaining digits being right justified and padded left with zeros as necessary.

This field is specified by the FIELD option and its length, in bytes, by the LENGTH option. A field of 1 byte will be returned unaltered, no matter what the field contains.

If the field ends with a minus sign or a '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 hexadecimal 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 first example below). 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.

For example, execution of the command:

```
EXEC CICS BIF DEEDIT
     FIELD(CONTG)
     LENGTH(9)
```

removes all characters other than digits from CONTG, a nine-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

A decimal point is an EBCDIC special character and as such is removed.

There are no exceptional conditions with DEEDIT.

Appendixes

Appendix A .... EXEC interface block

Appendix B .... Translation tables for the 2980

Appendix C .... CICS macros and equivalent commands

Appendix D .... Sample programs (assembler language)

Appendix E .... Sample programs (COBOL)

Appendix F .... Sample programs (PL/I)

Appendix A. EXEC interface block

This appendix describes the fields of the EXEC interface block (EIB) referred to in "Chapter 1.6. Access to system information" on page 51. An application program can access all of 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 application programming languages ASM, COBOL, and PL/I) are given. All fields contain zeros 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 the 3270.

```
ASM:    CL1
COBOL:  PIC X(1)
PL/I:   CHAR(1)
```

EIBATT

indicates that the RU contains attach header data (X'FF').

```
ASM:    CL1
COBOL:  PIC X(1)
PL/I:   CHAR(1)
```

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.

```
ASM:    H
COBOL:  PIC S9(4) COMP
PL/I:   FIXED BIN(15)
```

EIBCOMPL

indicates, on a terminal control RECEIVE command whether the data is complete (X'FF'). If the NOTRUNCATE option has been used on the RECEIVE command, CICS will retain data in excess of the amount requested via the LENGTH or MAXLENGTH option. EIBRECV will be set indicating that further RECEIVE commands are required. EIBCOMPL will not be set until the last of the data has been retrieved.

EIBCOMPL will always be set when a RECEIVE command without the NOTRUNCATE option is executed.

```
ASM:    CL1
COBOL:  PIC X(1)
PL/I:   CHAR(1)
```

EIBCONF

indicates that a CONFIRM request has been received (X'FF') for an LU6.2 conversation.

```
ASM:    CL1
COBOL:  PIC X(1)
PL/I:   CHAR(1)
```

EIBCPOSN

contains the cursor address (position) associated with the last terminal control or basic mapping support (BMS) input operation from a display device such as the 3270.

```
ASM:    H
COBOL:  PIC S9(4) COMP
PL/I:   FIXED BIN(15)
```

EIBDATE

contains the date the task is started (this field is updated by the ASKTIME command). The date is in packed decimal form (00YYDDD+).

```
ASM:    PL4
COBOL:  PIC S9(7) COMP-3
PL/I:   FIXED DEC(7,0)
```

EIBDS

contains the symbolic identifier of the last data set referred to in a file control request.

```
ASM:    CL8
COBOL:  PIC X(8)
PL/I:   CHAR(8)
```

EIBEOC

indicates that an end-of-chain indicator appears in the RU just received (X'FF').

```
ASM:    CL1
COBOL:  PIC X(1)
PL/I:   CHAR(1)
```

EIBERR

indicates that an error has been received (X'FF') on an LU6.2 conversation.

```
ASM:    CL1
COBOL:  PIC X(1)
PL/I:   CHAR(1)
```

EIBERRCD

when EIBERR is set, contains the error code that has been received. The following values can be returned in the first two bytes of EIBERRCD:

X'0889' Conversation error detected

X'0824' SYNCPOINT ROLLBACK requested

```
ASM:    CL4
COBOL:  PIC X(4)
PL/I:   CHAR(4)
```

EIBFMH

indicates that the user data just received or retrieved contains an FMH (X'FF').

ASM:	CL1
COBOL:	PIC X(1)
PL/I:	CHAR(1)

EIBFN

contains a code that identifies the last CICS command to be issued by the task (updated when the requested function has been completed).

Note: The INQUIRE and SET commands of the command level application program interface, together with the spool commands of the CICS interface to JES, are primarily for the use of the system programmer; they are not described in this book. For details of the commands, see the *CICS/MVS Customization Guide*. However, the EIBFN codes for these commands are included in the following list.

ASM:	CL2
COBOL:	PIC X(2)
PL/I:	CHAR(2)

Code	Command
0202	ADDRESS
0204	HANDLE CONDITION
0206	HANDLE AID
0208	ASSIGN
020A	IGNORE CONDITION
020C	PUSH
020E	POP
0402	RECEIVE
0404	SEND
0406	CONVERSE
0408	ISSUE EODS
040A	ISSUE COPY
040C	WAIT TERMINAL
040E	ISSUE LOAD
0410	WAIT SIGNAL
0412	ISSUE RESET
0414	ISSUE DISCONNECT
0416	ISSUE ENDOUTPUT
0418	ISSUE ERASEAUP
041A	ISSUE ENDFILE
041C	ISSUE PRINT
041E	ISSUE SIGNAL
0420	ALLOCATE
0422	FREE
0424	POINT
0426	BUILD ATTACH
0428	EXTRACT ATTACH
042A	EXTRACT TCT
042C	WAIT CONVID
042E	EXTRACT PROCESS
0430	ISSUE ABEND
0432	CONNECT PROCESS
0434	ISSUE CONFIRMATION
0436	ISSUE ERROR
0438	ISSUE PREPARE
043A	ISSUE PASS
043C	EXTRACT LOGONMSG
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	JOURNAL
1404	WAIT JOURNAL
1602	SYNCPOINT
1802	RECEIVE MAP
1804	SEND MAP
1806	SEND TEXT
1808	SEND PAGE
180A	PURGE MESSAGE
180C	ROUTE
180E	RECEIVE PARTN
1810	SEND PARTNSET
1812	SEND CONTROL
1A02	TRACE
1A04	ENTER

1C02	DUMP
1E02	ISSUE ADD
1E04	ISSUE ERASE
1E06	ISSUE REPLACE
1E08	ISSUE ABORT
1E0A	ISSUE QUERY
1E0C	ISSUE END
1E0E	ISSUE RECEIVE
1E10	ISSUE NOTE
1E12	ISSUE WAIT
1E14	ISSUE SEND
2002	BIF DEEDIT
2202	ENABLE
2004	DISABLE
2006	EXTRACT EXIT
4A02	ASKTIME ABSTIME
4A04	FORMATTIME
4C02	INQUIRE DATASET
4C04	SET DATASET
4E02	INQUIRE PROGRAM
4E04	SET PROGRAM
5002	INQUIRE TRANSACTION
5004	SET TRANSACTION
5202	INQUIRE TERMINAL
5204	SET TERMINAL
5206	INQUIRE NETNAME
5402	INQUIRE SYSTEM
5404	SET SYSTEM
5802	INQUIRE CONNECTION
5804	SET CONNECTION
5A02	INQUIRE MODENAME
5A04	SET MODENAME

EIBFREE

indicates that the application program cannot continue using the facility. The application program should either free the facility or should terminate so that the facility is freed by CICS (X'FF').

```
ASM:    CL1
COBOL:  PIC X(1)
PL/I:   CHAR(1)
```

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, 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 LU6.2 links only.

```
ASM:    CL1
COBOL:  PIC X(1)
PL/I:   CHAR(1)
```

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. For a normal response, this field contains hexadecimal zeros (6X'00').

Almost all of the information in this field can be used within application programs by the HANDLE CONDITION command.

```
ASM:    CL6
COBOL:  PIC X(6)
PL/I:   CHAR(6)
```

The following list contains the values of the various bytes together with the names of the conditions associated with the return codes. For a complete list of response codes, see the *CICS/MVS Problem Determination Guide*.

Note: The INQUIRE and SET commands of the command level application interface, together with the spool commands of the CICS interface to JES are primarily for the use of the system programmer; they are not described in this book. For details of the commands, see the *CICS/MVS Customization Guide*.

```
┌EIBFN (Byte 0)
│
│   ┌Byte(of EIBRCODE)
│   │
│   │  ┌EIBRCODE Value
│   │  │
│   │  │   ┌Condition
│   │  │   │

02 0 E0 INVREQ
02 0 E1 LENGERR

04 0 04 EOF
04 0 10 EODS
04 0 C1 EOF
04 0 C2 ENDINPT
04 0 D0 SYSIDERR[3]
04 0 D2 SESSIONERR[3]
04 0 D3 SYSBUSY
04 0 D4 SESSBUSY
04 0 D5 NOTALLOC
04 0 E0 INVREQ[4]
04 0 E1 LENGERR[5]
04 0 E3 WRBRK
04 0 E4 RDATT
04 0 E5 SIGNAL
04 0 E6 TERMIDERR
```

```
04 0 E7 NOPASSBKRD
04 0 E8 NOPASSBKWR
04 0 EA IGREQCD
04 0 EB CBIDERR
04 0 F1 TERMERR
04 1 20 EOC
04 1 40 INBFMH
04 3 F6 NOSTART
04 3 F7 NONVAL

06 0 01 DSIDERR
06 0 02 ILLOGIC[1]
06 0 08 INVREQ
06 0 0C NOTOPEN
06 0 0D DISABLED
06 0 0F ENDFILE
06 0 80 IOERR[1]
06 0 81 NOTFND
06 0 82 DUPREC
06 0 83 NOSPACE
06 0 84 DUPKEY
06 0 D0 SYSIDERR[3]
06 0 D1 ISCINVREQ
06 0 D6 NOTAUTH
06 0 E1 LENGERR

08 0 01 QZERO
08 0 02 QIDERR
08 0 04 IOERR
08 0 08 NOTOPEN
08 0 10 NOSPACE
08 0 C0 QBUSY
08 0 D0 SYSIDERR[3]
08 0 D1 ISCINVREQ
08 0 D6 NOTAUTH
08 0 E1 LENGERR

0A 0 01 ITEMERR
0A 0 02 QIDERR
0A 0 04 IOERR
0A 0 08 NOSPACE
0A 0 20 INVREQ
0A 0 D0 SYSIDERR[3]
0A 0 D1 ISCINVREQ
0A 0 D6 NOTAUTH
0A 0 E1 LENGERR

0C 0 E1 LENGERR
0C 0 E2 NOSTG

0E 0 01 PGMIDERR
0E 0 D6 NOTAUTH
0E 0 E0 INVREQ

10 0 01 ENDDATA
10 0 04 IOERR
10 0 11 TRANSIDERR
10 0 12 TERMIDERR
10 0 14 INVTSREQ
10 0 20 EXPIRED
10 0 81 NOTFND
10 0 D0 SYSIDERR[3]
10 0 D1 ISCINVREQ
10 0 D6 NOTAUTH
10 0 E1 LENGERR
10 0 E9 ENVDEFERR
10 0 FF INVREQ

12 0 32 ENQBUSY
12 0 E0 INVREQ
12 0 E1 LENGERR

14 0 01 JIDERR
14 0 02 INVREQ
14 0 05 NOTOPEN
14 0 06 LENGERR
14 0 07 IOERR
14 0 09 NOJBUFSP
14 0 D6 NOTAUTH

16 0 01 ROLLEDBACK

18 0 01 INVREQ
18 0 02 RETPAGE
18 0 04 MAPFAIL
18 0 08 INVMPSZ[2]
18 0 20 INVERRTERM
18 0 40 RTESOME
18 0 80 RTEFAIL
18 0 E1 LENGERR
18 0 E3 WRBRK
18 0 E4 RDATT
18 1 02 PARTNFAIL
18 1 04 INVPARTN
18 1 08 INVPARTNSET
18 1 10 INVLDC
18 1 20 UNEXPIN
18 1 40 IGREQCD
18 1 80 TSIOERR
18 2 01 OVERFLOW
18 2 04 EODS
18 2 08 EOC
18 2 10 IGREQID

1A 0 E0 INVREQ

1E 0 04 DSSTAT
1E 0 08 FUNCERR
1E 0 0C SELNERR
1E 0 10 UNEXPIN
1E 0 E1 LENGERR
1E 1 11 EODS
1E 1 2B IGREQCD
1E 2 20 EOC

4A 3 01 ERROR

56 3 0D NOTFND
56 3 10 INVREQ
56 3 11 IOERR
56 3 12 NOSPACE
56 3 14 ENDFILE
56 3 15 ILLOGIC
56 3 16 LENGERR
56 3 46 NOTAUTH
56 3 49 WRONGSTAT
```

```
56 3 4A NAMEERROR
56 3 4C CCERROR
56 3 4D MAPERROR
56 3 50 NOSPOOL
```

The following notes apply to the above list of conditions.

Notes:

1. When ILLOGIC or IOERR occurs during file control operations, further information is provided in field EIBRCODE, as follows:

   ```
   bytes 1-4 = BDAM response
   byte 1 = VSAM return code
   byte 2 = VSAM error code
   ```

   Details of these response codes are given in the *Data Management Macro Instructions* manual.

2. When INVMPSZ occurs during BMS operations, byte 3 of field EIBRCODE contains the terminal code; see "Terminal code table" on page 190.

3. When SYSIDERR occurs, further information is provided in

   ```
   0400 Request for an invalid function.
   0404 NOQUEUE option specified but no session
        available.
   0408 Modename not found[3].
   040C Modename invalid[3].
   0410 Task canceled or timed out during
        execution[3].
   0414 Mode group not available[3].
   0418 Mode group draining[3].
   0800 Sysid out of service or released.
   0804 Session could not be bound.
   0C00 Name not that of a TCTSE.
   0C04 Name not that of a valid TCTSE.
   ```

4. When SESSIONERR occurs, further information is provided in bytes 1 and 2 of EIBRCODE, as follows:

   ```
   0800 SYSID out of service.
   0804 Session out of service[4].
   0C00 Name not that of a TCTTE.
   ```

5. When INVREQ occurs during terminal control operations, further information is provided in byte 3 of EIBRCODE as follows:

   ```
   04 ALLOCATE command - TCTTE
      already allocated.
   08 FREE command - TCTTE in
      wrong state.
   0C CONNECT PROCESS command -
      SYNCLVL 2 has been requested
      but cannot be supported on
      the session in use.
   10 EXTRACT ATTACH command -
      invalid data.
   14 SEND command - CONFIRM
      option has been specified
      but conversation is not
      SYNCLVL 1.
   18 EXTRACT TCT command -
      invalid netname.
   1C An invalid command has been
      issued for the terminal or
      logical unit in use.
   20 An invalid command has been
      issued for the LU6.2
      conversation type in use.
   28 GETMAIN failure on ISSUE
      PASS command.
   ```

6. 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
      invalid (FROM)LENGTH has
      been specified, either less
      than zero or greater than
      32767.
   08 On input commands, an
      invalid (TO)LENGTH has
      been specified, greater
      than 32767.
   0C Length error has occurred
      on ISSUE PASS command.
   ```

EIBRECV

indicates that the application program is to continue receiving data from the facility by executing RECEIVE commands (X'FF').

```
ASM:   CL1
COBOL: PIC X(1)
PL/I:  CHAR(1)
```

[3] APPC only

[4] LUTYPE6.1 only

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.

ASM:	CL8
COBOL:	PIC X(8)
PL/I:	CHAR(8)

EIBRESP

contains a binary number corresponding to the condition that has been raised. These numbers are listed below in decimal.

Note: The INQUIRE and SET commands of the command level application interface, together with the spool commands of the CICS interface to JES, are primarily for the use of the system programmer; they are not described in this book. For details of the commands, see the *CICS/MVS Customization Guide*. However, the EIBRESP codes for these commands are included in the following list.

ASM:	F
COBOL:	PIC S(9) COMP
PL/I:	FIXED BIN(31)

No.	Condition
01	ERROR
02	RDATT
03	WRBRK
04	EOF
05	EODS
06	EOC
07	INBFMH
08	ENDINPT
09	NONVAL
10	NOSTART
11	TERMIDERR
12	DSIDERR
13	NOTFND
14	DUPREC
15	DUPKEY
16	INVREQ
17	IOERR
18	NOSPACE
19	NOTOPEN
20	ENDFILE
21	ILLOGIC
22	LENGERR
23	QZERO
24	SIGNAL
25	QBUSY
26	ITEMERR
27	PGMIDERR
28	TRANSIDERR
29	ENDDATA
30	INVTSREQ
31	EXPIRED
32	RETPAGE
33	RTEFAIL
34	RTESOME
35	TSIOERR
36	MAPFAIL
37	INVERRTERM
38	INVMPSZ
39	IGREQID
40	OVERFLOW
41	INVLDC
42	NOSTG
43	JIDERR
44	QIDERR
45	NOJBUFSP
46	DSSTAT
47	SELNERR
48	FUNCERR
49	UNEXPIN
50	NOPASSBKRD
51	NOPASSBKWR
52	-
53	SYSIDERR
54	ISCINVREQ
55	ENQBUSY
56	ENVDEFERR
57	IGREQCD
58	SESSIONERR
59	SYSBUSY
60	SESSBUSY
61	NOTALLOC
62	CBIDERR
63	INVEXITREQ
64	INVPARTNSET
65	INVPARTN
66	PARTNFAIL
67	-
68	-
69	-
70	NOTAUTH
71	-
72	-
73	WRONGSTAT
74	NAMEERROR
75	-
76	CCERROR
77	MAPERROR
78	-
79	-
80	NOSPOOL
81	TERMERR
82	ROLLEDBACK
83	END
84	DISABLED
85	ALLOCERR
86	STRELERR
87	OPENERR
88	SPOLBUSY
89	SPOLERR
90	NODEIDERR

EIBRESP2

contains more detailed information that may help explain why the RESP condition has been raised. This field will contain meaningful values (as decimal numbers) only for the INQUIRE, SET, and spool commands.

```
ASM:    F
COBOL:  PIC S(9) COMP
PL/I:   FIXED BIN(31)
```

Note: The INQUIRE and SET commands of the command level application interface, together with the spool commands of the CICS interface to JES, are primarily for the use of the system programmer; they are not described in this book. For details of these commands, see the *CICS/MVS Customization Guide*.

EIBRLDBK

indicates rollback

```
ASM:    CL1
COBOL:  PIC X(1)
PL/I:   CHAR(1)
```

EIBRSRCE

contains the symbolic identifier of the resource being accessed by the latest executed command. For file control commands this will be the name of the data set. For transient data and temporary storage commands it will be the name of the queue. For terminal control commands it will be the name of the terminal or logical unit, except for ISC commands when it will be the name of the LU6.1 session or the LUTYPE6.2 conversation.

Identifiers less than eight characters in length are padded on the right with blanks.

```
ASM:    CL8
COBOL:  PIC X(8)
PL/I:   CHAR(8)
```

EIBSIG

indicates that SIGNAL has been received (X'FF').

```
ASM:    CL1
COBOL:  PIC X(1)
PL/I:   CHAR(1)
```

EIBSYNC

indicates that the application program must take a sync point or terminate. Before either is done, the application program must ensure that any other facilities, owned by it, are put into the send state, or are freed (X'FF').

```
ASM:    CL1
COBOL:  PIC X(1)
PL/I:   CHAR(1)
```

EIBSYNRB

indicates that the application program should issue a SYNCPOINT ROLLBACK command (X'FF'). This field is only set in application programs holding a conversation on an LU6.2 or MRO link.

```
ASM:    CL1
COBOL:  PIC X(1)
PL/I:   CHAR(1)
```

EIBTASKN

contains the task number assigned to the task by CICS. This number will appear in trace table entries generated while the task is in control.

```
ASM:    PL4
COBOL:  PIC S9(7) COMP-3
PL/I:   FIXED DEC(7,0)
```

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+).

```
ASM:    PL4
COBOL:  PIC S9(7) COMP-3
PL/I:   FIXED DEC(7,0)
```

EIBTRMID

contains the symbolic terminal identifier of the principal facility (terminal or logical unit) associated with the task.

```
ASM:    CL4
COBOL:  PIC X(4)
PL/I:   CHAR(4)
```

EIBTRNID

contains the symbolic transaction identifier of the task.

```
ASM:    CL4
COBOL:  PIC X(4)
PL/I:   CHAR(4)
```

Appendix B. Translation tables for the 2980

This appendix contains translation tables for the components of the IBM 2980 General Banking Terminal System. The line codes and processor codes listed are unique to CICS and are represented as standard EBCDIC characters.

KEY	ENGRAVING		LINE	PROCESSOR CODE		HLL
No.	Top(LC)	Front(UC)	Code	Numeric(LC)	Alpha(UC)	ID
0	MSG ACK	1	F1	AA	F1	
1	SEND AGAIN	Q	D8	D9	D8	
2	CORR	A	C1	C3	C1	
3	HOLD OVRDE	2	F2	C8	F2	
4	VOID	Z	E9	E5	E9	
5	ACCT INQ	W	E6	D8	E6	
6	ACCT TFR	S	E2	AB	E2	2
7	CIF	3	F3	AC	F3	3
8	MISC	X	E7	AD	E7	4
9	CLSD ACCT	E	C5	E7	C5	
10	NO BOOK	D	C4	AE	C4	5
11	MORT LOAN	4	F4	AF	F4	6
12		C	C3	B0	C3	7
13	NEW ACCT	R	D9	B1	D9	8
14	BOOK BAL	F	C6	B2	C6	9
15	INST LOAN	5	F5	B3	F5	10
16	SPEC TRAN	V	E5	B4	E5	11
17	SAV BOND	T	E3	B5	E3	12
18	SAV	G	C7	B6	C7	13
19	XMAS CLUB	6	F6	B7	F6	14
20	"	B	C2	4B	C2	
21	DDA	Y	E8	B8	E8	15
22	00	H	C8	B9	C8	16
23	MON ORD	7	F7	BA	F7	17
24	0	N	D5	F0	D5	
25	7	U	E4	F7	E4	
26	4	J	D1	F4	D1	
27	CSHR CHK	8	F8	BB	F8	18
28	1	M	D4	F1	D4	
29	8	I	C9	F8	C9	
30	5	K	D2	F5	D2	
31	CASH RECD	9	F9	BC	F9	19
32	2	'	6B	F2	6B	
33	9	0	D6	F9	D6	
34	6	L	D3	F6	D3	
35	UTIL BILL	0	F0	E4	F0	
36	3	"	4B	F3	4B	
37	DEP +	P	D7	4E	D7	
38	WITH −	$	5B	60	5B	
39	FEES	−	60	C6	60	
40	TOTL	/	61	E3	61	
41	CASH IN	*	5C	BD	5C	20
42	CASH CHK	#	7B	BE	7B	21
43	VAL	&	50	STATION ID	50	
44	TAB		05	05	05	TABCHAR
45	ALPHA ENTRY		36			
46	NUM ENTRY		06			
47	SEND		26-ETB 03-ETX			
48	RETURN		15	15	15	JRNLCR
49	NUM ENTRY		06			
50	SPACE		40	40	40	
58	MSGLIGHT		17	17	17	MSGLITE

Figure 31. 2980-1 teller station character set/translate table

KEY No.	ENGRAVING Top(LC)	 Front(UC)	LINE Code	PROCESSOR CODE Numeric(LC)	 Alpha(UC)	HLL ID
0	= 1		F1	F1 (1)	7E (=)	
1	Q		D8	98 (q)	D8 (Q)	
2	A		C1	81 (a)	C1 (A)	
3	2		F2	F2 (2)	4C (<)	
4	Z		E9	A9 (z)	E9 (Z)	
5	W		E6	A6 (w)	E6 (W)	
6	S		E2	A2 (s)	E2 (S)	
7	; 3		F3	F3 (3)	5E (;)	
8	X		E7	A7 (x)	E7 (X)	
9	E		C5	85 (e)	C5 (E)	
10	D		C4	84 (d)	C4 (D)	
11	: 4		F4	F4 (4)	7A (:)	
12	C		C3	83 (c)	C3 (C)	
13	R		D9	99 (r)	D9 (R)	
14	F		C6	86 (f)	C6 (F)	
15	% 5		F5	F5 (5)	6C (%)	
16	V		E5	A5 (v)	E5 (V)	
17	T		E3	A3 (t)	E3 (T)	
18	G		C7	87 (g)	C7 (G)	
19	' 6		F6	F6 (6)	7D (')	
20	B		C2	82 (b)	C2 (B)	
21	Y		E8	A8 (y)	E8 (Y)	
22	H		C8	88 (h)	C8 (H)	
23	> 7		F7	F7 (7)	6E (>)	
24	N		D5	95 (n)	D5 (N)	
25	U		E4	A4 (u)	E4 (U)	
26	J		D1	91 (j)	D1 (J)	
27	* 8		F8	F8 (8)	5C (*)	
28	M		D4	94 (m)	D4 (M)	
29	I		C9	89 (i)	C9 (I)	
30	K		D2	92 (k)	D2 (K)	
31	(9		F9	F9 (9)	4D (()	
32	| ,		6B	6B (,)	4F (|)	
33	O		D6	96 (o)	D6 (O)	
34	L		D3	93 (l)	D3 (L)	
35	) 0		F0	F0 (0)	5D ())	
36	¬ .		4B	4B (.)	5F (¬)	
37	P		D7	97 (p)	D8 (P)	
38	! $		5B	5B ($)	5A (!)	
39	_ –		60	60 (–)	6D (_)	
40	? /		61	61 (/)	6F (?)	
41	c |		5C	70 (|)	4A (c)	
42	" #		7B	7B (#)	7F (")	
43	+ &		50	50 (&)	4E (+)	
44	TAB		05	05	05	
45	LOCK		36	36	36	
46	SHIFT		06	06	06	
47	BACKSPACE		16	10	16	BCKSPACE
48	RETURN		15	15	15	
49	SHIFT		06	06	06	
50	(SPACE)		40	40	40	
53	SEND		26–ETB 03–ETX			

Figure 32. 2980-2 administrative station character set/translate table

KEY	ENGRAVING		LINE	PROCESSOR CODE		HLL
No.	Top(LC)	Front(UC)	Code	Numeric(LC)	Alpha(UC)	ID
0	CK $	–	D9	BC	60	19
1		Q	D3	D3	D8	
2		A	C1	C1	C1	
3	CK #	0	C9	B7	C9	14
4		Z	E9	4B	E9	
5		W	E6	5C	E6	
6		S	E2	5B	E2	
7	IMD 2	1	5B	4F	F1	
8		X	E7	AE	E7	5
9		E	C5	C5	C5	
10		D	C4	6F	C4	
11	IMD 1	2	4B	BF	F2	
12		C	C3	C3	C3	
13		R	60	60	D9	
14		F	C6	C6	C6	
15	CODE	3	E8	BB	F3	
16		V	E5	A0	E5	22
17		T	E3	A1	E3	23
18		G	C7	C7	C7	
19	AMT	4	5C	BE	F4	21
20		B	C2	C2	C2	
21		Y	61	61	E8	
22		H	D7	D7	C8	
23	0B	5	D8	B2	F5	9
24		N	D5	D5	D5	
25		U	E4	AF	E4	6
26		J	D1	D1	D1	
27	ACCT #	6	C8	7B	F6	
28		N	D4	E7	D4	
29		I	D6	D6	C9	
30		K	D2	D2	D2	
31	7	7	F7	F7	F7	
32	...	...	6B	BLANK	6B	
33	4	0	F4	F4	D6	
34	1	L	F1	F1	D3	
35	8	8	F8	F8	F8	
36	0	.	F0	F0	4B	
37	5	P	F5	F5	D7	
38	2	$	F2	F2	5B	
39	9	9	F9	F9	F9	
40	...	...	7B	B0	7B	7
41	6	*	F6	F6	5C	
42	3	#	F3	F3	7B	
43	VAL	&	50	50	50	
44	TAB		05	05	05	
45	ALPHA		36			
46	NUMERIC		06			
47	SEND		26-ETB 03-ETX			
48	RETURN		15	15	15	
49	NUMERIC		06			
50	SPACE		40	40	40	
51	FEED OPEN		04			OPENCH

Figure 33. 2980-4 teller station character set/translate table

Appendix C. CICS macros and equivalent commands

This appendix provides a list of the macro instructions available to the CICS application programmer, and shows for each macro instruction the command that will perform the same function. Command options may have different defaults or functions from macro-level operands having similar names. Some CICS macros do not have an equivalent command; for example, there is only one CICS built-in function that can be invoked by a command.

Although the TYPE=CHECK macro performs a similar function to the HANDLE CONDITION command, it is used in a completely different way.

Macro	**Command**
DFHBFTA	-
DFHBIF	
TYPE=DEEDIT	BIF DEEDIT
DFHBMS	
TYPE=CHECK	HANDLE CONDITION
TYPE=IN	RECEIVE MAP
TYPE=MAP	RECEIVE MAP FROM
TYPE=OUT	SEND TEXT
TYPE=OUT,MAP=	SEND MAP
TYPE=PAGEBLD	SEND MAP ACCUM
TYPE=PAGEOUT	SEND PAGE
TYPE=PURGE	PURGE MESSAGE
TYPE=RETURN	SEND{MAP|TEXT} SET
TYPE=ROUTE	ROUTE
TYPE=STORE	SEND{MAP|TEXT} PAGING
TYPE=TEXTBLD	SEND TEXT ACCUM
DFHDC	
TYPE=CICS	DUMP TABLES
TYPE=COMPLETE	DUMP COMPLETE
TYPE=PARTIAL	
LIST=PROGRAM	DUMP PROGRAM
LIST=TERMINAL	DUMP TERMINAL
LIST=TRANSACTION	DUMP STORAGE
LIST=SEGMENT	DUMP FROM
TYPE=TRANSACTION	DUMP[TASK]
DFHDI	
TYPE=ABORT	ISSUE ABORT
TYPE=ADD	ISSUE ADD
TYPE=CHECK	HANDLE CONDITION
TYPE=END	ISSUE END
TYPE=ERASE	ISSUE ERASE
TYPE=NOTE	ISSUE NOTE
TYPE=QUERY	ISSUE QUERY
TYPE=RECEIVE	ISSUE RECEIVE
TYPE=REPLACE	ISSUE REPLACE
TYPE=SEND	ISSUE SEND
TYPE=WAIT	ISSUE WAIT
DFHFC	
TYPE=CHECK	HANDLE CONDITION
TYPE=DELETE	DELETE RIDFLD
(DL/I types)	-
TYPE=ESETL	ENDBR
TYPE=GET	READ

TYPE=GET,
TYPOPER=UPDATE
	READ UPDATE

TYPE=GETAREA	-
TYPE=GETNEXT	READNEXT
TYPE=GETPREV	READPREV

TYPE=PUT,
TYPOPER=DELETE
	DELETE

TYPE=PUT,
TYPOPER=NEWREC
	WRITE

TYPE=PUT,
TYPOPER=UPDATE
	REWRITE

TYPE=RELEASE	UNLOCK
TYPE=RESETL	RESETBR
TYPE=SETL	STARTBR
DFHIC	
TYPE=CANCEL	CANCEL
TYPE=CHECK	HANDLE CONDITION
TYPE=GET	RETRIEVE
TYPE=GETIME	ASKTIME
TYPE=INITIATE	START
TYPE=POST	POST
TYPE=PUT	START FROM
TYPE=RETRY	RETRIEVE
TYPE=WAIT	DELAY
DFHJC	
TYPE=CHECK	HANDLE CONDITION
TYPE=GETJCA	-
TYPE=PUT	JOURNAL WAIT
TYPE=WAIT	WAIT JOURNAL
TYPE=WRITE	JOURNAL
DFHKC	
TYPE=ATTACH	-
TYPE=CHAP	-
TYPE=DEQ	DEQ
TYPE=ENQ	ENQ
TYPE=NOPURGE	-
TYPE=PURGE	-
TYPE=WAIT	SUSPEND
TYPE=WAIT,ECADDR	WAIT EVENT
DFHMDF	-

DFHMDI	-
DFHMSD	-
DFHPC	
TYPE=ABEND	ABEND
TYPE=CHECK	HANDLE CONDITION
TYPE=COBADDR	-
TYPE=DELETE	RELEASE
TYPE=LINK	LINK
TYPE=LOAD	LOAD
TYPE=RESETXIT	HANDLE ABEND RESET
TYPE=RETURN	RETURN
TYPE=SETXIT	HANDLE ABEND
TYPE=XCTL	XCTL
DFHSC	
TYPE=FREEMAIN	FREEMAIN
TYPE=GETMAIN	GETMAIN
DFHSP	
TYPE=USER	SYNCPOINT
TYPE=ROLLBACK	SYNCPOINT ROLLBACK
DFHTC	
TYPE=CBUFF	SEND CBUFF
TYPE=CONVERSE	CONVERSE
TYPE=COPY	ISSUE COPY
TYPE=DISCONNECT	ISSUE DISCONNECT
TYPE=EODS	ISSUE EODS
TYPE=ERASEAUP	ISSUE ERASEAUP
TYPE=GET	RECEIVE
TYPE=PAGE	-
TYPE=PASSBK	SEND PASSBK
TYPE=PRINT	ISSUE PRINT
TYPE=PROGRAM	ISSUE LOAD
TYPE=PUT	SEND WAIT
TYPE=READ	RECEIVE(WAIT assumed)
TYPE=READB	RECEIVE BUFFER
TYPE=READL	RECEIVE LEAVEKB
TYPE=RESET	ISSUE RESET
TYPE=SIGNAL	WAIT SIGNAL
TYPE=WAIT	WAIT TERMINAL
TYPE=WRITE	SEND
TYPE=WRITEL	SEND LEAVEKB
DFHTD	
TYPE=CHECK	HANDLE CONDITION
TYPE=FEOV	-
TYPE=GET	READQ TD
TYPE=PURGE	DELETEQ TD
TYPE=PUT	WRITEQ TD
DFHTR	
TYPE=ENTRY	ENTER
TYPE=OFF	TRACE OFF
TYPE=ON	TRACE ON
DFHTS	
TYPE=CHECK	HANDLE CONDITION
TYPE=GET[1]	READQ TS
TYPE=GETQ	READQ TS
TYPE=PURGE	DELETEQ TS
TYPE=PUT[1]	WRITEQ TS
TYPE=PUTQ	WRITEQ TS
TYPE=RELEASE	DELETEQ TS

[1] Because single units of information cannot be handled by the command level interface, data stored by a DFHTS TYPE=PUT macro cannot be retrieved by a READQ TS command or be deleted by a DELETEQ TS command. Conversely, data stored by a WRITEQ TS command cannot be retrieved by a DFHTS TYPE=GET macro.

Appendix D. Sample programs (assembler language)

The assembler language sample programs described in this appendix are included, in both source and executable form, on the CICS distribution tape. The *CICS/MVS Installation Guide* describes how these sample programs, and associated resources, can be defined to CICS and how the programs can be executed online.

This appendix describes six CICS sample application programs, written in assembler language, as follows:

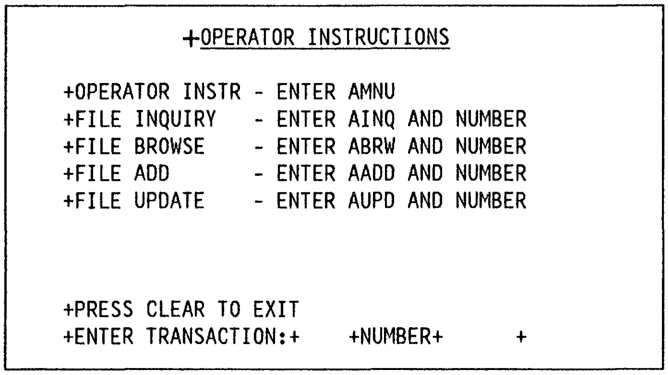
- Operator instruction
- Inquiry/update
- Browse
- Order entry
- Order entry queue print
- Low balance report.

These programs illustrate basic applications (such as inquire, browse, add, and update) that can serve as a framework for your installation's first programs. The programs operate using a VSAM file, known as FILEA, consisting of records containing details of individual customer accounts. Each program has a short description of what the program does, a listing of its source code, and a series of program notes. Numbered coding lines in the source listing correspond to the numbered program notes.

All the sample programs are for use with the IBM 3270 Information Display System.

The sample BMS maps include examples of how the COLOR, EXTATT, and HILIGHT attributes are specified in the map definition macros. However, due to production limitations, the associated screen layouts do not show you all the effects of these attributes.

You can add attributes without changing the application program by specifying EXTATT=MAPONLY in the DFHMSD map set definition macro. If you include an attribute that specifies a facility not available at the terminal, it will be ignored.



```
               +OPERATOR INSTRUCTIONS

    +OPERATOR INSTR - ENTER AMNU
    +FILE INQUIRY   - ENTER AINQ AND NUMBER
    +FILE BROWSE    - ENTER ABRW AND NUMBER
    +FILE ADD       - ENTER AADD AND NUMBER
    +FILE UPDATE    - ENTER AUPD AND NUMBER



    +PRESS CLEAR TO EXIT
    +ENTER TRANSACTION:+    +NUMBER+      +
```

The statements listed are those of the sample programs supplied with the initial release of CICS. Sample programs shipped with subsequent program temporary fixes (PTFs) may differ from these listings.

The BMS maps (which are unaligned) and the file record descriptions used by these sample programs are included at the end of the appendix.

After CICS is running, type AMNU onto a clear screen and press the enter key. The AMNU transaction identifier invokes the "Operator Instruction" sample program, which is a short program that produces a menu containing the transaction identifiers for two of the other sample programs, namely 'Inquiry/Update' and 'Browse'.

If you clear the screen, remember to reenter the transaction identifier, as no data is accepted from an unformatted screen.

You can run the sample programs using EDF but, because the CEDF transaction is defined with RSLC=YES, you must first sign on to CICS as an operator with an appropriate resource security level key.

The menu, on a screen that is 40 characters wide by 12 lines deep, is shown in the box above. The plus (+) sign in this and subsequent displays shows the position of the attribute byte. In an actual display, this position contains a blank.

To invoke any of the transactions AMNU, AINQ, ABRW, AADD, or AUPD, do as instructed, entering the 4-character transaction identifier and, when necessary, the 6-digit account number in the fields highlighted in the bottom line of the display. These special account numbers include the sequence 100000, 111111, 200000, 222222, ... , 999999.

These transaction identifiers give you access to the inquiry, add, and update functions of the 'Inquiry/Update' program, and access to the 'Browse' program.

You can invoke the three remaining sample programs 'Order Entry', 'Order Entry Queue Print', and 'Low Balance Report' separately by entering their transaction identifiers (AORD, AORQ, and AREP respectively) onto a clear screen.

Operator instruction program (ASM)

Description

The operator instruction sample program displays map DFH$AGA in response to the EXEC CICS SEND MAP command.

The map displays a menu that lists the transaction identifiers associated with two of the sample programs, "Inquiry /Update", and 'Browse', and gives instructions for the operator.

Source listing

```
        TITLE 'DFH$AMNU - CICS/MVS SAMPLE FILEA OPERATOR INSTRUCTION M*
              ENU - ASSEMBLER'
DFH$AMNU CSECT
 1      EXEC  CICS SEND MAP('DFH$AGA') MAPONLY ERASE
 2      EXEC  CICS RETURN
        END
```

Program notes

1. The BMS command erases the screen and displays map DFH$AGA.
2. The RETURN command ends the program.

Inquiry/update sample program (ASM)

Description

The inquiry/update sample program lets you make an inquiry about, add to, or update records in a file. You can select one of these by entering the appropriate transaction identifier (AINQ, AADD, or AUPD) in the menu that is displayed when you start operations by entering AMNU.

To make an inquiry, enter AINQ and an account number into the menu. The program maps in the account number and reads the record from FILEA. The required fields from the file area, and a title 'FILE INQUIRY' are moved to the map dsect for DFH$AGB. DFH$AGB, containing the record fields, is displayed at your screen.

To add a record, enter AADD and the account number into the menu. The account number and a title 'FILE ADD' are moved to the map area of DFH$AGB. DFH$AGB, containing empty data fields, is displayed at your screen. The data fields entered are mapped into DFH$AGB and moved to the file record area which is then written to FILEA. The addition is recorded on an update log (LOGA), which is a transient data queue. The operator instruction screen is displayed with the message 'RECORD ADDED'.

To update a record, enter AUPD and the account number into the menu, as before. The program reads and displays the requested FILEA record. Modified data fields are mapped in to DFH$AGB and edited. The sample program only suggests the type of editing you might want to do. Insert editing steps needed to ensure valid changes to the file. Those fields that have been changed are moved to the data record and the record is rewritten to FILEA. The update is recorded on LOGA. The message 'RECORD UPDATED' is moved to the dsect for DFH$AGA, the operator instruction menu map, which is then displayed at your screen.

This program is an example of pseudoconversational programming, in which control is returned to CICS together with a transaction identifier whenever a response is requested from the operator. Associated with each return of control to CICS is a storage area containing details of the previous invocation of the transaction.

Source listing for DFH$AALL

```
        TITLE 'DFH$AALL - CICS/MVS SAMPLE FILEA INQUIRY/UPDATE - ASSEM*
              BLER'
DFHEISTG DSECT
         COPY  DFH$AGA               MAP A
         COPY  DFH$AGB               MAP B
RETREG   EQU   2                     SET UP REGISTER USAGE
R06      EQU   6
R07      EQU   7
R08      EQU   8
R09      EQU   9
FILEDS   DS    0C
         COPY  DFH$AFIL              RECORD DESCRIPTION FOR FILEA
COMPTR   EQU   4                     POINTER TO COMMAREA
         COPY  DFH$ALOG              LOG FILE RECORD DESCRIPTION
         COPY  DFHBMSCA              BMS ATTRIBUTE BYTES
MESSAGES DS    CL39                  TEMP STORE FOR MESSAGES
KEYNUM   DS    CL9                   TEMP STORE FOR FILE RECORD KEY
COMLEN   DS    1H                    LENGTH OF COMMAREA
DFH$AALL CSECT
 1       CLC   EIBTRNID,=CL(L'EIBTRNID)'AINQ' IS INVOKING T-ID 'AINQ'?
         BE    OKTRANID              OK HERE, SO CONTINUE
         CLC   EIBTRNID,=CL(L'EIBTRNID)'AUPD' IS IT 'AUPD'?
         BE    OKTRANID              OK HERE, SO CONTINUE
         CLC   EIBTRNID,=CL(L'EIBTRNID)'AADD' FINALLY, IS IT 'AADD'?
         BNE   ERRORS                IF NOT, GO TO ERROR ROUTINE
OKTRANID DS    0H                    CORRECT INVOKING TRANSACTION ID HERE
 2       LH    COMPTR,EIBCALEN       HAS A COMMAREA BEEN RETURNED?
         LTR   COMPTR,COMPTR
         BNZ   COMRETND              ...YES, SO GO GET MAP
 3       EXEC CICS HANDLE CONDITION MAPFAIL(MFAIL) ERROR(ERRORS)
 4       EXEC CICS RECEIVE MAP('DFH$AGA')
 5       CLC   KEYL,=H'0'            IF ACCOUNT NUMBER NOT ENTERED
         BE    BADLENG                GO DISPLAY ERROR MS.
         TRT   KEYI,CHEKTAB          CHECK FOR NUMERIC ACCOUNT NUM,
         BNZ   BADCHARS               NO GOOD - DISPLAY ERROR MS.
         MVC   KEYNUM,KEYI           SAVE KEY TO FILE.
         XC    DFH$AGBO(DFH$AGBE-DFH$AGBO),DFH$AGBO CLEAR MAP
         CLC   EIBTRNID,=CL(L'EIBTRNID)'AADD' IS INVOKING T-ID 'AADD'?
         BNE   INQUPD                ..NO, SO GO TEST FOR OTHER ID'S
 6       MVC   TITLEO,=CL(L'TITLEO)'FILE ADD' SET UP TITLE
         MVC   MSG3O,=CL(L'MSG3O)'ENTER DATA AND PRESS ENTER KEY'
         MVC   NUMB,KEYI             PUT KEY IN COMMAREA
         MVC   NUMBO,KEYI            ...AND ON MAP ENTRY
         MVI   AMOUNTA,DFHBMUNN    ATTRIBUTE SET TO UNPROTECTED,
*                                  NUMERIC, DISPLAY, MDT BIT NOT SET
         MVC   AMOUNTO,=C'$0000.00'  PROMPTING FIELD FOR MAP
         MVC   COMLEN,=H'7'        SET UP LENGTH OF COMMAREA TO BE RTND
 7       BAL   RETREG,MAPSEND        GO SEND MAP
         B     CICSCONT              GO RETURN CONTROL TO CICS
INQUPD   DS    0H                    HERE INVOKING T-ID IS AINQ, OR AUPD
 8       EXEC CICS HANDLE CONDITION NOTFND(NOTFOUND)
 9       EXEC CICS READ DATASET('FILEA') INTO(FILEA) RIDFLD(KEYNUM)
         CLC   EIBTRNID,=CL(L'EIBTRNID)'AINQ' IS INVOKING T-ID AINQ?
         BNE   UPDTSECT              ..NO, SO BRANCH TO AUPD ROUTINE
```

Source listing for DFH$AALL (continued)

```
10       MVC   TITLEO,=CL(L'TITLEO)'FILE INQUIRY' SET UP TITLE ON MAP
         MVC   MSG3O,=CL(L'MSG3O)'PRESS ENTER TO CONTINUE'
         BAL   RETREG,MAPBUILD     GO BUILD MAP
*                                 PROTECT ALL FIELDS ON MAP
11       MVI   NAMEA,DFHBMPRO
         MVI   ADDRA,DFHBMPRO
         MVI   PHONEA,DFHBMPRO
         MVI   DATEA,DFHBMPRO
         MVI   AMOUNTA,DFHBMPRO
         MVI   COMMENTA,DFHBMPRO
12       BAL   RETREG,MAPSEND      GO SEND MAP
         EXEC CICS RETURN TRANSID('AMNU')
UPDTSECT DS    0H                  UPDATE ROUTINE
13       MVC   TITLEO,=CL(L'TITLEO)'FILE UPDATE' SET UP MAP TITLE
         MVC   MSG3O,=CL(L'MSG3O)'CHANGE FIELDS AND PRESS ENTER'
14       MVC   COMLEN,=H'80'      STORE LENGTH OF COMMAREA
15       BAL   RETREG,MAPBUILD     GO BUILD MAP
         BAL   RETREG,MAPSEND      GO SEND MAP
         B     CICSCONT            GO RETURN CONTROL TO CICS
***********************************************************************
*                                                                     *
*    HERE A COMMAREA HAS BEEN RETURNED, AND IS THEREFORE SECOND       *
*    INVOCATION OF THIS PROGRAM                                       *
*                                                                     *
***********************************************************************
COMRETND DS    0H                  HERE COMMAREA HAS BEEN RETURNED
16       L     COMPTR,DFHEICAP     GET ADDRESSABILITY TO COMMAREA
17       EXEC CICS HANDLE CONDITION MAPFAIL(NOTMODF) ERROR(ERRORS)     *
               DUPREC(DUPREC) NOTFND(NOTFOUND)
18       EXEC CICS RECEIVE MAP('DFH$AGB')
         CLC   EIBTRNID,=CL(L'EIBTRNID)'AUPD' IS INVOKING T-ID AUPD?
         BNE   SECADD              ..NO, SO BRANCH TO SECOND AADD ROUT
19       EXEC CICS READ UPDATE DATASET('FILEA') INTO(FILEA)            *
               RIDFLD(NUMB-FILEDS(COMPTR))
         CLC   FILEREC,FILEREC-FILEDS(COMPTR) RECORD CHANGED ON FILE?
         BE    OKREC               ..NO, SO BRANCH AND CONTINUE
20       MVC   MSG1O,=CL(L'MSG1O)'RECORD UPDATED BY OTHER USER, TRY AGA*
               IN'
         MVI   MSG1A,DFHBMASB      BRIGHTEN MESSAGE ON SCREEN
         MVI   MSG3A,DFHPROTN      DARK AND PROTECTED ATTRIBUTE
         BAL   RETREG,MAPBUILD     GO BUILD MAP
         EXEC CICS SEND MAP('DFH$AGB') DATAONLY
         MVC   COMLEN,=H'80'       SET UP LENGTH OF COMMEREA
         B     CICSCONT            GO RETURN CONTROL TO CICS
OKREC    DS    0H                  HERE RECORD IS OK FOR UPDATE
21       BAL   RETREG,CHECK        GO TEST RECORD TO BE UPDATED
22       MVI   STAT,C'U'           MOVE 'UPDATE' BYTE TO FILE RECORD
         BAL   RETREG,FILESTUP     GO SET UP FILE RECORD
23       MVC   MESSAGES,=CL(L'MESSAGES)'RECORD UPDATED' SET UP MESSAGE
         B     AMNU                COMPLETE, GO FINISH.
```

Source listing for DFH$AALL (continued)

```
SECADD   DS    0H                  SECOND ADD ROUTINE
         MVC   NUMB,NUMB-FILEDS(COMPTR)   MOVE SAVED RECORD KEY TO FILE
24       BAL   RETREG,CHECK        GO CHECK RECORD TO BE ADDED
         XC    FILEDS,FILEDS       RECORD IS OK HERE,SO CLEAR FILE AREA
25       MVI   STAT,C'A'           MOVE 'ADDED' BYTE TO FILE RECORD
         BAL   RETREG,FILESTUP     GO WRITE RECORD ON FILE
26       MVC   MESSAGES,=CL(L'MESSAGES)'RECORD ADDED'  SET UP MESSAGE
         B     AMNU                COMPLETE, GO FINISH.
CICSCONT DS    0H                  THIS ROUTINE RETURNS CONTROL TO CICS
27       EXEC CICS RETURN TRANSID(EIBTRNID) COMMAREA(FILEDS)           *
               LENGTH(COMLEN)
AMNU     DS    0H                  ENDING ROUTINE
28       XC    DFH$AGAO(DFH$AGAE-DFH$AGAO),DFH$AGAO CLEAR MAP
         MVI   MSGA,DFHBMASB       BRIGHTEN MESSAGE FIELD ON MAP
         MVC   MSGO,MESSAGES       MOVE ANY MESSAGE TO MAP AREA
29       EXEC CICS SEND MAP('DFH$AGA') ERASE
30       EXEC CICS RETURN
************************************************************************
*                                                                      *
*                  GENERAL ROUTINES                                    *
*                                                                      *
************************************************************************
MAPBUILD DS    0H                  ROUTINE TO BUILD MAP DFH$AGB
31       MVC   NUMBO,NUMB          MOVE FILE KEY TO MAP AREA
         MVC   NAMEO,NAME          MOVE NAME TO MAP AREA
         MVC   ADDRO,ADDRX         MOVE ADDRESS TO MAP AREA
         MVC   PHONEO,PHONE        MOVE PHONE TO MAP AREA
         MVC   DATEO,DATEX         MOVE DATE TO MAP AREA
         MVC   AMOUNTO,AMOUNT      MOVE AMOUNT TO MAP AREA
         MVC   COMMENTO,COMMENT    MOVE COMMENT TO MAP AREA
         BR    RETREG              RETURN
MAPSEND  DS    0H                  ROUTINE TO SEND MAP DFH$AGB
32       EXEC CICS SEND MAP('DFH$AGB') ERASE
         BR    RETREG              RETURN
CHECK    DS    0H                  ANY INPUT FROM SCREEN?
ROUTINE
33       LA    R06,DFH$AGBO        R6 POINTS TO START OF MAP DFH$AGB
         LA    R07,(DFH$AGBE-DFH$AGBO) R7 CONTAINS LENGTH OF MAP B
         LA    R08,HEXZERO         R8 POINTS TO HEXZERO
         LA    R09,L'HEXZERO       R9 CONTAINS LENGTH OF HEXZERO
         ICM   R09,B'1000',HEXZERO  X'00' INTO TOP BYTE OF R9
         CLCL  R06,R08             DOES MAP CONTAIN ANY INPUT?
         BE    NOTMODF             ..NO, SO RAISE NOTMODIFIED
         CLC   EIBTRNID,=CL(L'EIBTRNID)'AADD' IS INVOKING T-ID 'ADDS'?
         BE    ADNAMCHK            ..YES, SO GO TO 'AADD' NAME CHECK
UPNAMCHK DS    0H                  UPDATE TRANSACTION HERE
         OC    NAMEI,NAMEI         HAS NAME BEEN CHANGED?
         BZR   RETREG              ..NO, SO DON'T CHECK IT
ADNAMCHK TRT   NAMEO,TAB           ..YES, IS IT ALPHABETIC?
         BM    DATAERR             ..NO, SO RAISE ERROR
         BR    RETREG              ..YES, SO RETURN
FILESTUP DS    0H                  ROUTINE TO SET UP FILE RECORD
34       OC    NAMEI,NAMEI         HAS NAME BEEN ENTERED?
         BZ    ADRTST              ..NO, BRANCH
         MVC   NAME,NAMEI          ..YES, PUT IN IN FILE AREA
ADRTST   OC    ADDRI,ADDRI         HAS ADDRESS BEEN ENTERED?
         BZ    PHNTST              ..NO, BRANCH
         MVC   ADDRX,ADDRI         ..YES, PUT IN IN FILE AREA
```

Source listing for DFH$AALL (continued)

```
PHNTST   OC    PHONEI,PHONEI       HAS PHONE BEEN ENTERED?
         BZ    DATTST              ..NO, BRANCH
         MVC   PHONE,PHONEI        ..YES, PUT IN IN FILE AREA
DATTST   OC    DATEI,DATEI         HAS DATE BEEN ENTERED?
         BZ    AMTTST              ..NO, BRANCH
         MVC   DATEX,DATEI         ..YES, PUT IN IN FILE AREA
AMTTST   OC    AMOUNTI,AMOUNTI     HAS AMOUNT BEEN ENTERED?
         BZ    CHEKTRAN            ..NO, BRANCH
         TRT   AMOUNTI,CHEKTAB     IS AMOUNT NUMERIC
         BNZ   DATAERR             NO, ASK FOR CORRECT AMOUNT
         MVC   AMOUNT,AMOUNTI      ..YES, PUT IN IN FILE AREA
         B     COMTST
CHEKTRAN CLC   EIBTRNID,=CL(L'EIBTRNID)'AADD' IS INVOKING T-ID 'ADDS'?
         BNE   COMTST
*                                 PUT VALID AMOUNT IN NEW RECORD
         MVC   AMOUNT,=CL8'$0000.00'
COMTST   OC    COMMENTI,COMMENTI HAS COMMENT BEEN ENTERED?
         BZ    CONTINUE            ..NO, CONTINUE
         MVC   COMMENT,COMMENTI    ..YES, PUT IN IN FILE AREA
CONTINUE DS    0H                  FILE RECORD IS NOW SET UP
35       MVC   LOGREC,FILEREC      MOVE FILE RECORD TO LOG AREA
         MVC   LDAY,EIBDATE        MOVE DATE TO LOG AREA
         MVC   LTIME,EIBTIME       MOVE TIME TO LOG AREA
         MVC   LTERML,EIBTRMID     MOVE TERMINAL-ID TO LOG AREA
36       EXEC CICS WRITEQ TD QUEUE('LOGA') FROM(LOGA) LENGTH(92)
         CLC   EIBTRNID,=CL(L'EIBTRNID)'AUPD'  UPDATE REQUIRED?
         BNE   ADDWRITE            ..NO, SO BRANCH
37       EXEC CICS REWRITE DATASET('FILEA') FROM(FILEA)
         BR    RETREG              FINISHED, SO RETURN
ADDWRITE DS    0H                  ADD FUNCTION REQUIRED
38       EXEC CICS WRITE DATASET('FILEA') FROM(FILEA)                  *
               RIDFLD(NUMB-FILEDS(COMPTR))
         BR    RETREG              FINISHED, SO RETURN
DATAERR  DS    0H                  GENERAL ROUTINES
39       MVI   NAMEA,DFHBMFSE      PRESERVE CONTENTS OF SCREEN
         MVI   ADDRA,DFHBMFSE      BY SETTING THE MODIFIED DATA TAG
         MVI   PHONEA,DFHBMFSE     ON THE FIELDS ON THE SCREEN.
         MVI   DATEA,DFHBMFSE
         MVI   AMOUNTA,DFHUNNUM    NUMERIC AND MODIFIED FLAGS
         MVI   COMMENTA,DFHBMFSE
         MVI   MSG3A,DFHBMASB      BRIGHTEN ERROR MESSAGE
         MVI   MSG1A,DFHPROTN      DARK AND PROTECTED ATTRIBUTE
40       MVC   MSG3O,=CL(L'MSG3O)'DATA ERROR - CORRECT AND PRESS ENTER'
41       EXEC CICS SEND MAP('DFH$AGB') DATAONLY
         CLC   EIBTRNID,=CL(L'EIBTRNID)'AUPD'  UPDATE REQUIRED?
         BE    UPDTERR             ..YES, SO BRANCH
42       MVC   COMLEN,=H'7'       ..NO,SET UP COMLEN
         B     CICSCONT            GO RETURN CONTROL TO CICS
UPDTERR  DS    0H
         MVC   COMLEN,=H'80'       UPDATE, SET UP REQUIRED COMLEN
         B     CICSCONT
NOTMODF  DS    0H                  SCREEN NOT CHANGED
43       MVC   MESSAGES,=CL(L'MESSAGES)'RECORD NOT MODIFIED'  MESSAGE
         B     AMNU                COMPLETE, GO FINISH
DUPREC   DS    0H                  DUPLICATE RECORD
         MVC   MESSAGES,=CL(L'MESSAGES)'DUPLICATE RECORD'  MESSAGE
         B     AMNU                COMPLETE, GO FINISH
```

Source listing for DFH$AALL (continued)

```
BADLENG  DS    0H
         MVC   MESSAGES,=CL(L'MESSAGES)'PLEASE ENTER AN ACCOUNT NUMBER'
         B     AMNU
BADCHARS DS    0H
         MVC   MESSAGES,=CL(L'MESSAGES)'ACCOUNT NUMBER MUST BE NUMERIC'
         B     AMNU
NOTFOUND DS    0H                    RECORD NOT FOUND
         MVC  MESSAGES,=CL(L'MESSAGES)'INVALID NUMBER - PLEASE REENTER'
         B     AMNU                  COMPLETE, GO FINISH
MFAIL    DS    0H
         MVC   MESSAGES,=CL(L'MESSAGES)'PRESS CLEAR TO EXIT'
         B     AMNU
ERRORS   DS    0H                    GENERAL ERROR ROUTINE
44       EXEC CICS DUMP DUMPCODE('ERRS')
         MVC   MESSAGES,=CL(L'MESSAGES)'TRANSACTION TERMINATED'
         B     AMNU                  COMPLETE, GO FINISH
HEXZERO  DC    X'00'                 CONSTANT FOR COMPARISONS
TAB      DC    256X'FF'              TRANSLATE TABLE FOR NAME
         ORG   TAB+C' '              BLANK
         DC    X'00'
         ORG   TAB+C'.'              CHAR '.'
         DC    X'00'
         ORG   TAB+C'-'              CHAR '-'
         DC    X'00'
         ORG   TAB+C''''             CHAR QUOTE
         DC    X'00'
         ORG   TAB+C'A'              CHARS 'A' - 'I'
         DC    9X'00'
         ORG   TAB+C'J'              CHARS 'J' - 'R'
         DC    9X'00'
         ORG   TAB+C'S'              CHARS 'S' - 'Z'
         DC    8X'00'
         ORG
CHEKTAB  DC    256X'FF'              TRANSLATE TABLE FOR AMOUNT
         ORG   CHEKTAB+C'¢'          ALTERNATE CURRENCY
         DC    X'00'
         ORG   CHEKTAB+C'.'          ALLOW FULL STOP
         DC    X'00'
         ORG   CHEKTAB+C'$'          ALLOW CURRENCY
         DC    X'00'
         ORG   CHEKTAB+C'0'          NUMBERS 0-9
         DC    10X'00'
         ORG
         END
```

Program notes

1. The possible invoking transaction identifiers are tested.
2. The length of the COMMAREA is tested. If not zero then this is the validation stage of an add or update.
3. The program exits are set up.
4. The menu map DFH$AGA is received. The account number, if entered, is mapped into KEYI in the dsect for DFH$AGA.
5. The account number is validated and saved.
6. If the program is invoked by AADD, a title and command message are moved to the map area. The record key is moved to the map area and saved in COMMAREA. The amount field has the attribute byte set to numeric.
7. The add screen is displayed and the program terminates to await a reply from the terminal.
8. For an inquiry or update the exit for the record-not-found condition is set up.
9. The file control READ command reads the file record into the file area.
10. If the program is invoked by AINQ, a title and command message are moved to the map area. The file record fields are moved to the map area by a subroutine.
11. All field attributes are set to protected.

12. The inquiry screen is displayed and the program terminates. The TRANSID of AMNU causes the operator instruction program to be invoked when the next response is received from the terminal.

13. If the program is invoked by AUPD, a title and command message are moved to the map area.

14. The length of the COMMAREA to be returned is set up.

15. Data is moved to the map dsect and displayed. Control is returned to CICS.

16. Control is passed here when a test at OKTRANID finds that a COMMAREA has been received. This part of the program maps in data for an add or update request, performs validation and updates FILEA.

17. The error exits are set up.

18. The RECEIVE MAP command maps in the variables from the screen.

19. If this is an update request a file control READ UPDATE command reads the existing record using the number stored in COMMAREA by the last invocation of this program.

20. If the current file record is not the same as the one saved in COMMAREA then another user has updated the record. A warning message is displayed, with fields from the record read from FILEA, for reentry of the updates.

21. A subroutine checks that updates are valid.

22. The update flag is set in the modified record and the record is updated on FILEA.

23. The message 'RECORD UPDATED' is moved to the message area for display on the operator instruction screen.

24. If this is an add request a subroutine is called to check that the entered data is valid.

25. The add flag is set in the new record and the record is written to FILEA.

26. The message 'RECORD ADDED' is moved to the message area for display on the operator instruction screen.

27. After the FILE ADD or FILE UPDATE screen has been displayed the program branches here to return to CICS awaiting a response from the terminal. The RETURN gives CICS the transaction identifier for the next transaction at this terminal together with a COMMAREA containing all information that the program needs to continue the update. The COMMAREA is passed to the next invocation of this program, see note 2 above.

28. This code gets control when an add or update is complete. An information or error message is in MESSAGES. The operator instruction map area is cleared. The message is moved to the map area and highlighted.

29. The operator instruction map DFH$AGA is displayed on an erased screen.

30. The program terminates by returning to CICS. No transaction identifier or COMMAREA is specified.

31. This subroutine moves fields from the FILEA record to the map dsect for DFH$AGB.

32. MAPSEND sends the map DFH$AGB to the screen specifying that the screen is to be erased before the map is displayed.

33. Any required editing steps should be inserted here. A suitable form of editing should be used to ensure valid records are placed on the file.

34. FILESTUP creates or updates the account record and writes it to FILEA. Any field which has been entered is moved to the account record.

35. The record fields, the date, the time, and the termid are moved to the update log record area.

36. The record is written to the update log which is a transient data queue.

37. For an update request the updated account record is rewritten to FILEA.

38. For an add request the new account record is written to the file.

39. When a data error is detected the screen is redisplayed for errors to be corrected. The modified data tag is set on for all the data fields so that all data is received at the next RECEIVE MAP.

40. An error message is moved to the map area.

41. The contents of map DFH$AGB are sent to the screen. The constant information on the screen is not refreshed as a result of the use of the DATAONLY option.

42. The size of the COMMAREA is set to 7 for an add request or to 80 for an update request. Code at CICSCONT returns to CICS passing this value in the LENGTH operand.

43. These short error routines set up an error message in MESSAGES and branch to AMNU to display the message in the operator instruction menu DFH$AGA.

44. If a CICS command fails with the ERROR condition or if an unknown transaction identifier is used to invoke this program, a dump is taken and the message 'TRANSACTION TERMINATED' is moved to MESSAGES for display on the operator instruction screen.

Browse sample program (ASM)

Description

The browse program sequentially retrieves a page or set of records for display, starting at a point in a file specified by the terminal operator.

To start a browse, type ABRW and an account number into the menu and press the enter key. If you omit the account number browsing begins at the start of the file. Pressing PF1 or typing F retrieves the next page or pages forward. To re-examine the previous records displayed, press PF2 or type B. This lets you page backward.

The browse program uses READNEXT to forward page to the end of the file and READPREV to backward page to the start of the file.

Source listing for DFH$ABRW

```
         TITLE 'DFH$ABRW - CICS/MVS SAMPLE FILEA BROWSE - ASSEMBLER'
DFHEISTG DSECT
HEXZERO  DS    X'00'                        CONSTANT FOR CLEARING MAPS
CURROP   DS    CL1                          CURRENT OPERATION (F/B)
LASTOP   DS    CL1                          LAST OPERATION (F/B)
STATUS   DS    CL1        FILE STATUS - HI OR LO END OR NORMAL (H/L/N)
FLAGS    DS    X                            FLAGS...
PFIRST   EQU   X'80'                         FIRST BROWSE IS BACKWARD.
KEYS     DS    0CL18
RID      DS    CL6                          DATA-AREA FOR RIDFLD
RIDB     DS    CL6                          TO BUILD PREV BACK PAGE
RIDF     DS    CL6                          TO BUILD NEXT FWD PAGE
MESSAGES DS    CL80
*
         COPY  DFHBMSCA                     STANDARD BMS ATTRIBUTES
         COPY  DFH$AFIL                     FILEA RECORD DESCRIPTION
         COPY  DFH$AGA                      'GENERAL MENU' MAP
         COPY  DFH$AGC                      'BROWSE FILEA' MAP
*
DFH$ABRW CSECT
 1       MVI   KEYS,X'F0'                            '0' INTO TOP BYTE
         MVC   KEYS+1(L'KEYS-1),KEYS                 SET KEYS TO ZERO
         MVI   MESSAGES,X'40'                        ' ' INTO TOP BYTE
         MVC   MESSAGES+1(L'MESSAGES-1),MESSAGES      CLEAR MESSAGES
*
 2       EXEC CICS HANDLE AID                                          *
               CLEAR(SMSG)                                             *
               PF1(PAGEF)                                              *
               PF2(PAGEB)
*
 3       EXEC CICS HANDLE CONDITION                                    *
               ERROR(ERRORS)                                           *
               MAPFAIL(SMSG)                                           *
               NOTFND(NOTFOUND)
*
 4       EXEC CICS RECEIVE MAP('DFH$AGA')
***********************************************************************
*                                      SIMPLE CHECKS OF INPUT DATA    *
***********************************************************************
 5       CLC   KEYL,=H'0'          WAS ACCOUNT NUMBER OMITTED?
         BE    NOACCNUM             YES - FRONT OF FILE BY DEFAULT.
*
 6       TRT   KEYI,CHEKTAB        CHECK ACCOUNT NUMBER IS NUMERIC,
         BNZ   BADCHARS             IT ISN'T - GO DISPLAY MESSAGE.
         MVC   RID,KEYI
         CLC   RID,=C'999999'      IF ACCOUNT NUMBER IS MAXIMUM
         BNE   SETRID               SET RECORD KEY HIGH TO
         MVC   RID,=6X'FF'           BROWSE BACKWARD FIRST TIME.
SETRID   MVC   RIDF,RID
         MVC   RIDB,RID
         B     BRWSNOW
*
BADCHARS MVC   MESSAGES,=CL(L'MESSAGES)'ACCOUNT NUMBER MUST BE NUMERIC'
         B     AMNU
*
```

Source listing for DFH$ABRW (continued)

```
NOACCNUM MVC   RID,=C'000000'
         MVC   RIDF,=C'000000'                       0 DIGITS ENTERED
         B     BRWSNOW
************************************************************************
*                                      ESTABLISH START POINT           *
************************************************************************
BRWSNOW  DS    0H
         MVI   STATUS,C'N'                  SET FILE STATUS NORMAL
 7       EXEC CICS STARTBR DATASET('FILEA') RIDFLD(RID)
         CLC   RID,=6X'FF'
         BNE   PAGEF
*                                  TREAT AS HI-EOF,
         OI    FLAGS,PFIRST                 SET FIRST BROWSE BACK FLAG.
         MVI   STATUS,C'H'               AND
         B     PAGEB               PAGE BACKWARDS
************************************************************************
*                                      BUILD NEXT FORWARD PAGE         *
************************************************************************
PAGEF    DS    0H
         MVI   CURROP,C'F'                  IN CASE PF1 WAS USED
 8       EXEC CICS HANDLE CONDITION                                    *
               ENDFILE(TOOHIGH)
*                                           CLEAR MAP
         LA    4,1                          SET COUNTER TO 1
         LA    6,DFH$AGCO                   R6->START OF MAP DFH$AGC
         LA    7,(DFH$AGCE-DFH$AGCO)        R7-> LENGTH OF DFH$AGC
         LA    8,HEXZERO                    R8-> X'00'
         LA    9,L'HEXZERO                  R9-> LENGTH OF HEXZERO
         ICM   9,B'100',HEXZERO             X'00' INTO TOP BYTE OF R9
         MVCL  6,8                          MOVE X'00' INTO DFH$AGCO
*
         MVC   RID,RIDF                     RIDF->NEXT FORWARD PAGE
*
NEXTLINE DS    0H
 9       EXEC CICS READNEXT                                            *
               INTO(FILEA)                                             *
               DATASET('FILEA')                                        *
               RIDFLD(RID)
*
10       CH    4,=H'1'                      FIRST LINE?
         BNE   NEXT2                        ..NO, GO TEST FOR 2ND LINE
         MVC   NUMBER10,NUMB                MOVE NUMBER TO MAP AREA
         MVC   NAME10,NAME                  MOVE NAME TO MAP AREA
         MVC   AMOUNT10,AMOUNT              MOVE AMOUNT TO MAP AREA
         MVC   RIDB,RID                     RIDB->EXISTING A/C NO.
         B     NEXTCONT                     GOTO NEXTCONT
*
NEXT2    DS    0H
11       CH    4,=H'2'                      SECOND LINE?
         BNE   NEXT3                        ..NO, TEST FOR THIRD LINE
         MVC   NUMBER20,NUMB                MOVE NUMBER TO MAP AREA
         MVC   NAME20,NAME                  MOVE NAME TO MAP AREA
         MVC   AMOUNT20,AMOUNT              MOVE AMOUNT TO MAP AREA
         B     NEXTCONT                     GOTO NEXTCONT
```

Source listing for DFH$ABRW (continued)

```
*
NEXT3    DS    0H
         CH    4,=H'3'                     THIRD LINE?
         BNE   NEXT4                       ..NO, TEST FOR FOURTH LINE
         MVC   NUMBER3O,NUMB               MOVE NUMBER TO MAP AREA
         MVC   NAME3O,NAME                 MOVE NAME TO MAP AREA
         MVC   AMOUNT3O,AMOUNT             MOVE AMOUNT TO MAP AREA
         B     NEXTCONT                    GOTO NEXTCONT
*
NEXT4    DS    0H
         CH    4,=H'4'                     FOURTH LINE?
         BNE   NEXTCONT                    ..NO, GOTO NEXTCONT
         MVC   NUMBER4O,NUMB               MOVE NUMBER TO MAP AREA
         MVC   NAME4O,NAME                 MOVE NAME TO MAP AREA
         MVC   AMOUNT4O,AMOUNT             MOVE AMOUNT TO MAP AREA
*
NEXTCONT DS    0H
         LA    4,1(,4)                     INCREMENT COUNT
         CH    4,=H'5'                     FINISHED?
         BNE   NEXTLINE                    ..NO, GO BUILD NEXTLINE
*
         MVC   RIDF,RID                     RIDF->NEXT FORWARD PAGE
12       EXEC CICS SEND MAP('DFH$AGC') ERASE
         B     RECEIVE
***********************************************************************
*                                      BUILD PREVIOUS BACK PAGE      *
***********************************************************************
PAGEB    DS    0H
         MVI   CURROP,C'B'                 IN CASE PF2 WAS USED
         EXEC CICS HANDLE CONDITION                                    *
               ENDFILE(TOOLOW)
*                                          CLEAR MAP
         LA    4,1                         SET COUNTER TO 1
         LA    6,DFH$AGCO                  R6->START OF MAP DFH$AGC
         LA    7,(DFH$AGCE-DFH$AGCO)       R7-> LENGTH OF DFH$AGC
         LA    8,HEXZERO                   R8-> X'00'
         LA    9,L'HEXZERO                 R9-> LENGTH OF HEXZERO
         ICM   9,B'100',HEXZERO            X'00' INTO TOP BYTE OF R9
         MVCL  6,8                         MOVE X'00' INTO DFH$AGCO
*
         MVC   RID,RIDB                    RIDB->PREVIOUS BACK PAGE
         MVC   RIDF,RIDB                   RIDF->NEXT FORWARD PAGE
*
         CLI   LASTOP,C'B'                 SWITCHING DIRECTION?
         BE    PREVLINE                    NO. - NO SPECIAL ACTION
*                                          YES - DO EXTRA READPREV
         CLI   STATUS,C'H'                 - UNLESS AT HI EOF
         BE    PREVLINE
PREVXTRA DS    0H
         EXEC CICS READPREV                                            *
               INTO(FILEA)                                             *
               DATASET('FILEA')                                        *
               RIDFLD(RID)
*
```

Source listing for DFH$ABRW (continued)

```
PREVLINE DS    0H
13       EXEC CICS READPREV                                            *
               INTO(FILEA)                                             *
               DATASET('FILEA')                                        *
               RIDFLD(RID)
*                                      PUT FIELDS IN ASCENDING ORDER
         CH    4,=H'4'                      FOURTH LINE?
         BNE   PREV2                        ..NO, GO TEST FOR 3RD LINE
         MVC   NUMBER10,NUMB                MOVE NUMBER TO MAP AREA
         MVC   NAME10,NAME                  MOVE NAME TO MAP AREA
         MVC   AMOUNT10,AMOUNT              MOVE AMOUNT TO MAP AREA
         B     PREVCONT                     GOTO PREVCONT
*
PREV2    DS    0H
         CH    4,=H'3'                      THIRD LINE?
         BNE   PREV3                        ..NO, TEST FOR 2ND LINE
         MVC   NUMBER20,NUMB                MOVE NUMBER TO MAP AREA
         MVC   NAME20,NAME                  MOVE NAME TO MAP AREA
         MVC   AMOUNT20,AMOUNT              MOVE AMOUNT TO MAP AREA
         B     PREVCONT                     GOTO PREVCONT
*
PREV3    DS    0H
         CH    4,=H'2'                      SECOND LINE?
         BNE   PREV4                        ..NO, TEST FOR FIRST LINE
         MVC   NUMBER30,NUMB                MOVE NUMBER TO MAP AREA
         MVC   NAME30,NAME                  MOVE NAME TO MAP AREA
         MVC   AMOUNT30,AMOUNT              MOVE AMOUNT TO MAP AREA
         B     PREVCONT                     GOTO PREVCONT
*
PREV4    DS    0H
         CH    4,=H'1'                      FIRST LINE?
         BNE   PREVCONT                     ..NO, PREVCONT
         MVC   NUMBER40,NUMB                MOVE NUMBER TO MAP AREA
         MVC   NAME40,NAME                  MOVE NAME TO MAP AREA
         MVC   AMOUNT40,AMOUNT              MOVE AMOUNT TO MAP AREA
*
PREVCONT DS    0H
         LA    4,1(,4)                      INCREMENT COUNT
         CH    4,=H'5'                      FINISHED?
         BNE   PREVLINE                     ..NO, GO BUILD NEXT LINE
*
         MVC   RIDB,RID                     RIDB->NEXT FORWARD PAGE
         TM    FLAGS,PFIRST                 DID WE START AT HI-EOF?
         BO    HIGHMSG                       YES TELL HIM.
         EXEC CICS SEND MAP('DFH$AGC') ERASE
************************************************************************
*                                      RECEIVE NEXT PAGING REQUEST     *
************************************************************************
RECEIVE  DS    0H
         NI    FLAGS,X'FF'-PFIRST           SET OF FIRST TIME FLAG.
         MVC   LASTOP,CURROP                REMEMBER LAST OPERATION
14       EXEC CICS RECEIVE MAP('DFH$AGC')
         CLI   DIRI,C'F'                    PAGE FORWARD REQUIRED?
         BE    PAGEF                        ..YES, GO TO PAGEF ROUTINE
         CLI   DIRI,C'B'                    PAGE BACK REQUIRED?
```

Source listing for DFH$ABRW (continued)

```
         BE    PAGEB                        ..YES, GO TO PAGEB ROUTINE
         EXEC CICS SEND CONTROL FREEKB FRSET. IGNORE - RESET KEYBOARD.
         B     RECEIVE
***********************************************************************
*                                  HANDLE END OF FILE CONDITIONS      *
***********************************************************************
TOOHIGH  DS    0H
15       MVI   STATUS,C'H'                  SET STATUS 'HI END'
         MVC   RIDF,RID
         MVC   RIDB,RID
         MVI   DIRO,X'40'
HIGHMSG  MVC   MSG1O,=CL(L'MSG1O)'HI END OF FILE'
         MVI   MSG1A,DFHBMASB                                  MSG=BRT
         EXEC CICS SEND MAP('DFH$AGC') ERASE
         B     RECEIVE
*
TOOLOW   DS    0H
16       MVI   STATUS,C'L'                  SET STATUS 'LO END'
         MVC   RIDF,=C'000000'
         MVC   RIDB,=C'000000'
         MVI   DIRO,X'40'
         MVI   MSG2A,DFHBMASB                                  MSG=BRT
         MVC   MSG2O,=CL(L'MSG2O)'LO END OF FILE'
         EXEC CICS SEND MAP('DFH$AGC') ERASE
         B     RECEIVE
***********************************************************************
*                                  HANDLE GENERAL CONDITIONS          *
***********************************************************************
NOTFOUND DS    0H
17       MVC   MESSAGES,=CL(L'MESSAGES)'END OF FILE - PLEASE RESTART '
         B     AMNU
*
SMSG     DS    0H
18       MVC   MESSAGES,=CL(L'MESSAGES)'PRESS CLEAR TO EXIT'
         B     AMNU
*
ERRORS   DS    0H
19       EXEC CICS DUMP DUMPCODE('ERRS')
         MVC   MESSAGES,=CL(L'MESSAGES)'TRANSACTION TERMINATED'
***********************************************************************
*                                  DISPLAY GENERAL MENU THEN EXIT     *
***********************************************************************
AMNU     DS    0H
20       XC    DFH$AGAO(DFH$AGAE-DFH$AGAO),DFH$AGAO   CLEAR MAP A
         MVI   MSGA,DFHBMASB                          BRIGHTEN MESSAGE
         MVC   MSGO,MESSAGES                          MOVE MSGS TO MAP
         EXEC CICS SEND MAP('DFH$AGA') ERASE
21       EXEC CICS RETURN
***********************************************************************
*                                  DEFINE THE 256 BYTE TRANSLATE TABLE*
***********************************************************************
*                                  FOR LOCATING NON-NUMERIC DIGITS BY
*                                  MEANS OF THE "TRT" INSTRUCTION
CHEKTAB DC 256X'FF'
        ORG CHEKTAB+X'F0'
        DC 10X'00'
        ORG END
```

Program notes

1. Work areas are initialized to begin the browse.
2. The exits for CLEAR, PF1 and PF2 are set up.
3. The error exits are set up.
4. This command maps in the account number from the operator instruction screen.
5. If no account number is entered browsing begins at the start of the file.
6. If the format of the account number is valid the number is used to set the program's browse pointers, otherwise an error message is displayed on the operator instruction menu. Entering the maximum value (999999) for the account number begins a backward browse from the end of the file.
7. The STARTBR command establishes the browse starting point.
8. The forward browse end of file exit is set up.
9. The READNEXT reads the first record, and subsequently the next record, into the file area.
10. The account number, name, and amount are moved to the first line of the browse map area.
11. The same basic commands are repeated to read and set up the next three lines. The same file area is used for each read.
12. The screen is erased and the full page is displayed at the terminal.
13. Backward browsing uses the READPREV command to read the previous record and stores records in the map area starting at the bottom line. Note the need for an extra READPREV when changing from forward to backward browsing.
14. When the RECEIVE command executes control will go to one of the HANDLE AID exits (see note 2) if CLEAR, PF1 or PF2 is pressed. The program explicitly tests for F or B if no exit is taken. Any other terminal response is ignored.
15. If the end of file is reached any records read to that point are displayed together with a highlighted message 'HI END OF FILE'.
16. If the start of file is reached on a READPREV (backward browse) then the ENDFILE condition occurs and TOOLOW gets control. Any records read up to that point are displayed, together with a highlighted message 'LO END OF FILE'.
17. If the NOTFND condition occurs at the start browse (note 7) the message 'END OF FILE - PLEASE RESTART' is moved to MESSAGES for display on the operator instruction screen.
18. If the CLEAR key is pressed or when a MAPFAIL occurs a message 'PRESS CLEAR TO EXIT' is moved to MESSAGES for display on the operator instruction screen.
19. In some error situations a dump is taken and the message 'TRANSACTION TERMINATED' is moved to MESSAGES for display on the operator instruction screen.
20. This code displays the operator instruction menu with a message which has been stored in MESSAGES.
21. The program terminates by returning to CICS.

Order entry sample program (ASM)

Description

The order entry sample application program provides a data entry facility for customer orders for parts from a warehouse. Orders are recorded on a transient data queue which 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 AORD onto a blank screen and press ENTER. The order entry program displays the map DFH$AGK on the screen requesting the operator to enter order details, that is, customer number, part number, and the quantity of that part required. The customer number must be valid, that is, it must exist on FILEA. 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. A definition of the transient data queue is included in the sample destination control table listed in the *CICS/MVS Installation Guide*.

The trigger level may be changed using the CEMT command, as follows:

```
CEMT SET QUEUE(L860) TRIGGER(n)
```

where n is the destination trigger level (any integer from 0 through 32767).

Source listing for DFH$AREN

```
         TITLE 'DFH$AREN - CICS/MVS SAMPLE FILEA ORDER ENTRY - ASSEMBLE*
               R'
DFHEISTG DSECT
         COPY  DFH$AGK                                MAP DEFINITION
         COPY  DFH$AL86                               TD Q REC DESCRIPT
         COPY  DFH$AFIL                               FILE OF ACCOUNTS
         COPY  DFHBMSCA                               STD BMS ATTRIBUTE
*
FLAGS    DS    1B                                     ERROR FLAGS
FLERR    EQU   X'40'                                  FLERR-INPUTCHECKS
*
DFH$AREN CSECT
 1       EXEC CICS HANDLE AID CLEAR(ENDA)
*
 2       EXEC CICS HANDLE CONDITION MAPFAIL(MAPFAIL)                   *
               NOTFND(NOTFOUND) ERROR(ERRORS)
*
*                                                     CLEAR MAP VARS
         XC    DFH$AGKO(DFH$AGKE-DFH$AGKO),DFH$AGKO
*
*                                                     ERASE+DISPLAY MAP
 3       EXEC CICS SEND MAP('DFH$AGK') ERASE
***********************************************************************
*                                      PROCESS THE INPUTTED FIELDS    *
***********************************************************************
RECEIVE  DS    0H
*                                                     RECEIVE INPUTDATA
 4       EXEC CICS RECEIVE MAP('DFH$AGK')
         NI    FLAGS,X'FF'-FLERR                      SET FLERR TO 0
*
         MVI   CUSTNOA,DFHUNNUM                       SET MDT=1 IN CASE
         MVI   PARTNOA,DFHUNNUM                       FIELDS NEED TO BE
         MVI   QUANTA,DFHUNNUM                        RE-ENTERED
*
*                                                     CHECK FOR NUMERIC
 5       TRT   CUSTNOI,CHEKTAB
         BZ    TSTPART
         MVI   CUSTNOA,DFHUNINT              ATTR=BRI+UNPROT'D+NUMERIC
         OI    FLAGS,FLERR                            SET FLERR TO 1
*
TSTPART  TRT   PARTNOI,CHEKTAB
         BZ    TSTQUANT
         MVI   PARTNOA,DFHUNINT              ATTR=BRI+UNPROT'D+NUMERIC
         OI    FLAGS,FLERR                            SET FLERR TO 1
*
TSTQUANT TRT   QUANTI,CHEKTAB
         BZ    CHKFLERR
         MVI   QUANTA,DFHUNINT              ATTR=BRI+UNPROT'D+NUMERIC
         OI    FLAGS,FLERR                            SET FLERR TO 1
***********************************************************************
*                                      SIMPLE VALIDATION OF INPUT DATA*
***********************************************************************
CHKFLERR TM    FLAGS,FLERR
         BZ    QBUILD
 6       MVI   MSG2A,DFHBMASB
*                                                     ERASE+DISPLAY MAP
```

Source listing for DFH$AREN (continued)

```
         EXEC CICS SEND MAP('DFH$AGK') ERASE
         B     RECEIVE
***********************************************************************
*                                      CHECK CUSTOMER NUMBER EXISTS   *
***********************************************************************
 7
QBUILD   EXEC CICS READ DATASET('FILEA') INTO(FILEA) RIDFLD(CUSTNOI)
 8       MVC   CUSTNO,CUSTNOI
         MVC   PARTNO,PARTNOI
         MVC   QUANTITY,QUANTI
         MVC   TERMID,EIBTRMID
***********************************************************************
*                                      WRITE VALID ORDER TO TD QUEUE  *
***********************************************************************
 9       EXEC CICS WRITEQ TD QUEUE('L860') FROM(L860) LENGTH(22)
10       EXEC CICS SEND MAP('DFH$AGK') MAPONLY ERASE
         B     RECEIVE                                GET MORE INPUT
***********************************************************************
*                                      HANDLE ERRORS AND RESTART      *
***********************************************************************
*
NOTFOUND DS    0H                      INVALID CUSTOMER ACCOUNT NO.
11       MVI   CUSTNOA,DFHUNINT              ATTR=BRI+UNPROT'D+NUMERIC
         MVI   PARTNOA,DFHUNNUM                       MDT=1 TO PRESERVE
         MVI   QUANTA,DFHUNNUM                        ....THESE FIELDS
         MVI   MSG1A,DFHBMASB                         ERROR MSG=BRIGHT
         EXEC CICS SEND MAP('DFH$AGK')
         B     RECEIVE                                GET MORE INPUT
*
MAPFAIL  DS    0H                      NO DATA ENTERED IN INPUT FIELDS
12       XC    DFH$AGKO(DFH$AGKE-DFH$AGKO),DFH$AGKO   CLEAR MAP
         MVI   MSG2A,DFHBMASB                         ERROR MSG=BRIGHT
         EXEC CICS SEND MAP('DFH$AGK')
         B     RECEIVE                                GET MORE INPUT
***********************************************************************
*                                      EXIT PROGRAM                   *
***********************************************************************
ERRORS   DS    0H                      GENERAL ERROR CONDITIONS
13       MVI   MSG2A,DFHBMASB                         ERROR MSG=BRIGHT
         MVC   MSG2O,=C'TRANSACTION TERMINATED'
         EXEC CICS SEND MAP('DFH$AGK')
         EXEC CICS DUMP DUMPCODE('ERRS')
         B     EXIT                                   QUIT PROGRAM
ENDA     DS    0H
14       EXEC CICS SEND TEXT FROM (PRESMSG) ERASE
*                                                     SET INPUT INH OFF
         EXEC CICS SEND CONTROL FREEKB
EXIT     EXEC CICS RETURN
PRESMSG  DC    CL20'PROCESSING COMPLETED'
***********************************************************************
*                                 DEFINE THE 256 BYTE TRANSLATE TABLE *
*                                 FOR LOCATING NON-NUMERIC DIGITS BY  *
*                                 MEANS OF THE "TRT" INSTRUCTION      *
***********************************************************************
CHEKTAB  DC     256X'FF'
         ORG    CHEKTAB+X'F0'
         DC     10X'00'
         ORG
         END
```

Program notes

1. The CLEAR key exit is set up.
2. The error exits are set up.
3. The screen is erased and the order entry map is displayed at the terminal.
4. This RECEIVE MAP causes a read from the terminal and maps in the customer number, part number, and quantity. The program remains in virtual storage until the terminal response is received. Compare this technique with that used in the pseudoconversational inquiry/update sample program. If no data is received, CICS branches to the MAPFAIL exit (note 2).
5. The order details are checked, and invalid orders are redisplayed for correction. The user should add further editing steps necessary to ensure only valid orders are accepted.
6. The error message 'DATA ERROR – REENTER' is a constant in the map load module and is sent to the terminal, with any other constant information, unless DATAONLY is specified on the SEND MAP. The message is normally dark (non-display). This instruction overrides the dark attribute and the message appears in high intensity when the SEND MAP command is executed.
7. The file control READ command attempts to read the customer record from FILEA. If no record exists for the customer CICS branches to the NOTFND exit (note 2).
8. The order details are moved from the input map to the queue area.
9. The WRITEQ TD command writes the order record to a sequential file, a transient data queue.
10. The order entry map is redisplayed ready for the next order. Only the map load module is used to build the screen display, MAPONLY causes the data in the map dsect area to be ignored.
11. If there is no record for the customer on FILEA, CICS raises the NOTFND condition and branches here. The modified data tags are set on all data fields and an error message 'NUMBER NOT FOUND - REENTER' is set to display in high intensity (see note 6). The order is redisplayed for correction.
12. If no fields are entered, the MAPFAIL condition occurs. The message 'DATA ERROR-REENTER' is displayed in high intensity (see note 6).
13. If an error occurs a dump is taken, and the message 'TRANSACTION TERMINATED' is displayed in high intensity in the data error message area. The program terminates leaving the order entry screen displayed.
14. When the CLEAR key is pressed the program terminates. The message 'PROCESSING COMPLETED' is displayed on a blank screen, the keyboard is freed and control is returned to CICS.

Order entry queue print sample program (ASM)

Description

The order entry queue print sample program sends customer orders to a printer terminal at the warehouse. The order entry sample program, described earlier, records customer orders on a transient data queue which is read by this program.

The queue print transaction can be invoked in one of three ways:

- You can type the transaction identifier AORQ onto 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 COMPLETED' is displayed and your terminal is available for other work.
- One hour after you enter AORQ, the queue print transaction is automatically invoked by CICS interval control. In this case the terminal identifier, specified by the START, is L86O so the program prints the orders at the warehouse.
- The queue print transaction is "triggered" 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 will print the orders. The trigger level may be changed using the command:

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

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/update sample program. When all the orders have been processed, or if there were no orders to process, the message 'ORDER QUEUE IS EMPTY' is printed at the warehouse.

Source listing for DFH$ACOM

```
         TITLE 'DFH$ACOM - CICS/MVS SAMPLE FILEA ORDER ENTRY QUEUE PRIN*
               T - ASSEMBLER'
DFHEISTG DSECT
         COPY  DFH$AGL                 MAP
         COPY  DFH$AL86                Q RECORD
         COPY  DFH$AFIL                FILE RECORD
LOGORD   DS    0CL92                   RECORD TO BE WRITTEN ONTO LOGA
LDATE    DS    PL7
LTIME    DS    PL7
LITEM    DS    CL22
COMMNT   DS    CL11
FILLER   DS    CL51
QLENGTH  DS    1H      SIZE OF Q RECORD
DFH$ACOM CSECT
         MVC   COMMNT,=C'ORDER ENTRY'
         MVI   FILLER,X'40'
         MVC   FILLER+1(L'FILLER-1),FILLER
 1       EXEC CICS HANDLE CONDITION ERROR(ERRORS)QZERO(ENDA)
 2       CLC   EIBTRMID(4),=C'L860'    TERMID='L860'?
         BNE   TIME                    IF NOT START TRANSACTION LATER
         XC    DFH$AGLO(DFH$AGLE-DFH$AGLO),DFH$AGLO   CLEAR MAP
         MVC   QLENGTH,=H'+22' INITIALIZATION
QREAD    DS    0H
 3       EXEC CICS READQ TD INTO(L860)LENGTH(QLENGTH)QUEUE('L860')
 4       EXEC CICS READ DATASET('FILEA') INTO(FILEA) RIDFLD(CUSTNO)
 5       CLC   AMOUNT(8),=C'$0100.00' IS ORDER VALID?
         BNH   LWRITE                  IF¬>100  BRANCH AND WRITE LOG
 6       MVC   ADDRO,ADDRX             SET UP MAP
         MVC   NAMO,NAME
         MVC   PARTO,PARTNO
         MVC   NUMBO,CUSTNO
         MVC   LITEM,ITEM
         MVC   QUANTO,QUANTITY
 7       EXEC CICS SEND MAP('DFH$AGL') ERASE PRINT L80
         B     QREAD                  GET NEXT RECORD
LWRITE   DS    0H
 8       MVC   LDATE,EIBDATE           SET UP LOG RECORD
         MVC   LTIME,EIBTIME
         MVC   LITEM,ITEM
 9       EXEC CICS WRITEQ TD QUEUE('LOGA') FROM(LOGORD) LENGTH(92)
         B     QREAD                  GET NEXT RECORD
ERRORS   DS    0H
10       EXEC CICS DUMP DUMPCODE('ERRS')
         B     FIN                     BRANCH TO END
ENDA     DS    0H
         XC    DFH$AGLO(DFH$AGLE-DFH$AGLO),DFH$AGLO   CLEAR MAP
11       MVC   TITLEO,=CL(L'TITLEO)'ORDER QUEUE IS EMPTY' SET UP TITLE
         EXEC CICS SEND MAP('DFH$AGL') DATAONLY ERASE L80 PRINT
TIME     DS    0H
*                                 IF THE COMMENT DELIMITER IS REMOVED
*                                 FROM THE NEXT THREE ASSEMBLER
*                                 INSTRUCTIONS, THE APPLICATION WILL
*                                 BE RESTARTED IN AN HOUR IF THE TIME
*                                 OF DAY RIGHT NOW IS NOT LATER THAN
```

Source listing for DFH$ACOM (continued)

```
*                             1400 HRS.
*                             IF THE CODE IS LEFT UNCHANGED THE
*                             APPLICATION WILL BE RESTARTED
*                             UNCONDITIONALLY AFTER AN HOUR HAS
*                             ELAPSED
*        EXEC CICS ASKTIME
*        CP    EIBTIME,=P'0140000'  TIME AFTER 1400 HOURS?
*        BH    FIN                  ..YES, SO STOP
12       EXEC CICS START TRANSID('AORQ') INTERVAL(10000) TERMID('L860')
FIN      DS    0H
13       EXEC CICS SEND TEXT FROM (PRESMSG) ERASE
         EXEC CICS SEND CONTROL FREEKB
         EXEC CICS RETURN
PRESMSG  DC    CL20'PROCESSING COMPLETED'
         END
```

Program notes

1. The error exits are set up.
2. The termid is tested to see whether this transaction is started from a terminal or at the printer.
3. A queue item (customer order) is read into the program.
4. The file control READ command reads the record into a record area so that the amount may be checked.
5. The amount (bank balance) is tested. If it is over $100 then the order is acceptable, otherwise the order is rejected. This test is only a suggestion; a suitable form of editing should be inserted here to ensure valid orders are sent to the warehouse.
6. The order details are moved to the map area for DFH$AGL.
7. The order map is sent to the printer terminal at the warehouse.
8. The current date and time, and details of the rejected order, are moved to a log record area.
9. The WRITEQ TD command writes details of the rejected order to LOGA, a transient data queue.
10. If the ERROR condition occurs on any CICS command a dump is taken and the program terminates.
11. When the queue is empty, the message 'ORDER QUEUE IS EMPTY' is moved to the map area which is then sent to the printer terminal at the warehouse.
12. The START command starts the AORQ transaction (this program), after a one hour delay, with a terminal identifier of L86O. (The time interval could be changed, for demonstration purposes, by changing the INTERVAL value.) If the comment delimiters are removed from the three preceding statements, EIBTIME is refreshed and, if the time is before 1400 hours, the transaction is started in one hour. If the comment delimiters are not removed, the transaction is started unconditionally in one hour.
13. The message 'PROCESSING COMPLETED' is sent to the terminal associated with this invocation of AORQ, either the printer at the warehouse or the screen on which AORQ was entered. The program terminates by returning control to CICS.

Low balance report sample program (ASM)

Description

The low balance report sample program produces a report that lists all entries in the data set FILEA for which the amount is less than or equal to $50.00.

The program shows page building techniques and the use of the terminal paging facilities of BMS.

The transaction is invoked by entering AREP onto a clear screen. The program does a sequential scan through the file selecting each entry that obeys the search criterion.

The pages are built from four maps that comprise map set DFH$AGD, using the paging option so that the data is not displayed immediately but instead is stored for later retrieval. The HEADING map is inserted at the head of each page. The detail map (DFH$AGD) is written repeatedly until the overflow condition occurs. The FOOTING map is then written at the foot of the page and the HEADING map written at the top of the next page. The command to write the detail map that caused overflow is then repeated. When all the data has been written the FINAL map is written at the bottom of the last page and the transaction terminated.

The terminal operator then enters paging commands to display the data, clearing the screen before entering each paging command.

Source listing for DFH$AREP

```
         TITLE 'DFH$AREP - CICS/MVS SAMPLE FILEA LOW BALANCE INQUIRY - *
               ASSEMBLER'
DFHEISTG DSECT
KEYNUM   DS    CL6                   KEY TO FILE
TERMLENG DS    H                     MAXIMUM LENGTH OF KEYED DATA
TERMDATA DS    CL1                   INPUT AREA FOR KEYED DATA
*                                    (IN PRACTICE LENGTH OF KEYED DATA
*                                     WILL BE ZERO AS OPERATOR WILL ONLY
*                                     PRESS ENTER)
EDVAL    DS    CL3                   PAGE NUMBER EDITING FIELD
PAGEN    DS    CL2                   PAGE NUMBER FIELD
WORKREG  EQU   7
RETREG   EQU   4                     LINK REG
         COPY  DFH$AGD               OUTPUT MAP
         COPY  DFH$AFIL              FILEA'S RECORD DESCRIPTION
DFH$AREP CSECT
 1       MVC   KEYNUM(6),=C'000000' SET RECORD KEY TO ZERO
 2       EXEC CICS HANDLE CONDITION ERROR(ERRORS) OVERFLOW(OFLOW)      *
               ENDFILE(ENDFILE) LENGERR(ENDTASK)
         MVI   PAGENA,X'00'          MOVE X'00' TO ATTRIBUTE
         MVC   PAGEN,PAGE1           INITIALIZE PAGE NUMBER TO 1
 3       BAL   RETREG,MAPNUM        MOVE PAGENUMBER TO MAP AREA
 4       EXEC CICS SEND MAP('HEADING') MAPSET('DFH$AGD') ACCUM PAGING  *
               ERASE
 5       EXEC CICS STARTBR DATASET('FILEA') RIDFLD(KEYNUM)
REPEAT   DS    0H
 6       EXEC CICS READNEXT INTO(FILEA) DATASET('FILEA')               *
               RIDFLD(KEYNUM)
 7       CLC   AMOUNT,LOWLIM         COMPARE AMOUNT ON RECORD WITH LIM
         BH    REPEAT                ..OK, GREATER THAN $50, TRY NEXT
         XC    DFH$AGDO(DFH$AGDE-DFH$AGDO),DFH$AGDO CLEAR MAP
 8       MVC   AMOUNTO,AMOUNT        MOVE AMOUNT ON FILE TO MAP
         MVC   NUMBERO,NUMB          MOVE ACOUNT NUMBER TO MAP
         MVC   NAMEO,NAME            MOVE NAME TO MAP
 9       EXEC CICS SEND MAP('DFH$AGD') MAPSET('DFH$AGD') ACCUM PAGING
         B     REPEAT                GO BUILD NEXT MAP
***********************************************************************
*    END ROUTINE  AND GENERAL ROUTINES                                *
***********************************************************************
MAPNUM   DS    0H                    ROUTINE PUTS PAGE NUM IN CHAR FORM
         UNPK  EDVAL,PAGEN
         OI    EDVAL+L'EDVAL-1,X'F0' ZERO FILL PAGE NUMBER
         MVC   PAGENO,EDVAL          MOVE PAGE NUMBER TO OUTPUT MAP
         BR    RETREG                RETURN
```

Source listing for DFH$AREP (continued)

```
ENDFILE  DS    0H                  END OF FILE CONDITION RAISED
10       EXEC CICS SEND MAP ('FINAL') MAPSET ('DFH$AGD') MAPONLY      *
               ACCUM PAGING
11       EXEC CICS SEND PAGE
12       EXEC CICS SEND TEXT FROM (OPINSTR) ERASE
13       EXEC CICS ENDBR DATASET('FILEA')
*                                  A RECEIVE COMMAND IS ISSUED TO GIVE
*                                  THE TERMINAL OPERATOR A CHANCE TO
*                                  READ THE PROMPTING MESSAGE.
*
*                                  THE TRANSACTION WILL TERMINATE WHEN
*                                  THE OPERATOR PRESSES THE ENTER KEY.
*
*                                  PAGING COMMANDS CAN THEN BE ISSUED.
*
*                                  NO HARM IS DONE IF THE OPERATOR
*                                  TYPES IN DATA BEFORE PRESSING THE
*                                  ENTER KEY.
         LA   WORKREG,1
         STH  WORKREG,TERMLENG
14       EXEC CICS RECEIVE INTO(TERMDATA) LENGTH(TERMLENG)
ENDTASK  EQU *
15       EXEC CICS RETURN
ERRORS   DS    0H
16       EXEC CICS HANDLE CONDITION ERROR
         EXEC CICS PURGE MESSAGE
         EXEC CICS ABEND ABCODE('ERRS')
OFLOW    DS    0H                  PAGE BUILT HERE
17       EXEC CICS SEND MAP('FOOTING') MAPSET('DFH$AGD')               *
               MAPONLY ACCUM PAGING ERASE
         AP    PAGEN,=P'1'         INCREMENT PAGE COUNT
         MVI   PAGENA,X'00'        MOVE X'00' INTO ATTRIBUTE
         BAL   RETREG,MAPNUM       GO SET UP PAGE NUMBER ON MAP
18       EXEC CICS SEND MAP('HEADING') MAPSET('DFH$AGD') ACCUM PAGING  *
               ERASE
19       EXEC CICS SEND MAP('DFH$AGD') MAPSET('DFH$AGD') ACCUM PAGING
         B     REPEAT
PAGE1    DC    PL2'1'              INITIAL PAGE NUM
LOWLIM   DC    CL8'$0050.00'       LOWER LIMIT FOR OK AMOUNT
OPINSTR  DC    CL52'PRESS THE ENTER KEY AND FOLLOW WITH PAGING COMMANDS*
               '                   OPERATOR INSTRUCTION
         END
```

Program notes

1. The initial key value is set up for the START BROWSE command.
2. The program exits are set up.
3. A page number of 1 is moved to the heading map.
4. This BMS command sets up the heading in the page build operation. BMS builds the pages in temporary storage.
5. The STARTBR command sets up the file browse to begin at the first record with a key equal to or greater than the RIDFLD, in this case the first record on file.
6. This command reads the next customer record from FILEA.
7. The search criterion for creating the report is that the customer has a bank balance which is $50 or less.
8. Fields are moved from the selected customer record to the map area for the detail line.
9. The customer detail map is set up for subsequent paging.
10. When the ENDFILE condition is raised, the last map is sent to BMS.
11. The SEND PAGE command makes all the pages of the report available for paging, at the terminal, when the current transaction terminates.
12. A message is sent to the terminal. This message will be displayed before the pages of the low balance report.
13. The file browse is terminated.

14. This RECEIVE MAP command reads from the terminal and allows the terminal operator to read the prompting message before the first page of the report is displayed.

15. The program ends, the first page of the report will now be displayed.

16. If the ERROR condition occurs on a CICS command this routine gains control. Handling of the ERROR condition is suppressed, any data sent to BMS so far is purged and the program terminates abnormally with a transaction dump.

17. If the OVERFLOW condition occurs, when a detail line is sent to BMS, CICS branches here. This routine completes the current page and starts the next one. This BMS command sets up the footing for the current page.

18. This BMS command sets up the heading for the next page.

19. This BMS command sends the detail line which caused the OVERFLOW condition.

Maps and screen layouts for ASM sample programs

The preceding sample programs assume that the following map sets have been cataloged with names the same as the map names.

The names of the source maps are all of the form DFH$AMx, whereas output generated by the assembly of maps is in the form DFH$AGx. Use different names for the map source and the generated dsect only if you want to store both in the same source library.

DFH$AGA map definition

```
         TITLE 'FILEA - MAP FOR OPERATOR INSTRUCTIONS - ASSEMBLER'
MAPSETA  DFHMSD TYPE=&SYSPARM,MODE=INOUT,CTRL=(FREEKB,FRSET),          *
               LANG=ASM,TIOAPFX=YES,EXTATT=MAPONLY,COLOR=BLUE
DFH$AGA  DFHMDI SIZE=(12,40)
         DFHMDF POS=(1,10),LENGTH=21,INITIAL='OPERATOR INSTRUCTIONS',  *
               HILIGHT=UNDERLINE
         DFHMDF POS=(3,1),LENGTH=29,INITIAL='OPERATOR INSTR - ENTER AMN*
               U'
         DFHMDF POS=(4,1),LENGTH=38,INITIAL='FILE INQUIRY   - ENTER AIN*
               Q AND NUMBER'
         DFHMDF POS=(5,1),LENGTH=38,INITIAL='FILE BROWSE    - ENTER ABR*
               W AND NUMBER'
         DFHMDF POS=(6,1),LENGTH=38,INITIAL='FILE ADD       - ENTER AAD*
               D AND NUMBER'
         DFHMDF POS=(7,1),LENGTH=38,INITIAL='FILE UPDATE    - ENTER AUP*
               D AND NUMBER'
MSG      DFHMDF POS=(11,1),LENGTH=39,INITIAL='PRESS CLEAR TO EXIT'
         DFHMDF POS=(12,1),LENGTH=18,INITIAL='ENTER TRANSACTION:'
         DFHMDF POS=(12,20),LENGTH=4,ATTRB=IC,COLOR=GREEN,             *
               HILIGHT=REVERSE
         DFHMDF POS=(12,25),LENGTH=6,INITIAL='NUMBER'
KEY      DFHMDF POS=(12,32),LENGTH=6,ATTRB=NUM,COLOR=GREEN,            *
               HILIGHT=REVERSE
         DFHMDF POS=(12,39),LENGTH=1
         DFHMSD TYPE=FINAL
         END
```

The assembler language DSECT produced as a result of the above statements would be as follows:

DSECT generated by DFH$AGA

```
             DS    0H                 ENSURE ALIGNMENT
DFH$AGAI DS    0C .                   INPUT MAP ORIGIN
DFH$AGAO DS    0C .                   OUTPUT MAP ORIGIN
         DS    12C .             TIOA PREFIX
        SPACE
MSGL    DS    CL2 .                    INPUT DATA FIELD LENGTH
MSGF    DS    0C .                     DATA FIELD FLAG
MSGA    DS    C .                      DATA FIELD ATTRIBUTE
MSGI    DS    0CL39 .               INPUT DATA FIELD
MSGO    DS    CL39 .                OUTPUT DATA FIELD
        SPACE
KEYL    DS    CL2 .                    INPUT DATA FIELD LENGTH
KEYF    DS    0C .                     DATA FIELD FLAG
KEYA    DS    C .                      DATA FIELD ATTRIBUTE
KEYI    DS    0CL6 .               INPUT DATA FIELD
KEYO    DS    CL6 .                OUTPUT DATA FIELD
        SPACE
DFH$AGAE  EQU   *     .               END OF MAP DEFINITION
* * * END OF MAP DEFINITION * * *
         SPACE 3
         ORG
MAPSETAT EQU *       * END OF MAP SET
* * * END OF MAP SET DEFINITION * * *
         SPACE 3
```

DFH$AGA screen layout

```
          +OPERATOR INSTRUCTIONS

+OPERATOR INSTR - ENTER AMNU
+FILE INQUIRY   - ENTER AINQ AND NUMBER
+FILE BROWSE    - ENTER ABRW AND NUMBER
+FILE ADD       - ENTER AADD AND NUMBER
+FILE UPDATE    - ENTER AUPD AND NUMBER



+PRESS CLEAR TO EXIT
+ENTER TRANSACTION:+XXXX+NUMBER+XXXXXX+
```

DFH$AGB map definition

```
         TITLE 'FILEA - MAP FOR FILE INQUIRY/UPDATE - ASSEMBLER'
MAPSETB  DFHMSD TYPE=&SYSPARM,MODE=INOUT,CTRL=(FREEKB,FRSET),          *
               LANG=ASM,TIOAPFX=YES,EXTATT=MAPONLY
DFH$AGB  DFHMDI SIZE=(12,40)
TITLE    DFHMDF POS=(1,15),LENGTH=12
         DFHMDF POS=(3,1),LENGTH=8,INITIAL='NUMBER:',COLOR=BLUE
NUMB     DFHMDF POS=(3,10),LENGTH=6
         DFHMDF POS=(3,17),LENGTH=1
         DFHMDF POS=(4,1),LENGTH=8,INITIAL='NAME:   ',COLOR=BLUE
NAME     DFHMDF POS=(4,10),LENGTH=20,ATTRB=(UNPROT,IC)
         DFHMDF POS=(4,31),LENGTH=1
         DFHMDF POS=(5,1),LENGTH=8,INITIAL='ADDRESS:',COLOR=BLUE
ADDR     DFHMDF POS=(5,10),LENGTH=20,ATTRB=UNPROT
         DFHMDF POS=(5,31),LENGTH=1
         DFHMDF POS=(6,1),LENGTH=8,INITIAL='PHONE:   ',COLOR=BLUE
PHONE    DFHMDF POS=(6,10),LENGTH=8,ATTRB=UNPROT
         DFHMDF POS=(6,19),LENGTH=1
         DFHMDF POS=(7,1),LENGTH=8,INITIAL='DATE:    ',COLOR=BLUE
DATE     DFHMDF POS=(7,10),LENGTH=8,ATTRB=UNPROT
         DFHMDF POS=(7,19),LENGTH=1
         DFHMDF POS=(8,1),LENGTH=8,INITIAL='AMOUNT:  ',COLOR=BLUE
AMOUNT   DFHMDF POS=(8,10),LENGTH=8,ATTRB=NUM
         DFHMDF POS=(8,19),LENGTH=1
         DFHMDF POS=(9,1),LENGTH=8,INITIAL='COMMENT:',COLOR=BLUE
COMMENT  DFHMDF POS=(9,10),LENGTH=9,ATTRB=UNPROT
         DFHMDF POS=(9,20),LENGTH=1
MSG1     DFHMDF POS=(11,1),LENGTH=39
MSG3     DFHMDF POS=(12,1),LENGTH=39
         DFHMSD TYPE=FINAL
         END
```

The assembler language DSECT produced as a result of the above statements would be as follows:

DSECT generated by DFH$AGB

```
             DS    0H                 ENSURE ALIGNMENT
DFH$AGBI DS    0C .                   INPUT MAP ORIGIN
DFH$AGBO DS    0C .                   OUTPUT MAP ORIGIN
         DS    12C .             TIOA PREFIX
        SPACE
TITLEL    DS    CL2 .                    INPUT DATA FIELD LENGTH
TITLEF    DS    0C .                     DATA FIELD FLAG
TITLEA    DS    C .                      DATA FIELD ATTRIBUTE
TITLEI    DS    0CL12 .               INPUT DATA FIELD
TITLEO    DS    CL12 .                OUTPUT DATA FIELD
        SPACE
NUMBL    DS    CL2 .                    INPUT DATA FIELD LENGTH
NUMBF    DS    0C .                     DATA FIELD FLAG
NUMBA    DS    C .                      DATA FIELD ATTRIBUTE
NUMBI    DS    0CL6 .               INPUT DATA FIELD
NUMBO    DS    CL6 .                OUTPUT DATA FIELD
        SPACE
NAMEL    DS    CL2 .                    INPUT DATA FIELD LENGTH
NAMEF    DS    0C .                     DATA FIELD FLAG
NAMEA    DS    C .                      DATA FIELD ATTRIBUTE
NAMEI    DS    0CL20 .               INPUT DATA FIELD
NAMEO    DS    CL20 .                OUTPUT DATA FIELD
        SPACE
ADDRL    DS    CL2 .                    INPUT DATA FIELD LENGTH
ADDRF    DS    0C .                     DATA FIELD FLAG
ADDRA    DS    C .                      DATA FIELD ATTRIBUTE
ADDRI    DS    0CL20 .               INPUT DATA FIELD
ADDRO    DS    CL20 .                OUTPUT DATA FIELD
        SPACE
PHONEL    DS    CL2 .                    INPUT DATA FIELD LENGTH
PHONEF    DS    0C .                     DATA FIELD FLAG
PHONEA    DS    C .                      DATA FIELD ATTRIBUTE
PHONEI    DS    0CL8 .               INPUT DATA FIELD
PHONEO    DS    CL8 .                OUTPUT DATA FIELD
        SPACE
DATEL    DS    CL2 .                    INPUT DATA FIELD LENGTH
DATEF    DS    0C .                     DATA FIELD FLAG
DATEA    DS    C .                      DATA FIELD ATTRIBUTE
DATEI    DS    0CL8 .               INPUT DATA FIELD
DATEO    DS    CL8 .                OUTPUT DATA FIELD
        SPACE
AMOUNTL    DS    CL2 .                    INPUT DATA FIELD LENGTH
AMOUNTF    DS    0C .                     DATA FIELD FLAG
AMOUNTA    DS    C .                      DATA FIELD ATTRIBUTE
AMOUNTI    DS    0CL8 .               INPUT DATA FIELD
AMOUNTO    DS    CL8 .                OUTPUT DATA FIELD
        SPACE
COMMENTL    DS    CL2 .                    INPUT DATA FIELD LENGTH
COMMENTF    DS    0C .                     DATA FIELD FLAG
COMMENTA    DS    C .                      DATA FIELD ATTRIBUTE
COMMENTI    DS    0CL9 .               INPUT DATA FIELD
COMMENTO    DS    CL9 .                OUTPUT DATA FIELD
        SPACE
```

DSECT generated by DFH$AGB (continued)

```
MSG1L    DS    CL2 .                    INPUT DATA FIELD LENGTH
MSG1F    DS    0C .                     DATA FIELD FLAG
MSG1A    DS    C .                      DATA FIELD ATTRIBUTE
MSG1I    DS    0CL39 .               INPUT DATA FIELD
MSG1O    DS    CL39 .                OUTPUT DATA FIELD
        SPACE
MSG3L    DS    CL2 .                    INPUT DATA FIELD LENGTH
MSG3F    DS    0C .                     DATA FIELD FLAG
MSG3A    DS    C .                      DATA FIELD ATTRIBUTE
MSG3I    DS    0CL39 .               INPUT DATA FIELD
MSG3O    DS    CL39 .                OUTPUT DATA FIELD
        SPACE
DFH$AGBE  EQU   *      .              END OF MAP DEFINITION
* * * END OF MAP DEFINITION * * *
         SPACE 3
         ORG
MAPSETBT EQU *       * END OF MAP SET
* * * END OF MAP SET DEFINITION * * *
         SPACE 3
```

DFH$AGB screen layout

```
              +XXXXXXXXXXXX

+NUMBER: +XXXXXX+
+NAME:   +XXXXXXXXXXXXXXXXXXXX+
+ADDRESS:+XXXXXXXXXXXXXXXXXXXX+
+PHONE:  +XXXXXXXX+
+DATE:   +XXXXXXXX+
+AMOUNT: +XXXXXXXX+
+COMMENT:+XXXXXXXXX+

+XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
+XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

DFH$AGC map definition

```
         TITLE 'FILEA - MAP FOR FILE BROWSE - ASSEMBLER'
MAPSETC  DFHMSD TYPE=&SYSPARM,MODE=INOUT,CTRL=(FREEKB,FRSET),          *
               LANG=ASM,TIOAPFX=YES,EXTATT=MAPONLY
DFH$AGC  DFHMDI SIZE=(12,40)
DIR      DFHMDF POS=(1,1),LENGTH=1,ATTRB=IC
         DFHMDF POS=(1,3),LENGTH=1
         DFHMDF POS=(1,15),LENGTH=11,INITIAL='FILE BROWSE',            *
               COLOR=BLUE,HILIGHT=UNDERLINE
         DFHMDF POS=(3,1),LENGTH=6,INITIAL='NUMBER',COLOR=BLUE
         DFHMDF POS=(3,17),LENGTH=4,INITIAL='NAME',COLOR=BLUE
         DFHMDF POS=(3,32),LENGTH=6,INITIAL='AMOUNT',COLOR=BLUE
NUMBER1  DFHMDF POS=(4,1),LENGTH=6
NAME1    DFHMDF POS=(4,9),LENGTH=20
AMOUNT1  DFHMDF POS=(4,30),LENGTH=8
NUMBER2  DFHMDF POS=(5,1),LENGTH=6
NAME2    DFHMDF POS=(5,9),LENGTH=20
AMOUNT2  DFHMDF POS=(5,30),LENGTH=8
NUMBER3  DFHMDF POS=(6,1),LENGTH=6
NAME3    DFHMDF POS=(6,9),LENGTH=20
AMOUNT3  DFHMDF POS=(6,30),LENGTH=8
NUMBER4  DFHMDF POS=(7,1),LENGTH=6
NAME4    DFHMDF POS=(7,9),LENGTH=20
AMOUNT4  DFHMDF POS=(7,30),LENGTH=8
MSG0     DFHMDF POS=(10,1),LENGTH=39,COLOR=BLUE,                       *
               INITIAL='PRESS CLEAR TO END BROWSE OPERATION'
MSG1     DFHMDF POS=(11,1),LENGTH=39,COLOR=BLUE,                       *
               INITIAL='PRESS PF1 OR TYPE F TO PAGE FORWARD'
MSG2     DFHMDF POS=(12,1),LENGTH=39,COLOR=BLUE,                       *
               INITIAL='PRESS PF2 OR TYPE B TO PAGE BACKWARD'
         DFHMSD TYPE=FINAL
         END
```

The assembler language DSECT produced as a result of the above statements would be as follows:

DSECT generated by DFH$AGC

```
             DS    0H                 ENSURE ALIGNMENT
DFH$AGCI DS    0C .                   INPUT MAP ORIGIN
DFH$AGCO DS    0C .                   OUTPUT MAP ORIGIN
         DS    12C .             TIOA PREFIX
        SPACE
DIRL    DS    CL2 .                    INPUT DATA FIELD LENGTH
DIRF    DS    0C .                     DATA FIELD FLAG
DIRA    DS    C .                      DATA FIELD ATTRIBUTE
DIRI    DS    0CL1 .               INPUT DATA FIELD
DIRO    DS    CL1 .                OUTPUT DATA FIELD
        SPACE
NUMBER1L    DS    CL2 .                    INPUT DATA FIELD LENGTH
NUMBER1F    DS    0C .                     DATA FIELD FLAG
NUMBER1A    DS    C .                      DATA FIELD ATTRIBUTE
NUMBER1I    DS    0CL6 .               INPUT DATA FIELD
NUMBER1O    DS    CL6 .                OUTPUT DATA FIELD
        SPACE
NAME1L    DS    CL2 .                    INPUT DATA FIELD LENGTH
NAME1F    DS    0C .                     DATA FIELD FLAG
NAME1A    DS    C .                      DATA FIELD ATTRIBUTE
NAME1I    DS    0CL20 .              INPUT DATA FIELD
NAME1O    DS    CL20 .               OUTPUT DATA FIELD
        SPACE
AMOUNT1L    DS    CL2 .                    INPUT DATA FIELD LENGTH
AMOUNT1F    DS    0C .                     DATA FIELD FLAG
AMOUNT1A    DS    C .                      DATA FIELD ATTRIBUTE
AMOUNT1I    DS    0CL8 .               INPUT DATA FIELD
AMOUNT1O    DS    CL8 .                OUTPUT DATA FIELD
        SPACE
NUMBER2L    DS    CL2 .                    INPUT DATA FIELD LENGTH
NUMBER2F    DS    0C .                     DATA FIELD FLAG
NUMBER2A    DS    C .                      DATA FIELD ATTRIBUTE
NUMBER2I    DS    0CL6 .               INPUT DATA FIELD
NUMBER2O    DS    CL6 .                OUTPUT DATA FIELD
        SPACE
NAME2L    DS    CL2 .                    INPUT DATA FIELD LENGTH
NAME2F    DS    0C .                     DATA FIELD FLAG
NAME2A    DS    C .                      DATA FIELD ATTRIBUTE
NAME2I    DS    0CL20 .              INPUT DATA FIELD
NAME2O    DS    CL20 .               OUTPUT DATA FIELD
        SPACE
AMOUNT2L    DS    CL2 .                    INPUT DATA FIELD LENGTH
AMOUNT2F    DS    0C .                     DATA FIELD FLAG
AMOUNT2A    DS    C .                      DATA FIELD ATTRIBUTE
AMOUNT2I    DS    0CL8 .               INPUT DATA FIELD
AMOUNT2O    DS    CL8 .                OUTPUT DATA FIELD
        SPACE
NUMBER3L    DS    CL2 .                    INPUT DATA FIELD LENGTH
NUMBER3F    DS    0C .                     DATA FIELD FLAG
NUMBER3A    DS    C .                      DATA FIELD ATTRIBUTE
NUMBER3I    DS    0CL6 .               INPUT DATA FIELD
NUMBER3O    DS    CL6 .                OUTPUT DATA FIELD
        SPACE
NAME3L    DS    CL2 .                    INPUT DATA FIELD LENGTH
```

DSECT generated by DFH$AGC (continued)

```
NAME3F    DS    0C .                     DATA FIELD FLAG
NAME3A    DS    C .                      DATA FIELD ATTRIBUTE
NAME3I    DS    0CL20 .               INPUT DATA FIELD
NAME3O    DS    CL20 .                OUTPUT DATA FIELD
        SPACE
AMOUNT3L    DS    CL2 .                    INPUT DATA FIELD LENGTH
AMOUNT3F    DS    0C .                     DATA FIELD FLAG
AMOUNT3A    DS    C .                      DATA FIELD ATTRIBUTE
AMOUNT3I    DS    0CL8 .               INPUT DATA FIELD
AMOUNT3O    DS    CL8 .                OUTPUT DATA FIELD
        SPACE
NUMBER4L    DS    CL2 .                    INPUT DATA FIELD LENGTH
NUMBER4F    DS    0C .                     DATA FIELD FLAG
NUMBER4A    DS    C .                      DATA FIELD ATTRIBUTE
NUMBER4I    DS    0CL6 .               INPUT DATA FIELD
NUMBER4O    DS    CL6 .                OUTPUT DATA FIELD
        SPACE
NAME4L    DS    CL2 .                    INPUT DATA FIELD LENGTH
NAME4F    DS    0C .                     DATA FIELD FLAG
NAME4A    DS    C .                      DATA FIELD ATTRIBUTE
NAME4I    DS    0CL20 .               INPUT DATA FIELD
NAME4O    DS    CL20 .                OUTPUT DATA FIELD
        SPACE
AMOUNT4L    DS    CL2 .                    INPUT DATA FIELD LENGTH
AMOUNT4F    DS    0C .                     DATA FIELD FLAG
AMOUNT4A    DS    C .                      DATA FIELD ATTRIBUTE
AMOUNT4I    DS    0CL8 .               INPUT DATA FIELD
AMOUNT4O    DS    CL8 .                OUTPUT DATA FIELD
        SPACE
MSG0L    DS    CL2 .                    INPUT DATA FIELD LENGTH
MSG0F    DS    0C .                     DATA FIELD FLAG
MSG0A    DS    C .                      DATA FIELD ATTRIBUTE
MSG0I    DS    0CL39 .               INPUT DATA FIELD
MSG0O    DS    CL39 .                OUTPUT DATA FIELD
        SPACE
MSG1L    DS    CL2 .                    INPUT DATA FIELD LENGTH
MSG1F    DS    0C .                     DATA FIELD FLAG
MSG1A    DS    C .                      DATA FIELD ATTRIBUTE
MSG1I    DS    0CL39 .               INPUT DATA FIELD
MSG1O    DS    CL39 .                OUTPUT DATA FIELD
        SPACE
MSG2L    DS    CL2 .                    INPUT DATA FIELD LENGTH
MSG2F    DS    0C .                     DATA FIELD FLAG
MSG2A    DS    C .                      DATA FIELD ATTRIBUTE
MSG2I    DS    0CL39 .               INPUT DATA FIELD
MSG2O    DS    CL39 .                OUTPUT DATA FIELD
        SPACE
DFH$AGCE  EQU   *     .               END OF MAP DEFINITION
* * * END OF MAP DEFINITION * * *
         SPACE 3
         ORG
MAPSETCT EQU *       * END OF MAP SET
* * * END OF MAP SET DEFINITION * * *
         SPACE 3
```

DFH$AGC screen layout

```
               +FILE BROWSE

+NUMBER         +NAME          +AMOUNT
+XXXXXX +XXXXXXXXXXXXXXXXXXXX+XXXXXXXX
+XXXXXX +XXXXXXXXXXXXXXXXXXXX+XXXXXXXX
+XXXXXX +XXXXXXXXXXXXXXXXXXXX+XXXXXXXX
+XXXXXX +XXXXXXXXXXXXXXXXXXXX+XXXXXXXX


+PRESS CLEAR TO END BROWSE OPERATION
+PRESS PF1 OR TYPE F TO PAGE FORWARD
+PRESS PF2 OR TYPE B TO PAGE BACKWARD
```

DFH$AGD map definition

```
         TITLE 'FILEA - MAPSET FOR LOW BALANCE REPORT - ASSEMBLER'
MAPSETD  DFHMSD TYPE=&SYSPARM,MODE=OUT,CTRL=(FREEKB,FRSET),            *
               LANG=ASM,STORAGE=AUTO,EXTATT=MAPONLY,COLOR=BLUE
DFH$AGD  DFHMDI SIZE=(1,40),COLOR=GREEN
NUMBER   DFHMDF POS=(1,1),LENGTH=6
NAME     DFHMDF POS=(1,9),LENGTH=20
AMOUNT   DFHMDF POS=(1,30),LENGTH=8
HEADING  DFHMDI SIZE=(3,40),HEADER=YES
         DFHMDF POS=(1,5),LENGTH=18,INITIAL='LOW BALANCE REPORT',      *
               HILIGHT=UNDERLINE
         DFHMDF POS=(1,30),LENGTH=4,INITIAL='PAGE'
PAGEN    DFHMDF POS=(1,35),LENGTH=3
         DFHMDF POS=(3,1),LENGTH=6,INITIAL='NUMBER'
         DFHMDF POS=(3,17),LENGTH=4,INITIAL='NAME'
         DFHMDF POS=(3,32),LENGTH=6,INITIAL='AMOUNT'
FOOTING  DFHMDI SIZE=(2,40),TRAILER=YES,JUSTIFY=LAST
         DFHMDF POS=(2,1),LENGTH=38,                                   *
               INITIAL='PRESS CLEAR AND TYPE P/N TO SEE PAGE N'
FINAL    DFHMDI SIZE=(2,40),TRAILER=YES,JUSTIFY=LAST
         DFHMDF POS=(2,10),LENGTH=14,INITIAL='END OF REPORT.'
         DFHMSD TYPE=FINAL
         END
```

The assembler language DSECT produced as a result of the above statements would be as follows:

DSECT generated by DFH$AGD

```
             DS    0H                 ENSURE ALIGNMENT
DFH$AGDO DS    0C .
         DS    12C .             TIOA PREFIX
        SPACE
         DS    CL2 .                    INPUT DATA FIELD LENGTH
NUMBERA    DS    C .                      DATA FIELD ATTRIBUTE
NUMBERO    DS    CL6 .                DATA FIELD
        SPACE
         DS    CL2 .                    INPUT DATA FIELD LENGTH
NAMEA    DS    C .                      DATA FIELD ATTRIBUTE
NAMEO    DS    CL20 .                DATA FIELD
        SPACE
         DS    CL2 .                    INPUT DATA FIELD LENGTH
AMOUNTA    DS    C .                      DATA FIELD ATTRIBUTE
AMOUNTO    DS    CL8 .                DATA FIELD
        SPACE
DFH$AGDE  EQU   *     .               END OF MAP DEFINITION
* * * END OF MAP DEFINITION * * *
         SPACE 3
             DS    0H                 ENSURE ALIGNMENT
HEADINGO DS    0C .
         DS    12C .             TIOA PREFIX
        SPACE
         DS    CL2 .                    INPUT DATA FIELD LENGTH
PAGENA    DS    C .                      DATA FIELD ATTRIBUTE
PAGENO    DS    CL3 .                DATA FIELD
        SPACE
HEADINGE  EQU   *     .               END OF MAP DEFINITION
* * * END OF MAP DEFINITION * * *
         SPACE 3
             DS    0H                 ENSURE ALIGNMENT
FOOTINGO DS    0C .
         DS    12C .             TIOA PREFIX
        SPACE
FOOTINGE  EQU   *     .               END OF MAP DEFINITION
* * * END OF MAP DEFINITION * * *
         SPACE 3
             DS    0H                 ENSURE ALIGNMENT
FINALO DS    0C .
         DS    12C .             TIOA PREFIX
        SPACE
FINALE  EQU   *     .               END OF MAP DEFINITION
* * * END OF MAP DEFINITION * * *
         SPACE 3
         ORG
MAPSETDT EQU *       * END OF MAP SET
* * * END OF MAP SET DEFINITION * * *
         SPACE 3
```

DFH$AGD screen layout

```
     +LOW BALANCE REPORT          +PAGE+XXX

+NUMBER         +NAME          +AMOUNT
+XXXXXX+XXXXXXXXXXXXXXXXXXXX  XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX  XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX  XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX  XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX  XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX  XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX  XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX  XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX  XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX  XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX  XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX  XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX  XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX  XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX  XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX  XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX  XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX  XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX  XXXXXXXX

+PRESS CLEAR AND TYPE P/N TO SEE PAGE N
```

DFH$AGK map definition

```
         TITLE 'FILEA - MAP FOR ORDER ENTRY - ASSEMBLER'
MAPSETK  DFHMSD TYPE=&SYSPARM,MODE=INOUT,CTRL=(FREEKB,FRSET),          *
               TIOAPFX=YES,LANG=ASM,EXTATT=MAPONLY
DFH$AGK  DFHMDI SIZE=(12,40)
         DFHMDF POS=(01,10),LENGTH=11,ATTRB=(BRT,ASKIP),               *
               INITIAL='ORDER ENTRY',COLOR=BLUE,HILIGHT=UNDERLINE
MSG1     DFHMDF POS=(03,04),LENGTH=26,ATTRB=(DRK,ASKIP),               *
               INITIAL='NUMBER NOT FOUND - REENTER',                   *
               COLOR=RED,HILIGHT=BLINK
MSG2     DFHMDF POS=(04,04),LENGTH=22,ATTRB=(DRK,ASKIP),               *
               INITIAL='DATA ERROR - REENTER',                         *
               COLOR=RED,HILIGHT=BLINK
         DFHMDF POS=(05,04),LENGTH=09,ATTRB=PROT,                      *
               INITIAL='NUMBER  :'
CUSTNO   DFHMDF POS=(05,14),LENGTH=06,ATTRB=(IC,NUM)
         DFHMDF POS=(05,21),LENGTH=01
         DFHMDF POS=(06,04),LENGTH=09,ATTRB=PROT,COLOR=BLUE,           *
               INITIAL='PART NO :'
PARTNO   DFHMDF POS=(06,14),LENGTH=06,ATTRB=NUM
         DFHMDF POS=(06,21),LENGTH=01
         DFHMDF POS=(07,04),LENGTH=09,ATTRB=PROT,COLOR=BLUE,           *
               INITIAL='QUANTITY:'
QUANT    DFHMDF POS=(07,14),LENGTH=06,ATTRB=NUM
         DFHMDF POS=(07,21),LENGTH=01
         DFHMDF POS=(09,01),LENGTH=38,ATTRB=ASKIP,COLOR=BLUE,          *
               INITIAL='PRESS ENTER TO CONTINUE,CLEAR TO QUIT'
         DFHMSD TYPE=FINAL
         END
```

The assembler language DSECT produced as a result of the above statements would be as follows:

DSECT generated by DFH$AGK

```
             DS    0H                 ENSURE ALIGNMENT
DFH$AGKI DS    0C .                   INPUT MAP ORIGIN
DFH$AGKO DS    0C .                   OUTPUT MAP ORIGIN
         DS    12C .             TIOA PREFIX
        SPACE
MSG1L    DS    CL2 .                    INPUT DATA FIELD LENGTH
MSG1F    DS    0C .                     DATA FIELD FLAG
MSG1A    DS    C .                      DATA FIELD ATTRIBUTE
MSG1I    DS    0CL26 .               INPUT DATA FIELD
MSG1O    DS    CL26 .                OUTPUT DATA FIELD
        SPACE
MSG2L    DS    CL2 .                    INPUT DATA FIELD LENGTH
MSG2F    DS    0C .                     DATA FIELD FLAG
MSG2A    DS    C .                      DATA FIELD ATTRIBUTE
MSG2I    DS    0CL22 .               INPUT DATA FIELD
MSG2O    DS    CL22 .                OUTPUT DATA FIELD
        SPACE
CUSTNOL    DS    CL2 .                  INPUT DATA FIELD LENGTH
CUSTNOF    DS    0C .                   DATA FIELD FLAG
CUSTNOA    DS    C .                    DATA FIELD ATTRIBUTE
CUSTNOI    DS    0CL6 .             INPUT DATA FIELD
CUSTNOO    DS    CL6 .              OUTPUT DATA FIELD
        SPACE
PARTNOL    DS    CL2 .                  INPUT DATA FIELD LENGTH
PARTNOF    DS    0C .                   DATA FIELD FLAG
PARTNOA    DS    C .                    DATA FIELD ATTRIBUTE
PARTNOI    DS    0CL6 .             INPUT DATA FIELD
PARTNOO    DS    CL6 .              OUTPUT DATA FIELD
        SPACE
QUANTL    DS    CL2 .                   INPUT DATA FIELD LENGTH
QUANTF    DS    0C .                    DATA FIELD FLAG
QUANTA    DS    C .                     DATA FIELD ATTRIBUTE
QUANTI    DS    0CL6 .              INPUT DATA FIELD
QUANTO    DS    CL6 .               OUTPUT DATA FIELD
        SPACE
DFH$AGKE  EQU    *     .              END OF MAP DEFINITION
* * * END OF MAP DEFINITION * * *
         SPACE 3
         ORG
MAPSETKT EQU *      * END OF MAP SET
* * * END OF MAP SET DEFINITION * * *
         SPACE 3
```

DFH$AGK screen layout

```
          +ORDER ENTRY

   +NUMBER NOT FOUND  REENTER
   +DATA ERROR  REENTER
   +NUMBER  :+XXXXXX+
   +PART NO :+XXXXXX+
   +QUANTITY:+XXXXXX+

+PRESS ENTER TO CONTINUE,CLEAR TO QUIT
```

DFH$AGL map definition

```
         TITLE 'FILEA - MAP FOR ORDER ENTRY QUEUE PRINT - ASSEMBLER'
MAPSETL  DFHMSD TYPE=&SYSPARM,MODE=OUT,                                *
               TIOAPFX=YES,LANG=ASM
DFH$AGL  DFHMDI SIZE=(05,80)
TITLE    DFHMDF POS=(01,01),LENGTH=43,                                 *
               INITIAL='NUMBER     NAME                     ADDRESS'
NUMB     DFHMDF POS=(02,01),LENGTH=06
NAM      DFHMDF POS=(02,12),LENGTH=20
ADDR     DFHMDF POS=(02,37),LENGTH=20
         DFHMDF POS=(03,01),LENGTH=09,                                 *
               INITIAL='PART NO :'
PART     DFHMDF POS=(03,11),LENGTH=06
         DFHMDF POS=(04,01),LENGTH=09,                                 *
               INITIAL='QUANTITY:'
QUANT    DFHMDF POS=(04,11),LENGTH=06
         DFHMDF POS=(05,01),LENGTH=1,                                  *
               INITIAL=' '
         DFHMSD TYPE=FINAL
         END
```

The assembler language DSECT produced as a result of the above statements would be as follows:

DSECT generated by DFH$AGL

```
             DS    0H                 ENSURE ALIGNMENT
DFH$AGLO DS    0C .
         DS    12C .             TIOA PREFIX
        SPACE
         DS    CL2 .                    INPUT DATA FIELD LENGTH
TITLEA    DS    C .                      DATA FIELD ATTRIBUTE
TITLEO    DS    CL43 .                DATA FIELD
        SPACE
         DS    CL2 .                    INPUT DATA FIELD LENGTH
NUMBA    DS    C .                      DATA FIELD ATTRIBUTE
NUMBO    DS    CL6 .                DATA FIELD
        SPACE
         DS    CL2 .                    INPUT DATA FIELD LENGTH
NAMA    DS    C .                      DATA FIELD ATTRIBUTE
NAMO    DS    CL20 .                DATA FIELD
        SPACE
         DS    CL2 .                    INPUT DATA FIELD LENGTH
ADDRA    DS    C .                      DATA FIELD ATTRIBUTE
ADDRO    DS    CL20 .                DATA FIELD
        SPACE
         DS    CL2 .                    INPUT DATA FIELD LENGTH
PARTA    DS    C .                      DATA FIELD ATTRIBUTE
PARTO    DS    CL6 .                DATA FIELD
        SPACE
         DS    CL2 .                    INPUT DATA FIELD LENGTH
QUANTA    DS    C .                      DATA FIELD ATTRIBUTE
QUANTO    DS    CL6 .                DATA FIELD
        SPACE
DFH$AGLE  EQU   *     .               END OF MAP DEFINITION
* * * END OF MAP DEFINITION * * *
         SPACE 3
         ORG
MAPSETLT EQU *       * END OF MAP SET
* * * END OF MAP SET DEFINITION * * *
         SPACE 3
```

DFH$AGL print format

```
+NUMBER     NAME                    ADDRESS
+XXXXXX    +XXXXXXXXXXXXXXXXXXXX    +XXXXXXXXXXXXXXXXXXXX
+PART NO :+XXXXXX
+QUANTITY:+XXXXXX
+X
```

Record descriptions for ASM sample programs

FILEA record description

The sample programs use the FILEA record description. It is defined in copy code DFH$AFIL and has the following format:

```
FILEA    DS    0CL80
FILEREC  DS    0CL80
STAT     DS    CL1
NUMB     DS    CL6
NAME     DS    CL20
ADDRX    DS    CL20
PHONE    DS    CL8
DATEX    DS    CL8
AMOUNT   DS    CL8
COMMENT  DS    CL9
```

LOGA record description

The sample programs use the LOGA record description when an audit trail is written to a transient data file. It is defined in copy code DFH$ALOG and has the following format:

```
LOGA     DS    0CL92
LOGHDR   DS    0CL12
LDAY     DS    PL4
LTIME    DS    PL4
LTERML   DS    CL4
LOGREC   DS    0CL80
LSTAT    DS    CL1
LNUMB    DS    CL6
LNAME    DS    CL20
LADDR    DS    CL20
LPHONE   DS    CL8
LDATE    DS    CL8
LAMOUNT  DS    CL8
LCOMMENT DS    CL9
```

L86O record description

The Order Entry Queue Print sample program uses the L86O record description when it writes to the transient data queue 'L86O'. It is defined in copy code DFH$AL86 and has the following format:

```
L86O     DS    0CL22
ITEM     DS    0CL22
CUSTNO   DS    CL6
PARTNO   DS    CL6
QUANTITY DS    CL6
TERMID   DS    CL4
```

Appendix E. Sample programs (COBOL)

The COBOL sample programs described in this appendix are included, in source form, on the CICS distribution tape. The *CICS/MVS Installation Guide* describes how these sample programs, and associated resources, can be defined to CICS and how the programs can be executed online.

This appendix describes six CICS sample application programs, written in COBOL, as follows:

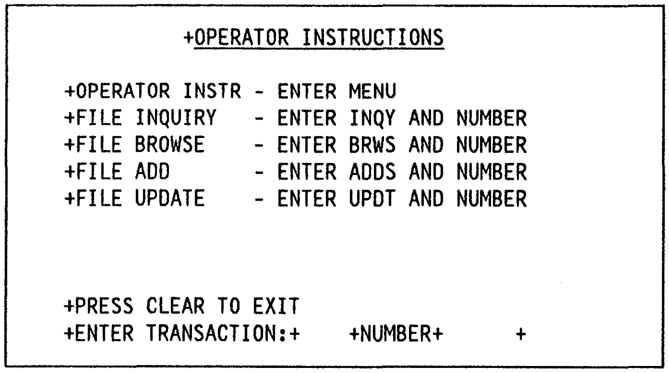
- Operator instruction
- Inquiry/update
- Browse
- Order entry
- Order entry queue print
- Low balance report.

These programs illustrate basic applications (such as inquire, browse, add, and update) that can serve as a framework for your installation's first programs. The programs operate using a VSAM file, known as FILEA, consisting of records containing details of individual customer accounts. Each program has a short description of what the program does, a listing of its source code, and a series of program notes. Numbered coding lines in the source listing correspond to the numbered program notes. The programs contain COPY statements coded according to the 1968 COBOL standard.

All the sample programs are for use with the IBM 3270 Information Display System.

The sample BMS maps include examples of how the COLOR, EXTATT, and HILIGHT attributes are specified in the map definition macros. However, due to production limitations, the associated screen layouts do not show you all the effects of these attributes.

You can add attributes without changing the application program by specifying EXTATT=MAPONLY in the DFHMSD map set definition macro. If you include an attribute that specifies a facility not available at the terminal, it will be ignored.



```
              +OPERATOR INSTRUCTIONS

    +OPERATOR INSTR - ENTER MENU
    +FILE INQUIRY   - ENTER INQY AND NUMBER
    +FILE BROWSE    - ENTER BRWS AND NUMBER
    +FILE ADD       - ENTER ADDS AND NUMBER
    +FILE UPDATE    - ENTER UPDT AND NUMBER



    +PRESS CLEAR TO EXIT
    +ENTER TRANSACTION:+    +NUMBER+      +
```

The statements listed are those of the sample programs supplied with the initial release of CICS. Sample programs shipped with subsequent program temporary fixes (PTFs) may differ from these listings.

The BMS maps (which are unaligned) and the file record descriptions used by these sample programs are included at the end of the appendix.

Once CICS is running, type MENU onto a clear screen and press the enter key. The MENU transaction identifier invokes the 'Operator Instruction' sample program, which is a short program that produces a menu containing the transaction identifiers for two of the other sample programs, namely 'Inquiry/Update' and 'Browse'.

If you clear the screen, remember to reenter the transaction identifier, as no data is accepted from an unformatted screen.

You can run the sample programs using EDF but, because the CEDF transaction is defined with RSLC=YES, you must first sign on to CICS as an operator with an appropriate resource security level key.

The menu, on a screen that is 40 characters wide by 12 lines deep, is as shown in the box above. The plus (+) sign in this and subsequent displays shows the position of the attribute byte. In an actual display, this position contains a blank.

To invoke any of the transactions MENU, INQY, BRWS, ADDS, or UPDT, do as instructed, entering the four-character transaction identifier and six-digit account number in the fields highlighted in the bottom line of the display. These specific account numbers include the sequence 100000, 111111, 200000, 222222, ..., 999999.

These transaction identifiers give you access to the inquiry, add, and update functions of the 'Inquiry/Update' program, and access to the 'Browse' program.

You can invoke the three remaining sample programs 'Order Entry', 'Order Entry Queue Print', and 'Low Balance Report' separately by entering their transaction identifiers (OREN, OREQ, and REPT respectively) onto a clear screen.

Operator instruction program (COBOL)

Description

The operator instruction sample program displays map DFH$CGA in response to the EXEC CICS SEND MAP command.

The map displays a menu that lists the transaction identifiers associated with two of the sample programs, 'Inquiry /Update', and 'Browse', and gives instructions for the operator.

Source listing for DFH$CMNU

```
*****************************************************************
*    DFH$CMNU - CICS/VS SAMPLE FILEA OPERATOR INSTRUCTION MENU  *
*****************************************************************
 IDENTIFICATION DIVISION.
 PROGRAM-ID. FILECMNU.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
*
 PROCEDURE DIVISION.
     EXEC CICS SEND MAP('MENU') MAPSET('DFH$CGA')
              MAPONLY ERASE END-EXEC.
     EXEC CICS RETURN END-EXEC.
     GOBACK.
```

Program notes

1. The BMS command erases the screen and displays map DFH$CGA.
2. The RETURN command ends the program.

Inquiry/update sample program (COBOL)

Description

The inquiry/update sample program lets you make an inquiry about, add to, or update records in a file. You can select one of these by entering the appropriate transaction identifier (INQY, ADDS, or UPDT) in the menu that is displayed when you start operations by entering MENU.

To make an inquiry, enter INQY and an account number into the menu. The program maps in the account number and reads the record from FILEA. The required fields from the file area, and a title 'FILE INQUIRY' are moved to the map dsect for DFH$CGB. DFH$CGB, containing the record fields, is displayed at your screen.

To add a record, enter ADDS and the account number into the menu. The account number and a title 'FILE ADD' are moved to the map area of DFH$CGB. DFH$CGB, containing empty data fields, is displayed at your screen. The data fields entered are mapped into DFH$CGB and moved to the file record area which is then written to FILEA. The addition is recorded on an update log (LOGA), which is a transient data queue. The operator instruction screen is displayed with the message 'RECORD ADDED'.

To update a record, enter UPDT and the account number into the menu, as before. The program reads and displays the requested FILEA record. Modified data fields are mapped in to DFH$CGB and edited. The sample program only suggests the type of editing you might wish to do. Insert editing steps needed to ensure valid changes to the file. Those fields that have been changed are moved to the data record and the record is rewritten to FILEA. The update is recorded on LOGA. The message 'RECORD UPDATED' is moved to the dsect for DFH$CGA, the operator instruction menu map, which is then displayed at your screen.

This program is an example of pseudoconversational programming, in which control is returned to CICS together with a transaction identifier whenever a response is requested from the operator. Associated with each return of control to CICS is a storage area containing details of the previous invocation of the transaction.

Source listing for DFH$CALL

```
   *****************************************************************
   *    DFH$CALL - CICS/VS SAMPLE FILEA INQUIRY/UPDATE - COBOL     *
   *****************************************************************
    IDENTIFICATION DIVISION.
    PROGRAM-ID. FILECALL.
    ENVIRONMENT DIVISION.
    DATA DIVISION.
    WORKING-STORAGE SECTION.
    77  MESSAGES     PIC X(39).
    77  TEMP-NAME    PIC X(20).
    77  KEYNUM       PIC 9(6).
    77  COMLEN       PIC S9(4) COMP.
                     COPY DFH$CGA.
                     COPY DFH$CGB.
   *    NEXT FIELD TO VERIFY AMOUNTI
    01  AMOUNTN.
        03  AMOUNTN1  PIC X.
        03  AMOUNTN25 PIC X(4).
        03  AMOUNTN6  PIC X.
        03  AMOUNTN78 PIC X(2).
    01  FILEA.       COPY DFH$CFIL.
    01  LOGA.        COPY DFH$CLOG.
                     COPY DFHBMSCA.
    01  COMMAREA.    COPY DFH$CFIL.
    LINKAGE SECTION.
    01  DFHCOMMAREA. COPY DFH$CFIL.
    PROCEDURE DIVISION.
 1      IF EIBTRNID NOT = 'INQY'
          AND EIBTRNID NOT = 'ADDS'
          AND EIBTRNID NOT = 'UPDT' THEN GO TO ERRORS.
 2      IF EIBCALEN NOT = 0 THEN
 3         MOVE DFHCOMMAREA TO COMMAREA GO TO READ-INPUT.
 4      EXEC CICS HANDLE CONDITION MAPFAIL(MFAIL)
                                   ERROR(ERRORS) END-EXEC.
 5      EXEC CICS RECEIVE MAP('MENU') MAPSET('DFH$CGA') END-EXEC.
        IF KEYL = ZERO THEN GO TO BADLENG.
 6      MOVE KEYI TO KEYNUM.
        IF KEYI IS NOT NUMERIC THEN GO TO BADCHARS.
        MOVE LOW-VALUES TO DETAILO.
 7      IF EIBTRNID = 'ADDS' THEN
           MOVE 'FILE ADD' TO TITLEO
           MOVE 'ENTER DATA AND PRESS ENTER KEY' TO MSG3O
 8         MOVE KEYI TO NUMB IN COMMAREA, NUMBO
 9         MOVE DFHBMUNN TO AMOUNTA
           MOVE '$0000.00' TO AMOUNTO
           MOVE 7 TO COMLEN GO TO MAP-SEND.
10      EXEC CICS HANDLE CONDITION NOTFND(NOTFOUND) END-EXEC.
11      EXEC CICS READ DATASET('FILEA') INTO(FILEA) RIDFLD(KEYNUM)
                                                    END-EXEC.
        IF EIBTRNID = 'INQY' THEN
12         MOVE 'FILE INQUIRY' TO TITLEO
           MOVE 'PRESS ENTER TO CONTINUE' TO MSG3O
   *                           PROTECT ALL THE MAP FIELDS
           MOVE DFHBMPRO TO NAMEA
           MOVE DFHBMPRO TO ADDRA
13         MOVE DFHBMPRO TO PHONEA
           MOVE DFHBMPRO TO DATEA
           MOVE DFHBMPRO TO AMOUNTA
           MOVE DFHBMPRO TO COMMENTA
```

Source listing for DFH$CALL (continued)

```
14         PERFORM MAP-BUILD THRU MAP-SEND
15         EXEC CICS RETURN TRANSID('MENU') END-EXEC.
        IF EIBTRNID = 'UPDT' THEN
16         MOVE 'FILE UPDATE' TO TITLEO
           MOVE 'CHANGE FIELDS AND PRESS ENTER' TO MSG3O
17         MOVE FILEREC IN FILEA TO FILEREC IN COMMAREA
           MOVE 80 TO COMLEN.
    MAP-BUILD.
        MOVE NUMB    IN FILEA TO NUMBO.
        MOVE NAME    IN FILEA TO NAMEO.
18      MOVE ADDRX   IN FILEA TO ADDRO.
        MOVE PHONE   IN FILEA TO PHONEO.
        MOVE DATEX   IN FILEA TO DATEO.
        MOVE AMOUNT  IN FILEA TO AMOUNTO.
        MOVE COMMENT IN FILEA TO COMMENTO.
    MAP-SEND.
19      EXEC CICS SEND MAP('DETAIL') MAPSET('DFH$CGB')
                  ERASE END-EXEC.
    FIN.
        GO TO CICS-CONTROL.
20  READ-INPUT.
21      EXEC CICS HANDLE CONDITION MAPFAIL(NOTMODF) NOTFND(NOTFOUND)
                  ERROR(ERRORS) DUPREC(DUPREC) END-EXEC.
22      EXEC CICS RECEIVE MAP('DETAIL') MAPSET('DFH$CGB') END-EXEC.
        IF EIBTRNID = 'UPDT' THEN
23         EXEC CICS READ UPDATE DATASET('FILEA') INTO(FILEA)
                       RIDFLD(NUMB IN COMMAREA) END-EXEC
24         IF FILEREC IN FILEA NOT = FILEREC IN COMMAREA THEN
              MOVE 'RECORD UPDATED BY OTHER USER, TRY AGAIN' TO MSG1O
              MOVE DFHBMASB TO MSG1A
              MOVE DFHPROTN TO MSG3A
              PERFORM MAP-BUILD
              EXEC CICS SEND MAP('DETAIL') MAPSET('DFH$CGB') END-EXEC
              MOVE 80 TO COMLEN
              MOVE FILEREC IN FILEA TO FILEREC IN COMMAREA
              GO TO CICS-CONTROL
            ELSE
25            MOVE 'U' TO STAT IN FILEA
              PERFORM CHECK THRU FILE-WRITE
              MOVE 'RECORD UPDATED' TO MESSAGES GO TO MENU.
26      IF EIBTRNID = 'ADDS' THEN
              MOVE LOW-VALUES TO FILEREC IN FILEA
              MOVE 'A' TO STAT IN FILEA
              PERFORM CHECK THRU FILE-WRITE
              MOVE 'RECORD ADDED' TO MESSAGES GO TO MENU.
   *    CHECK FIELDS ADDED/UPDATED
    CHECK.
        IF NAMEI    = LOW-VALUES AND
           ADDRI    = LOW-VALUES AND
27         PHONEI   = LOW-VALUES AND
           DATEI    = LOW-VALUES AND
           AMOUNTI  = LOW-VALUES AND
           COMMENTI = LOW-VALUES GO TO NOTMODF.
   *    INSP-NAME CHANGES ALL NON-ALPHABETIC CHARACTERS THAT ARE
   *    VALID IN A NAME TO SPACES SO THAT AN ALPHABETIC TEST MAY
   *    BE DONE ON THE NAME. THE CHANGED NAME IS RETURNED IN FIELD
   *    TEMP-NAME.
        PERFORM INSP-NAME.
        IF EIBTRNID = 'ADDS' THEN
           IF TEMP-NAME NOT ALPHABETIC THEN GO TO DATA-ERROR.
```

Source listing for DFH$CALL (continued)

```
        IF EIBTRNID = 'UPDT' THEN
            IF NAMEI NOT = LOW-VALUES
            AND TEMP-NAME NOT ALPHABETIC THEN GO TO DATA-ERROR.
   *    AMOUNTI MUST BE IN FORMAT ¢NNNN.NN OR $NNNN.NN
        IF AMOUNTI = LOW-VALUE THEN GO TO FILE-WRITE.
        MOVE AMOUNTI TO AMOUNTN.
        IF (AMOUNTN1 = '¢' OR '$') AND
           (AMOUNTN25 IS NUMERIC)  AND
           (AMOUNTN6 = '.')        AND
           (AMOUNTN78 IS NUMERIC)
            THEN GO TO FILE-WRITE
        ELSE
            THEN GO TO DATA-ERROR.

    INSP-NAME.
        MOVE NAMEI TO TEMP-NAME
        INSPECT TEMP-NAME REPLACING ALL '.' BY SPACES.
        INSPECT TEMP-NAME REPLACING ALL '-' BY SPACES.
        INSPECT TEMP-NAME REPLACING ALL QUOTES BY SPACES.



    FILE-WRITE.
        IF EIBTRNID = 'ADDS' THEN MOVE NUMB IN COMMAREA TO
                                       NUMB IN FILEA.
        IF NAMEI   NOT = LOW-VALUE MOVE NAMEI   TO NAME   IN FILEA.
28      IF ADDRI   NOT = LOW-VALUE MOVE ADDRI   TO ADDRX  IN FILEA.
        IF PHONEI  NOT = LOW-VALUE MOVE PHONEI  TO PHONE  IN FILEA.
        IF DATEI   NOT = LOW-VALUE MOVE DATEI   TO DATEX  IN FILEA.
        IF AMOUNTI NOT = LOW-VALUE MOVE AMOUNTI TO AMOUNT IN FILEA.
        IF AMOUNTI = LOW-VALUE AND EIBTRNID = 'ADDS' THEN
           MOVE '$0000.00' TO AMOUNT IN FILEA.
        IF COMMENTI NOT = LOW-VALUE THEN
           MOVE COMMENTI TO COMMENT IN FILEA.
        MOVE FILEREC IN FILEA TO LOGREC.
        MOVE EIBDATE TO LDAY.
29      MOVE EIBTIME TO LTIME.
        MOVE EIBTRMID TO LTERML.
30      EXEC CICS WRITEQ TD QUEUE('LOGA') FROM(LOGA) LENGTH(92)
                                                     END-EXEC.
        IF EIBTRNID = 'UPDT' THEN
31         EXEC CICS REWRITE DATASET('FILEA') FROM(FILEA) END-EXEC
        ELSE
32         EXEC CICS WRITE DATASET('FILEA') FROM(FILEA)
                                            RIDFLD(NUMB IN COMMAREA)
                                            END-EXEC.
    DATA-ERROR.
        MOVE DFHBMASB TO MSG3A.
33      MOVE 'DATA ERROR - CORRECT AND PRESS ENTER' TO MSG3O
   *                   THE FIELD ATTRIBUTE IS SET TO
   *                   MODIFIED SO DATA WILL DISPLAY
   *                   AMOUNT IS SET NUMERIC ALSO
        MOVE DFHUNNUM TO AMOUNTA.
34      MOVE DFHBMFSE TO NAMEA, ADDRA, PHONEA, DATEA,
             COMMENTA.
35      EXEC CICS SEND MAP('DETAIL') MAPSET('DFH$CGB')
                  DATAONLY END-EXEC.
36      IF EIBTRNID = 'ADDS' THEN MOVE 7 TO COMLEN
        ELSE MOVE 80 TO COMLEN.
```

Source listing for DFH$CALL (continued)

```
    CICS-CONTROL.
37      EXEC CICS RETURN TRANSID(EIBTRNID) COMMAREA(COMMAREA)
                                           LENGTH(COMLEN) END-EXEC.
38  NOTMODF.
        MOVE 'RECORD NOT MODIFIED' TO MESSAGES.
        GO TO MENU.
    DUPREC.
        MOVE 'DUPLICATE RECORD' TO MESSAGES.
        GO TO MENU.
    BADLENG.
        MOVE 'PLEASE ENTER AN ACCOUNT NUMBER' TO MESSAGES.
        GO TO MENU.
    BADCHARS.
        MOVE 'ACCOUNT NUMBER MUST BE NUMERIC' TO MESSAGES.
        GO TO MENU.
    NOTFOUND.
        MOVE 'INVALID NUMBER - PLEASE REENTER' TO MESSAGES.
        GO TO MENU.
    MFAIL.
        MOVE 'PRESS CLEAR TO EXIT' TO MESSAGES.
        GO TO MENU.
    ERRORS.
39      EXEC CICS DUMP DUMPCODE('ERRS') END-EXEC.
        MOVE 'TRANSACTION TERMINATED' TO MESSAGES.
40  MENU.
        MOVE LOW-VALUE TO MENUO.
        MOVE DFHBMASB  TO MSGA.
        MOVE MESSAGES  TO MSGO.
41      EXEC CICS SEND MAP('MENU') MAPSET('DFH$CGA') ERASE END-EXEC.
42      EXEC CICS RETURN END-EXEC.
        GOBACK.
```

Program notes

1. The possible invoking transaction-ids are tested.
2. The length of the COMMAREA is tested. If not zero then this is the validation stage of an add or update.
3. If it has a length, the COMMAREA returned is moved to working storage in the program.
4. The program exits are set up.
5. The menu map DFH$CGA is received. The account number, if entered, is mapped into KEYI in the dsect for DFH$CGA.
6. The account number is validated and saved.
7. If program is invoked by 'ADDS', a title and command message are moved to the map area. The record key is moved to the map area and saved in COMMAREA.
8. The record key is moved to the COMMAREA and to the map area.
9. For the ADDS transaction, the amount field has the attribute byte set to numeric so only numeric data can be entered.
10. For an inquiry or update the exit for the record-not-found condition is set up.
11. The file control READ command reads the file record into the file area.
12. If program is invoked by 'INQY', a title and command message are moved to the map area.
13. All field attributes are protected.
14. The file record fields are moved to the map area, and the inquiry screen is displayed.
15. This invocation of the program terminates. The TRANSID of MENU causes the operator instruction program to be invoked when the next response is received from the terminal.
16. If program is invoked by 'UPDT' a title and command message are moved to the map area.
17. The file record is moved to the COMMAREA and the length of the COMMAREA to be returned is set up.
18. The fields from the file area are moved to the map area.
19. MAP-SEND sends the map DFH$CGB to the screen specifying that the screen is to be erased before the map is displayed.
20. Control is passed here when the test of EIBCALEN, at the beginning of the program, finds that a COMMAREA has been received. This part of the program maps in data for an add or update request, performs validation and updates FILEA.
21. The error exits are set up.

22. The RECEIVE MAP command maps in the variables from the screen.

23. If this is an update request a file control READ UPDATE reads the existing record using the number stored in COMMAREA by the last invocation of this program.

24. If the current file record is not the same as the one saved in the COMMAREA then another user has updated the record. A warning message is displayed, with fields from the record read from FILEA, for reentry of the updates.

25. The update flag is set in the record area and the message 'RECORD UPDATED' is moved to the message area ready for display of the operator instruction screen.

26. If this is an add request the add flag is set in the new record and the message 'RECORD ADDED' is moved to the message area ready for display of the operator instruction screen.

27. Any required editing steps should be inserted here. A suitable form of editing should be used to ensure valid records are placed on the file.

28. This code creates or updates the account record. Any field which has been entered is moved to the account record.

29. The record fields, the date, the time, and the terminal identification are moved to update log record area.

30. The record is written to the update log which is a transient data queue.

31. For an update request the updated account record is rewritten to FILEA.

32. For an add request the new account record is written to FILEA.

33. When a data error is detected the screen is redisplayed for errors to be corrected. An error message is moved to the map area and highlighted.

34. The modified data tag is set on for all the data fields so that all the data is received at the next RECEIVE MAP.

35. The contents of map DFH$CGB are sent to the screen. The constant information on the screen is not refreshed as a result of the use of the DATAONLY option.

36. The size of the COMMAREA is set to 7 for an add request or to 80 for an update request.

37. After the FILE ADD or FILE UPDATE screen has been displayed the program branches here to return to CICS awaiting a response from the terminal. The RETURN gives CICS the transaction identifier for the next transaction at this terminal together with a COMMAREA containing all the information that the program needs to continue the update. The COMMAREA is passed to the next invocation of this program, see note 2 on page 399.

38. These short error routines set up an error message in MESSAGES and branch to MENU to display the message is the operator instruction menu DFH$CGA.

39. If a CICS command fails with the ERROR condition or if an unknown transaction identifier is used to invoke this program, a dump is taken and the message 'TRANSACTION TERMINATED' is moved to MESSAGES for display on the operator instruction screen.

40. This code gets control when an add or update is complete. An information or error message is in MESSAGES. The operator instruction map area is cleared. The message is moved to the map area and highlighted.

41. The operator instruction map DFH$CGA is displayed on an erased screen.

42. The program terminates by returning to CICS. No transaction identifier or COMMAREA is specified.

Browse sample program (COBOL)

Description

The browse program sequentially retrieves a page or set of records for display, starting at a point in a file specified by the terminal operator.

To start a browse, type BRWS and an account number into the menu and press the enter key. If you omit the account number browsing begins at the start of the file.

Depressing the PF1 key or typing F causes retrieval of the next page or paging forward. If you wish to reexamine the previous records displayed, press PF2 or type B. This lets you page backward.

The browse program uses READNEXT to forward page to the end of the file and READPREV to backward page to the start of the file.

Source listing for DFH$CBRW

```
  *****************************************************************
  *    DFH$CBRW - CICS/VS SAMPLE FILEA BROWSE - COBOL             *
  *****************************************************************
   IDENTIFICATION DIVISION.
   PROGRAM-ID. FILECBRW.
   ENVIRONMENT DIVISION.
   DATA DIVISION.
   WORKING-STORAGE SECTION.
   77  I        PIC 999   USAGE IS COMP.
   77  MESSAGES PIC X(39) VALUE ' '.
  *    GOING FWD OR BACK... F/B, AT LO/HI END OF FILE
   77  CURROP   PIC X(1)  VALUE ' '.
   77  LASTOP   PIC X(1)  VALUE ' '.
   77  STATS    PIC X(1)  VALUE ' '.
  *                                          DATA-AREAS FOR RIDFLD
   77  RID      PIC 9(6)  VALUE IS ZERO.
  *                                          BUILDS PREV BACK PAGE
   77  RIDB     PIC 9(6)  VALUE IS ZERO.
  *                                          BUILDS NEXT FWD PAGE
   77  RIDF     PIC 9(6)  VALUE IS ZERO.
  *
  *                                          BMS STD ATTRIBUTES
                COPY DFHBMSCA.
  *                                          FILEA RECORD DESCRIPT'N
   01  FILEA.   COPY DFH$CFIL.
  *                                          GENERAL MENU MAP
                COPY DFH$CGA.
  *                                          BROWSE FILEA MAP
                COPY DFH$CGC.
  *
  *
   PROCEDURE DIVISION.
1      EXEC CICS HANDLE CONDITION
                        ERROR(ERRORS)
                        MAPFAIL (SMSG)
                        NOTFND(NOTFOUND) END-EXEC.
  *
2      EXEC CICS RECEIVE MAP('MENU') MAPSET('DFH$CGA') END-EXEC.
  *
3      EXEC CICS HANDLE AID
                        CLEAR(SMSG)
                        PF1 (PAGE-FORWARD)
                        PF2 (PAGE-BACKWARD) END-EXEC.
  *****************************************************************
  *                                SIMPLE CHECKS OF INPUT DATA    *
  *****************************************************************
4      IF KEYL NOT = ZERO THEN
          IF KEYI IS NUMERIC THEN
  *                                                  VALID INPUT
             MOVE KEYI TO RID
             MOVE KEYI TO RIDF
             MOVE KEYI TO RIDB
          ELSE
  *                                                  NOT NUMERIC
             MOVE
             'ACCOUNT NUMBER MUST BE NUMERIC'
             TO MESSAGES
             GO TO MENU
```

Source listing for DFH$CBRW (continued)

```
            ELSE
       *                                              ACCOUNT NO OMITTED
               MOVE '000000' TO RID
               MOVE '000000' TO RIDF.
       *****************************************************************
       *                               ESTABLISH START POINT          *
       *****************************************************************
     5      EXEC CICS STARTBR DATASET('FILEA')
                              RIDFLD(RID) END-EXEC.
       *
     6      IF RID NOT EQUAL '999999' THEN GO TO PAGE-FORWARD.
       *
            MOVE 'H' TO STATS.
            GO TO PAGE-BACKWARD.
       *****************************************************************
       *                               BUILD NEXT FORWARD PAGE        *
       *****************************************************************
        PAGE-FORWARD.
            MOVE 'F' TO CURROP.
       *                                                  TOP END OF FILE
     7      EXEC CICS HANDLE CONDITION
                             ENDFILE(TOOHIGH) END-EXEC.
       *                                                  RESET MAP 'C'
            MOVE LOW-VALUES TO BROWSEO.
       *                                                  RID>NEXT FPAGE
            MOVE RIDF TO RID.
            MOVE 1 TO I.
        NEXT-LINE.
       *                                                  MOVE FIELDS>MAP
     8      EXEC CICS READNEXT INTO(FILEA)
                               DATASET('FILEA')
                               RIDFLD(RID) END-EXEC.
       *                                                  READ 4 RECORDS
     9      IF I = 1 THEN
               MOVE NUMB TO NUMBER1O
               MOVE NAME TO NAME1O
               MOVE AMOUNT TO AMOUNT1O
       *                                          RIDB NEEDS EXISTING A/C
               MOVE RID TO RIDB
            ELSE IF I = 2 THEN
                    MOVE NUMB TO NUMBER2O
                    MOVE NAME TO NAME2O
                    MOVE AMOUNT TO AMOUNT2O
                 ELSE IF I = 3 THEN
                         MOVE NUMB TO NUMBER3O
                         MOVE NAME TO NAME3O
                         MOVE AMOUNT TO AMOUNT3O
                      ELSE IF I = 4 THEN
                         MOVE NUMB TO NUMBER4O
                         MOVE NAME TO NAME4O
                         MOVE AMOUNT TO AMOUNT4O.
            ADD 1 TO I.
            IF I NOT EQUAL 5 THEN GO TO NEXT-LINE.
       *                                                  RID>NEXT FPAGE
            MOVE RID TO RIDF.
    10      EXEC CICS SEND MAP('BROWSE') MAPSET('DFH$CGC')
                      ERASE END-EXEC.
       *                                                  GET NEXT RQUEST
            GO TO PROMPT.
```

Source listing for DFH$CBRW (continued)

```
   *****************************************************************
   *                                BUILD PREVIOUS BACK PAGE       *
   *****************************************************************
11  PAGE-BACKWARD.
        MOVE 'B' TO CURROP.
   *                                                  LOW END OF FILE
        EXEC CICS HANDLE CONDITION
                         ENDFILE(TOOLOW) END-EXEC.
   *                                                  RESET MAP 'C'
        MOVE LOW-VALUES TO BROWSEO.
   *                                                  RID>PREV BPAGE
        MOVE RIDB TO RID.
   *                                                  RIDF>NEXT FPAGE
        MOVE RIDB TO RIDF.
   *
        IF LASTOP EQUAL 'B' THEN GO TO PREV.
        IF STATS EQUAL 'H' THEN GO TO PREV.
        EXEC CICS READPREV INTO(FILEA)
                           DATASET('FILEA')
                           RIDFLD(RID) END-EXEC.
   *
    PREV.
        MOVE 1 TO I.
   *
    PREV-LINE.
   *                            MOVE FIELDS>MAP
        EXEC CICS READPREV INTO(FILEA)
                  DATASET('FILEA')
                  RIDFLD(RID) END-EXEC.
   *                                       READ 4 RECORDS IN
   *                                       ASCENDING ORDER
        IF I = 4 THEN
           MOVE NUMB TO NUMBER1O
           MOVE NAME TO NAME1O
           MOVE AMOUNT TO AMOUNT1O
        ELSE IF I = 3 THEN
                MOVE NUMB TO NUMBER2O
                MOVE NAME TO NAME2O
                MOVE AMOUNT TO AMOUNT2O
             ELSE IF I = 2 THEN
                     MOVE NUMB TO NUMBER3O
                     MOVE NAME TO NAME3O
                     MOVE AMOUNT TO AMOUNT3O
                  ELSE IF I = 1 THEN
                     MOVE NUMB TO NUMBER4O
                     MOVE NAME TO NAME4O
                     MOVE AMOUNT TO AMOUNT4O.
   *
        ADD 1 TO I.
        IF I NOT EQUAL 5 THEN GO TO PREV-LINE.
   *                                                  RID>NEXT BPAGE
        MOVE RID TO RIDB.
        EXEC CICS SEND MAP('BROWSE') MAPSET('DFH$CGC')
                  ERASE END-EXEC.
   *                                                  GET NEXT RQUEST
        GO TO PROMPT.
   *****************************************************************
   *                                 PROMPT FOR NEXT PAGING REQUEST*
   *****************************************************************
```

Source listing for DFH$CBRW (continued)

```
    PROMPT.
        MOVE CURROP TO LASTOP.
12      EXEC CICS RECEIVE MAP('BROWSE') MAPSET('DFH$CGC') END-EXEC.
        IF DIRI EQUAL 'F' THEN GO TO PAGE-FORWARD.
        IF DIRI EQUAL 'B' THEN GO TO PAGE-BACKWARD.
   *                                                  INVALID-RESEND
        EXEC CICS SEND MAP('BROWSE') MAPSET('DFH$CGC') END-EXEC.
        GO TO PROMPT.
   *****************************************************************
   *                                HANDLE END OF FILE CONDITIONS  *
   *****************************************************************
13  TOOHIGH.
        MOVE 'H' TO STATS.
        MOVE RID TO RIDF.
        MOVE RID TO RIDB.
        MOVE ' ' TO DIRO.
        MOVE 'HI-END OF FILE' TO MSG1O.
   *                                                  BRT+PROT ATTR
        MOVE DFHBMASB TO MSG1A.
        EXEC CICS SEND MAP('BROWSE') MAPSET('DFH$CGC')
                  ERASE END-EXEC.
        GO TO PROMPT.
14  TOOLOW.
        MOVE 'L' TO STATS.
        MOVE '000000' TO RIDF.
        MOVE '000000' TO RIDB.
        MOVE ' ' TO DIRO.
        MOVE 'LO-END OF FILE' TO MSG2O.
   *                                                  BRT+PROT ATTR
        MOVE DFHBMASB TO MSG2A.
        EXEC CICS SEND MAP('BROWSE') MAPSET('DFH$CGC')
                  ERASE END-EXEC.
        GO TO PROMPT.
   *****************************************************************
   *                                HANDLE GENERAL CONDITIONS      *
   *****************************************************************
15  NOTFOUND.
        MOVE 'END OF FILE - PLEASE RESTART' TO MESSAGES.
        GO TO MENU.
    SMSG.
        MOVE 'PRESS CLEAR TO EXIT' TO MESSAGES.
        GO TO MENU.
    ERRORS.
16      EXEC CICS DUMP DUMPCODE('ERRS') END-EXEC.
        MOVE 'TRANSACTION TERMINATED' TO MESSAGES.
   *****************************************************************
   *                                DISPLAY GENERAL MENU THEN EXIT *
   *****************************************************************
    MENU.
   *                                                  RESET MAP 'A'
        MOVE LOW-VALUE TO MENUO.
        MOVE DFHBMASB TO MSGA.
        MOVE MESSAGES TO MSGO.
17      EXEC CICS SEND MAP('MENU') MAPSET('DFH$CGA')
                  ERASE END-EXEC.
18      EXEC CICS RETURN END-EXEC.
        GOBACK.
```

Program notes

1. The error exits are set up.
2. This command maps in the account number from the operator instruction screen.
3. The exits for CLEAR, PF1 and PF2 are set up.
4. If the format of the account number is valid the number is used to set up the program's browse pointers. If no account number is entered browsing begins at the start of the file.
5. The STARTBR command establishes the browse starting point.
6. Entering the maximum value (999999) for the account number begins a backward browse from the end of file.
7. The forward browse end of file exit is set up.
8. The READNEXT reads the first record into the file area.
9. The screen is built with 4 records.
10. The screen is erased and the page is displayed at the terminal.
11. The backward browse is similar to the forward browse. Note the need for an extra READPREV when changing from forward to backward browsing.
12. When the RECEIVE command executes control will go to one of the HANDLE AID exits (see note 3) if CLEAR, PF1 or PF2 is pressed. The program explicitly tests for F or B if no exit is taken. Any other terminal response is ignored.
13. If the end of file is reached, on a READNEXT, any records read to that point are displayed, together with a highlighted message 'HI-END OF FILE'.
14. If the start of file is reached on a READPREV (backward browse) then the ENDFILE condition occurs and TOOLOW gets control. Any records read up to that point are displayed together with a highlighted message 'LO-END OF FILE'.
15. If the NOTFND condition occurs at the start browse the message 'END OF FILE - PLEASE RESTART' is moved to MESSAGES for display on the operator instruction screen.
16. In some error situations a dump is taken and the message 'TRANSACTION TERMINATED' is moved to MESSAGES for display on the operator instruction screen.
17. This code displays the operator instruction menu with a message which has been stored in MESSAGES.
18. The program terminates by returning to CICS.

Order entry sample program (COBOL)

Description

The order entry sample application program provides a data entry facility for customer orders for parts from a warehouse. Orders are recorded on a transient data queue which 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 OREN onto a blank screen and press ENTER. The order entry program displays the map DFH$CGK on the screen requesting the operator to enter order details, that is, customer number, part number, and the quantity of that part required. The customer number must be valid, that is, it must exist on FILEA. 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. A definition of the transient data queue is included in the sample destination control table listed in the *CICS/MVS Installation Guide*. The TRANSID specified in the DCT entry for L86O must be changed from AORQ to OREQ for the COBOL program to be triggered.

The trigger level may be changed using CEMT, as follows:

```
CEMT SET QUEUE(L860) TRIGGER(n)
```

where n is the destination trigger level (any integer from 0 through 32767).

Source listing for DFH$CREN

```
   *****************************************************************
   *    DFH$CREN - CICS/VS SAMPLE FILEA ORDER ENTRY - COBOL        *
   *****************************************************************
    IDENTIFICATION DIVISION.
    PROGRAM-ID. FILECREN.
    ENVIRONMENT DIVISION.
    DATA DIVISION.
    WORKING-STORAGE SECTION.
    77  ERROR-FLAG PIC 9.
    77  PRESMSG PICTURE X(20) VALUE 'PROCESSING COMPLETED'.
                 COPY DFH$CGK.
    01  FILEA.   COPY DFH$CFIL.
    01  L860.    COPY DFH$CL86.
                 COPY DFHBMSCA.
   *
    PROCEDURE DIVISION.
   *
   *                                                HANDLE CONDITIONS
 1      EXEC CICS HANDLE AID CLEAR(ENDA) END-EXEC.
 2      EXEC CICS HANDLE CONDITION MAPFAIL(MAPFAIL)
                                   NOTFND(NOTFOUND)
                                   ERROR(ERRORS) END-EXEC.
   *
   *                                                CLEAR MAP
        MOVE LOW-VALUES TO ORDERO.
   *
 3      EXEC CICS SEND MAP('ORDER') MAPSET('DFH$CGK')
                  ERASE END-EXEC.
   *****************************************************************
   *                                               PROCESS INPUT   *
   *****************************************************************
    RECEIVM.
 4      EXEC CICS RECEIVE MAP('ORDER') MAPSET('DFH$CGK') END-EXEC.
   *
        MOVE 0 TO ERROR-FLAG.
        MOVE DFHBMFSE TO CUSTNOA, PARTNOA, QUANTA.
   *****************************************************************
   *                                               CHECK DATA      *
   *****************************************************************
   *
 5      IF CUSTNOI NOT NUMERIC THEN
           MOVE DFHUNINT TO CUSTNOA MOVE 1 TO ERROR-FLAG.
   *
        IF PARTNOI NOT NUMERIC THEN
           MOVE DFHUNINT TO PARTNOA MOVE 1 TO ERROR-FLAG.
   *
        IF QUANTI NOT NUMERIC THEN
           MOVE DFHUNINT TO QUANTA MOVE 1 TO ERROR-FLAG.
   *
   *                                              DATA ERROR-REENTER
        IF ERROR-FLAG = 1 THEN
 6         MOVE DFHBMASB TO MSG2A
           EXEC CICS SEND MAP('ORDER') MAPSET('DFH$CGK') END-EXEC
           GO TO RECEIVM.
   *****************************************************************
   *                                               READ CUST RECORD*
   *****************************************************************
   *
```

Source listing for DFH$CREN (continued)

```
 7      EXEC CICS READ DATASET('FILEA') INTO(FILEA) RIDFLD(CUSTNOI)
                                                    END-EXEC.
        MOVE CUSTNOI  TO CUSTNO.
 8      MOVE PARTNOI  TO PARTNO.
        MOVE QUANTI   TO QUANTITY.
        MOVE EIBTRMID TO TERMID.
   *****************************************************************
   *                                              WRITE VALID ORDER*
   *****************************************************************
   *
 9      EXEC CICS WRITEQ TD QUEUE('L860') FROM(L860) LENGTH(22)
                                                     END-EXEC.
10      EXEC CICS SEND MAP('ORDER') MAPSET('DFH$CGK')
                  MAPONLY ERASEAUP END-EXEC.
        GO TO RECEIVM.
   *
   *****************************************************************
   *                                              PROCESS ERRORS   *
   *****************************************************************
11  NOTFOUND.
        MOVE DFHBMASB TO MSG1A.
        EXEC CICS SEND MAP('ORDER') MAPSET('DFH$CGK') END-EXEC.
        GO TO RECEIVM.
   *
12  MAPFAIL.
        MOVE LOW-VALUES TO ORDERO.
        MOVE DFHBMASB TO MSG2A.
        EXEC CICS SEND MAP('ORDER') MAPSET('DFH$CGK') END-EXEC.
        GO TO RECEIVM.
   *****************************************************************
   *                                              EXIT FROM PROGRAM*
   *****************************************************************
   *
13  ERRORS.
        MOVE 'TRANSACTION TERMINATED' TO MSG2O.
        MOVE DFHBMASB TO MSG2A.
        EXEC CICS SEND MAP('ORDER') MAPSET('DFH$CGK') END-EXEC.
        EXEC CICS DUMP DUMPCODE('ERRS') END-EXEC.
   *
14  ENDA.
        EXEC CICS SEND TEXT FROM(PRESMSG) LENGTH(20) ERASE END-EXEC.
        EXEC CICS SEND CONTROL FREEKB END-EXEC.
        EXEC CICS RETURN END-EXEC.
        GOBACK.
```

Program notes

1. The CLEAR key exit is set up.
2. The error exits are set up.
3. The screen is erased and the order entry map is displayed at the terminal.
4. This RECEIVE MAP causes a read from the terminal and maps in the customer number, part number and quantity. The program remains in virtual storage until the terminal response is received. Compare this technique with that used in the pseudoconversational inquiry/update sample program. If no data is received CICS branches to the MAPFAIL exit (note 2).
5. The order details are checked, invalid orders are redisplayed for correction. Error fields are highlighted and have MDT set on. The user should add further editing steps necessary to ensure only valid orders are accepted.
6. The error message 'DATA ERROR - REENTER' is a constant in the map load module and is sent to the terminal, with any other constant information, unless DATAONLY is specified on the SEND MAP. The message is normally dark (non-display). This instruction overrides the dark attribute and the message appears in high intensity when the SEND MAP is executed.

7. The file control READ command attempts to read the customer record from FILEA. If no record exists for the customer CICS branches to the NOTFND exit (note 2).

8. The order details are moved from the input map to the queue area.

9. The WRITEQ TD command writes the order record to a sequential file, a transient data queue.

10. The order entry map is redisplayed ready for the next order. Only the map load module is used to build the screen display, MAPONLY causes the data in the map dsect area to be ignored. ERASEAUP erases all the unprotected data on the screen, that is, the customer number, part number and quantity.

11. If there is no record for the customer on FILEA, CICS raises the NOTFND and branches here. The attribute for the customer number field is set to high intensity with MDT on and an error message 'NUMBER NOT FOUND - REENTER' is set to display in high intensity (see note 6 on page 409). The order is redisplayed for correction.

12. If no fields are entered, the MAPFAIL condition occurs. The message 'DATA ERROR - REENTER' is displayed in high intensity (see note 6 on page 409).

13. If an error occurs a dump is taken, and the message 'TRANSACTION TERMINATED' is displayed in high intensity in the data error message area. The program terminates leaving the order entry screen displayed.

14. When the CLEAR key is pressed the program terminates. The message 'PROCESSING COMPLETED' is displayed on a blank screen, the keyboard is freed and control is returned to CICS.

Order entry queue print sample program (COBOL)

Description

The order entry queue print sample program sends customer orders to a printer terminal at the warehouse. The order entry sample program, described earlier, records customer orders on a transient data queue which is read by this program.

The queue print transaction can be invoked in one of three ways:

- You can type the transaction identifier OREQ onto 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 COMPLETED' is displayed and your terminal is available for other work.
- One hour after you enter OREQ, the queue print transaction is automatically invoked by CICS interval control. In this case the terminal identifier, specified by the START, is L86O so the program prints the orders at the warehouse.
- The queue print transaction is 'triggered' 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 will print the orders. The TRANSID specified in the DCT entry for L86O must be changed from AORQ to OREQ for the COBOL program to be triggered. The trigger level may be changed using CEMT, as follows:

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

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/update sample program. When all the orders have been processed, or if there were no orders to process, the message 'ORDER QUEUE IS EMPTY' is printed at the warehouse.

Source listing for DFH$CCOM

```
   *****************************************************************
   *    DFH$CCOM - CICS/VS SAMPLE FILEA ORDER ENTRY QUEUE PRINT    *
   *****************************************************************
    IDENTIFICATION DIVISION.
    PROGRAM-ID. FILECCOM.
    ENVIRONMENT DIVISION.
    DATA DIVISION.
    WORKING-STORAGE SECTION.
    77  Q-LENGTH      PIC 9(4)  COMP VALUE 22.
    77  PRESMSG       PIC X(20) VALUE 'PROCESSING COMPLETED'.
    01  LOGORD.
        02  LOGTIME.
            03  LDAY  PIC S9(7) COMP-3.
            03  LTIME PIC S9(7) COMP-3.
        02  LITEM     PIC X(22).
        02  COMMENT   PIC X(11) VALUE 'ORDER ENTRY'.
        02  FILLER    PIC X(51) VALUE SPACES.
                 COPY DFH$CGL.
    01  FILEA.   COPY DFH$CFIL.
    01  L860.    COPY DFH$CL86.
                 COPY DFHBMSCA.
    PROCEDURE DIVISION.
 1      EXEC CICS HANDLE CONDITION ERROR(ERRORS)
                                   QZERO(ENDA) END-EXEC.
 2      IF EIBTRMID NOT = 'L860' THEN
           GO TO ASK-TIME.
        MOVE LOW-VALUES TO PRINTO.
    Q-READ.
 3      EXEC CICS READQ TD INTO(L860) LENGTH(Q-LENGTH)
                           QUEUE('L860') END-EXEC.
    MAP-BUILD.
 4      EXEC CICS READ DATASET('FILEA') INTO(FILEA) RIDFLD(CUSTNO)
                                                    END-EXEC.
 5      IF AMOUNT  > '$0100.00' THEN
           MOVE ADDRX    TO ADDRO
           MOVE NAME     TO NAMO
 6         MOVE PARTNO   TO PARTO
           MOVE CUSTNO   TO NUMBO
           MOVE ITEM     TO LITEM
           MOVE QUANTITY TO QUANTO
 7         EXEC CICS SEND MAP('PRINT') MAPSET('DFH$CGL')
                     ERASE PRINT L80 END-EXEC
        ELSE
           MOVE EIBDATE  TO LDAY
 8         MOVE EIBTIME  TO LTIME
           MOVE ITEM     TO LITEM
 9         EXEC CICS WRITEQ TD QUEUE('LOGA')
                               FROM(LOGORD) LENGTH(92) END-EXEC.
           GO TO Q-READ.
    ERRORS.
10         EXEC CICS DUMP DUMPCODE('ERRS') END-EXEC.
           GO TO FIN.
    ENDA.
        MOVE LOW-VALUES  TO PRINTO
11      MOVE 'ORDER QUEUE IS EMPTY' TO TITLEO
        EXEC CICS SEND MAP('PRINT') MAPSET('DFH$CGL')
                  DATAONLY ERASE PRINT L80 END-EXEC.
    ASK-TIME.
   *                           IF THE COMMENT DELIMITER IS
```

Source listing for DFH$CCOM (continued)

```
      *                       REMOVED FROM THE NEXT TWO COBOL
      *                       STATEMENTS, THE APPLICATION WILL
      *                       BE RESTARTED IN AN HOUR IF THE
      *                       TIME OF DAY RIGHT NOW IS NOT LATER
      *                       THAN 1400 HRS.
      *                       IF THE CODE IS LEFT UNCHANGED THE
      *                       APPLICATION WILL BE RESTARTED
      *                       UNCONDITIONALLY AFTER AN HOUR HAS
      *                       ELAPSED
      *    EXEC CICS ASKTIME END-EXEC.
      *    IF EIBTIME NOT > 140000 THEN
 12           EXEC CICS START TRANSID('OREQ') INTERVAL(10000)
                              TERMID('L860') END-EXEC.
        FIN.
 13        EXEC CICS SEND TEXT FROM(PRESMSG) LENGTH(20) ERASE END-EXEC
           EXEC CICS SEND CONTROL FREEKB END-EXEC.
           EXEC CICS RETURN END-EXEC.
           GOBACK.
```

Program notes

1. The error exits are set up.
2. The terminal-id is tested to see whether this transaction was started from a terminal or at the printer.
3. A queue item (customer order) is read into the program.
4. The file control READ command reads the record into a record area so that the amount may be checked.
5. The amount (bank balance) is tested. If it is over $100 then the order is acceptable, otherwise the order is rejected. This test is only a suggestion; a suitable form of editing should be inserted here to ensure valid orders are sent to the warehouse.
6. The order details are moved to the map area for DFH$CGL.
7. The order map is sent to the printer terminal at the warehouse.
8. The current date and time, and details of the rejected order, are moved to a log record area.
9. The WRITEQ TD command writes details of the rejected order to LOGA, a transient data queue.
10. If the ERROR condition occurs on any CICS command a dump is taken and the program terminates.
11. When the queue is empty, the message 'ORDER QUEUE IS EMPTY' is moved to the map area which is then sent to the printer terminal at the warehouse.
12. The START command starts the OREQ transaction (this program), after a one hour delay, with a terminal identifier of L86O. (The time interval could be changed, for demonstration purposes, by changing the INTERVAL value). If the comment delimiters are removed from the two preceding statements, EIBTIME is refreshed and, if the time is before 1400 hours, the transaction is started in one hour. If the comment delimiters are not removed, the transaction is started unconditionally in one hour.
13. The message 'PROCESSING COMPLETED' is sent to the terminal associated with this invocation of OREQ, either the printer at the warehouse or the screen on which OREQ was entered. The program terminates by returning control to CICS.

Low balance report sample program (COBOL)

Description

The low balance report sample program produces a report that lists all entries in the data set FILEA for which the amount is less than or equal to $50.00.

The program illustrates page building techniques and the use of the terminal paging facilities of BMS.

The transaction is invoked by entering REPT onto a clear screen. The program does a sequential scan through the file selecting each entry that obeys the search criterion.

The pages are built from four maps which comprise map set DFH$CGD, using the paging option so that the data is not displayed immediately but instead is stored for later retrieval. The HEADING map is inserted at the head of each page. The detail map (DFH$CGD) is written repeatedly until the overflow condition occurs. The FOOTING map is then written at the foot of the page and the HEADING map written at the top of the next page. The command to write the detail map that caused overflow is then repeated. When all the data has been written the FINAL map is written at the bottom of the last page and the transaction terminated.

The terminal operator then enters paging commands to display the data, clearing the screen before entering each paging command.

Source listing for DFH$CREP

```
   *****************************************************************
   *    DFH$CREP - CICS/VS SAMPLE FILEA LOW BALANCE INQUIRY        *
   *****************************************************************
    IDENTIFICATION DIVISION.
    PROGRAM-ID. FILECREP.
    ENVIRONMENT DIVISION.
    DATA DIVISION.
    WORKING-STORAGE SECTION.
    77  LOWLIM       PIC X(8)  VALUE '$0050.00'.
    77  KEYNUM       PIC 9(6)  VALUE 0.
   *                 THE INPUT AREA FOR KEYED DATA AND THE
   *                 MAXIMUM LENGTH OF KEYED DATA FOLLOW.
   *                 IN PRACTICE THE OPERATOR WILL ONLY
   *                 PRESS ENTER.
    77  TERMDATA     PIC X(1).
    77  TERMLENG     PIC S9(4) COMP.
    77  PAGEN        PIC 9(3)  VALUE 1.
    77  OPINSTR      PIC X(52) VALUE 'PRESS THE ENTER KEY AND FOLLOW
   -                                 'WITH PAGING COMMANDS.'.
                 COPY DFH$CGD.
    01  FILEA.   COPY DFH$CFIL.
    PROCEDURE DIVISION.
 1      EXECUTE CICS HANDLE CONDITION ERROR(ERRORS)
             OVERFLOW(OFLOW) ENDFILE(ENDFILE)
             LENGERR(ENDTASK) END-EXEC
        MOVE LOW-VALUE TO PAGENA
 2      MOVE PAGEN     TO PAGENO
 3      EXEC CICS SEND MAP('HEADING') MAPSET('DFH$CGD') ACCUM
             PAGING ERASE END-EXEC
 4      EXEC CICS STARTBR DATASET('FILEA') RIDFLD(KEYNUM) END-EXEC.
    REPEAT.
 5      EXEC CICS READNEXT INTO(FILEA) RIDFLD(KEYNUM)
                           DATASET('FILEA') END-EXEC
        MOVE AMOUNT    TO AMOUNTO
 6      IF AMOUNTO GREATER THAN LOWLIM GO TO REPEAT.
        MOVE LOW-VALUE TO LINEO
        MOVE AMOUNT    TO AMOUNTO
 7      MOVE NUMB      TO NUMBERO
        MOVE NAME      TO NAMEO
 8      EXEC CICS SEND MAP('LINE') MAPSET('DFH$CGD')
                                 ACCUM PAGING END-EXEC
        GO TO REPEAT.
    ENDFILE.
 9      EXEC CICS SEND MAP('FINAL') MAPSET('DFH$CGD')
               MAPONLY ACCUM PAGING END-EXEC
10      EXEC CICS SEND PAGE END-EXEC
11      EXEC CICS SEND TEXT FROM(OPINSTR) LENGTH(52) ERASE END-EXEC
12      EXEC CICS ENDBR DATASET('FILEA') END-EXEC
   *                 A RECEIVE COMMAND IS ISSUED TO GIVE THE
   *                 TERMINAL OPERATOR A CHANCE TO READ THE
   *                 PROMPTING MESSAGE.
   *
   *                 THE TRANSACTION WILL TERMINATE WHEN THE
   *                 OPERATOR PRESSES THE ENTER KEY.
   *
   *                 PAGING COMMANDS CAN THEN BE ISSUED
   *
   *                 NO HARM IS DONE IF THE OPERATOR TYPES IN
   *                 DATA BEFORE PRESSING THE ENTER KEY.
```

Source listing for DFH$CREP (continued)

```
13      EXEC CICS RECEIVE INTO(TERMDATA) LENGTH(TERMLENG) END-EXEC.
    ENDTASK.
14      EXEC CICS RETURN END-EXEC.
        GOBACK.
    ERRORS.
15      EXEC CICS HANDLE CONDITION ERROR END-EXEC
        EXEC CICS PURGE MESSAGE END-EXEC
        EXEC CICS ABEND ABCODE('ERRS') END-EXEC.
     OFLOW.
16      EXEC CICS SEND MAP('FOOTING') MAPSET('DFH$CGD')
                          MAPONLY ACCUM PAGING END-EXEC
        ADD 1     TO PAGEN
        MOVE PAGEN TO PAGENO
17      EXEC CICS SEND MAP('HEADING') MAPSET('DFH$CGD')
                          ACCUM PAGING ERASE END-EXEC.
18      EXEC CICS SEND MAP('LINE') MAPSET('DFH$CGD')
                                 ACCUM PAGING END-EXEC
        GO TO REPEAT.
```

Program notes

1. The program exits are set up.
2. A page number of 1 is moved to the heading map.
3. This BMS command sets up the heading in the page build operation. BMS builds the pages in temporary storage.
4. The STARTBR command sets up the file browse to begin at the first record with a key equal to or greater than the RIDFLD, in this case the first record on file.
5. This command reads the next customer record from FILEA.
6. The search criterion for creating the report is that the customer has a bank balance which is $50 or less.
7. Fields are moved from the selected customer record to the map area for the detail line.
8. The customer detail map is set up for subsequent paging.
9. When the ENDFILE condition is raised, the last map is sent to BMS.
10. The SEND PAGE command makes all the pages of the report available for paging, at the terminal, when the current transaction terminates.
11. A message is sent to the terminal. This message will be displayed before the pages of the low balance report.
12. The file browse operation is terminated.
13. The RECEIVE MAP command reads from the terminal and allows the terminal operator to read the prompting message before the first page of the report is displayed.
14. The program ends, the first page of the report will now be displayed.
15. If the ERROR condition occurs on a CICS command this routine gains control. Handling of the ERROR condition is suppressed, any data sent to BMS is purged and the program terminates abnormally with a transaction dump.
16. If the OVERFLOW condition occurs, when a detail line is sent to BMS, CICS branches here. This routine completes the current page and starts the next one. This BMS command sets up the footing for the current page.
17. This BMS command sets up the heading for the next page.
18. This BMS command resends the detail line which caused the OVERFLOW condition.

Maps and screen layouts for COBOL sample programs

The preceding sample programs assume that the following map sets have been cataloged with names the same as the map names.

The names of the source maps are all of the form DFH$CMx, whereas output generated by the assembly of maps is in the form DFH$CGx. Use different names for the map source and the generated dsect only if you wish to store both in the same source library.

DFH$CGA map definition

```
         TITLE 'FILEA - MAP FOR OPERATOR INSTRUCTIONS - COBOL'
DFH$CGA  DFHMSD TYPE=&SYSPARM,MODE=INOUT,CTRL=(FREEKB,FRSET),          *
               LANG=COBOL,TIOAPFX=YES,EXTATT=MAPONLY,COLOR=BLUE
MENU     DFHMDI SIZE=(12,40)
         DFHMDF POS=(1,10),LENGTH=21,INITIAL='OPERATOR INSTRUCTIONS',  *
               HILIGHT=UNDERLINE
         DFHMDF POS=(3,1),LENGTH=29,INITIAL='OPERATOR INSTR - ENTER MEN*
               U'
         DFHMDF POS=(4,1),LENGTH=38,INITIAL='FILE INQUIRY   - ENTER INQ*
               Y AND NUMBER'
         DFHMDF POS=(5,1),LENGTH=38,INITIAL='FILE BROWSE    - ENTER BRW*
               S AND NUMBER'
         DFHMDF POS=(6,1),LENGTH=38,INITIAL='FILE ADD       - ENTER ADD*
               S AND NUMBER'
         DFHMDF POS=(7,1),LENGTH=38,INITIAL='FILE UPDATE    - ENTER UPD*
               T AND NUMBER'
MSG      DFHMDF POS=(11,1),LENGTH=39,INITIAL='PRESS CLEAR TO EXIT'
         DFHMDF POS=(12,1),LENGTH=18,INITIAL='ENTER TRANSACTION:'
         DFHMDF POS=(12,20),LENGTH=4,ATTRB=IC,COLOR=GREEN,             *
               HILIGHT=REVERSE
         DFHMDF POS=(12,25),LENGTH=6,INITIAL='NUMBER'
KEY      DFHMDF POS=(12,32),LENGTH=6,ATTRB=NUM,COLOR=GREEN,            *
               HILIGHT=REVERSE
         DFHMDF POS=(12,39),LENGTH=1
         DFHMSD TYPE=FINAL
         END
```

The symbolic storage definition produced as a result of the above statements would be as follows:

DSECT generated by DFH$CGA

```
    01  MENUI.
        02  FILLER PIC X(12).
        02  MSGL    COMP  PIC  S9(4).
        02  MSGF    PICTURE X.
        02  FILLER REDEFINES MSGF.
          03 MSGA    PICTURE X.
        02  MSGI  PIC X(39).
        02  KEYL    COMP  PIC  S9(4).
        02  KEYF    PICTURE X.
        02  FILLER REDEFINES KEYF.
          03 KEYA    PICTURE X.
        02  KEYI  PIC X(6).
    01  MENUO REDEFINES MENUI.
        02  FILLER PIC X(12).
        02  FILLER PICTURE X(3).
        02  MSGO  PIC X(39).
        02  FILLER PICTURE X(3).
        02  KEYO  PIC X(6).
```

DFH$CGA screen layout

```
          +OPERATOR INSTRUCTIONS

+OPERATOR INSTR  ENTER MENU
+FILE INQUIRY    ENTER INQY AND NUMBER
+FILE BROWSE     ENTER BRWS AND NUMBER
+FILE ADD        ENTER ADDS AND NUMBER
+FILE UPDATE     ENTER UPDT AND NUMBER



+PRESS CLEAR TO EXIT
+ENTER TRANSACTION:+XXXX+NUMBER+XXXXXX+
```

DFH$CGB map definition

```
         TITLE 'FILEA - MAP FOR FILE INQUIRY/UPDATE - COBOL'
DFH$CGB  DFHMSD TYPE=&SYSPARM,MODE=INOUT,CTRL=(FREEKB,FRSET),          *
               LANG=COBOL,TIOAPFX=YES,EXTATT=MAPONLY
DETAIL   DFHMDI SIZE=(12,40)
TITLE    DFHMDF POS=(1,15),LENGTH=12
         DFHMDF POS=(3,1),LENGTH=8,INITIAL='NUMBER:',COLOR=BLUE
NUMB     DFHMDF POS=(3,10),LENGTH=6
         DFHMDF POS=(3,17),LENGTH=1
         DFHMDF POS=(4,1),LENGTH=8,INITIAL='NAME:   ',COLOR=BLUE
NAME     DFHMDF POS=(4,10),LENGTH=20,ATTRB=(UNPROT,IC)
         DFHMDF POS=(4,31),LENGTH=1
         DFHMDF POS=(5,1),LENGTH=8,INITIAL='ADDRESS:',COLOR=BLUE
ADDR     DFHMDF POS=(5,10),LENGTH=20,ATTRB=UNPROT
         DFHMDF POS=(5,31),LENGTH=1
         DFHMDF POS=(6,1),LENGTH=8,INITIAL='PHONE:   ',COLOR=BLUE
PHONE    DFHMDF POS=(6,10),LENGTH=8,ATTRB=UNPROT
         DFHMDF POS=(6,19),LENGTH=1
         DFHMDF POS=(7,1),LENGTH=8,INITIAL='DATE:    ',COLOR=BLUE
DATE     DFHMDF POS=(7,10),LENGTH=8,ATTRB=UNPROT
         DFHMDF POS=(7,19),LENGTH=1
         DFHMDF POS=(8,1),LENGTH=8,INITIAL='AMOUNT:  ',COLOR=BLUE
AMOUNT   DFHMDF POS=(8,10),LENGTH=8,ATTRB=NUM
         DFHMDF POS=(8,19),LENGTH=1
         DFHMDF POS=(9,1),LENGTH=8,INITIAL='COMMENT:',COLOR=BLUE
COMMENT  DFHMDF POS=(9,10),LENGTH=9,ATTRB=UNPROT
         DFHMDF POS=(9,20),LENGTH=1
MSG1     DFHMDF POS=(11,1),LENGTH=39
MSG3     DFHMDF POS=(12,1),LENGTH=39
         DFHMSD TYPE=FINAL
         END
```

The symbolic storage definition produced as a result of the above statements would be as follows:

DSECT generated by DFH$CGB

```
01  DETAILI.
    02  FILLER PIC X(12).
    02  TITLEL    COMP  PIC  S9(4).
    02  TITLEF    PICTURE X.
    02  FILLER REDEFINES TITLEF.
      03 TITLEA    PICTURE X.
    02  TITLEI  PIC X(12).
    02  NUMBL    COMP  PIC  S9(4).
    02  NUMBF    PICTURE X.
    02  FILLER REDEFINES NUMBF.
      03 NUMBA    PICTURE X.
    02  NUMBI  PIC X(6).
    02  NAMEL    COMP  PIC  S9(4).
    02  NAMEF    PICTURE X.
    02  FILLER REDEFINES NAMEF.
      03 NAMEA    PICTURE X.
    02  NAMEI  PIC X(20).
    02  ADDRL    COMP  PIC  S9(4).
    02  ADDRF    PICTURE X.
    02  FILLER REDEFINES ADDRF.
      03 ADDRA    PICTURE X.
    02  ADDRI  PIC X(20).
    02  PHONEL    COMP  PIC  S9(4).
    02  PHONEF    PICTURE X.
    02  FILLER REDEFINES PHONEF.
      03 PHONEA    PICTURE X.
    02  PHONEI  PIC X(8).
    02  DATEL    COMP  PIC  S9(4).
    02  DATEF    PICTURE X.
    02  FILLER REDEFINES DATEF.
      03 DATEA    PICTURE X.
    02  DATEI  PIC X(8).
    02  AMOUNTL    COMP  PIC  S9(4).
    02  AMOUNTF    PICTURE X.
    02  FILLER REDEFINES AMOUNTF.
      03 AMOUNTA    PICTURE X.
    02  AMOUNTI  PIC X(8).
    02  COMMENTL    COMP  PIC  S9(4).
    02  COMMENTF    PICTURE X.
    02  FILLER REDEFINES COMMENTF.
      03 COMMENTA    PICTURE X.
    02  COMMENTI  PIC X(9).
    02  MSG1L    COMP  PIC  S9(4).
    02  MSG1F    PICTURE X.
    02  FILLER REDEFINES MSG1F.
      03 MSG1A    PICTURE X.
    02  MSG1I  PIC X(39).
    02  MSG3L    COMP  PIC  S9(4).
    02  MSG3F    PICTURE X.
    02  FILLER REDEFINES MSG3F.
      03 MSG3A    PICTURE X.
    02  MSG3I  PIC X(39).
01  DETAILO REDEFINES DETAILI.
    02  FILLER PIC X(12).
    02  FILLER PICTURE X(3).
    02  TITLEO  PIC X(12).
    02  FILLER PICTURE X(3).
```

DSECT generated by DFH$CGB (continued)

```
           02  NUMBO  PIC X(6).
           02  FILLER PICTURE X(3).
           02  NAMEO  PIC X(20).
           02  FILLER PICTURE X(3).
           02  ADDRO  PIC X(20).
           02  FILLER PICTURE X(3).
           02  PHONEO  PIC X(8).
           02  FILLER PICTURE X(3).
           02  DATEO  PIC X(8).
           02  FILLER PICTURE X(3).
           02  AMOUNTO  PIC X(8).
           02  FILLER PICTURE X(3).
           02  COMMENTO  PIC X(9).
           02  FILLER PICTURE X(3).
           02  MSG1O  PIC X(39).
           02  FILLER PICTURE X(3).
           02  MSG3O  PIC X(39).
```

DFH$CGB screen layout

```
         +XXXXXXXXXXXX

+NUMBER: +XXXXXX+
+NAME:   +XXXXXXXXXXXXXXXXXXXX+
+ADDRESS:+XXXXXXXXXXXXXXXXXXXX+
+PHONE:  +XXXXXXXX+
+DATE:   +XXXXXXXX+
+AMOUNT: +XXXXXXXX+
+COMMENT:+XXXXXXXXX+

+XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
+XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

DFH$CGC map definition

```
         TITLE 'FILEA - MAP FOR FILE BROWSE - COBOL'
DFH$CGC  DFHMSD TYPE=&SYSPARM,MODE=INOUT,CTRL=(FREEKB,FRSET),          *
               LANG=COBOL,TIOAPFX=YES,EXTATT=MAPONLY
BROWSE   DFHMDI SIZE=(12,40)
DIR      DFHMDF POS=(1,1),LENGTH=1,ATTRB=IC
         DFHMDF POS=(1,3),LENGTH=1
         DFHMDF POS=(1,15),LENGTH=11,INITIAL='FILE BROWSE',            *
               COLOR=BLUE,HILIGHT=UNDERLINE
         DFHMDF POS=(3,1),LENGTH=6,INITIAL='NUMBER',COLOR=BLUE
         DFHMDF POS=(3,17),LENGTH=4,INITIAL='NAME',COLOR=BLUE
         DFHMDF POS=(3,32),LENGTH=6,INITIAL='AMOUNT',COLOR=BLUE
NUMBER1  DFHMDF POS=(4,1),LENGTH=6
NAME1    DFHMDF POS=(4,9),LENGTH=20
AMOUNT1  DFHMDF POS=(4,30),LENGTH=8
NUMBER2  DFHMDF POS=(5,1),LENGTH=6
NAME2    DFHMDF POS=(5,9),LENGTH=20
AMOUNT2  DFHMDF POS=(5,30),LENGTH=8
NUMBER3  DFHMDF POS=(6,1),LENGTH=6
NAME3    DFHMDF POS=(6,9),LENGTH=20
AMOUNT3  DFHMDF POS=(6,30),LENGTH=8
NUMBER4  DFHMDF POS=(7,1),LENGTH=6
NAME4    DFHMDF POS=(7,9),LENGTH=20
AMOUNT4  DFHMDF POS=(7,30),LENGTH=8
MSG0     DFHMDF POS=(10,1),LENGTH=39,COLOR=BLUE,                       *
               INITIAL='PRESS CLEAR TO END BROWSE OPERATION'
MSG1     DFHMDF POS=(11,1),LENGTH=39,COLOR=BLUE,                       *
               INITIAL='PRESS PF1 OR TYPE F TO PAGE FORWARD'
MSG2     DFHMDF POS=(12,1),LENGTH=39,COLOR=BLUE,                       *
               INITIAL='PRESS PF2 OR TYPE B TO PAGE BACKWARD'
         DFHMSD TYPE=FINAL
         END
```

The symbolic storage definition produced as a result of the above statements would be as follows:

DSECT generated by DFH$CGC

```
01  BROWSEI.
    02  FILLER PIC X(12).
    02  DIRL    COMP  PIC  S9(4).
    02  DIRF    PICTURE X.
    02  FILLER REDEFINES DIRF.
      03 DIRA    PICTURE X.
    02  DIRI  PIC X(1).
    02  NUMBER1L    COMP  PIC  S9(4).
    02  NUMBER1F    PICTURE X.
    02  FILLER REDEFINES NUMBER1F.
      03 NUMBER1A    PICTURE X.
    02  NUMBER1I  PIC X(6).
    02  NAME1L    COMP  PIC  S9(4).
    02  NAME1F    PICTURE X.
    02  FILLER REDEFINES NAME1F.
      03 NAME1A    PICTURE X.
    02  NAME1I  PIC X(20).
    02  AMOUNT1L    COMP  PIC  S9(4).
    02  AMOUNT1F    PICTURE X.
    02  FILLER REDEFINES AMOUNT1F.
      03 AMOUNT1A    PICTURE X.
    02  AMOUNT1I  PIC X(8).
    02  NUMBER2L    COMP  PIC  S9(4).
    02  NUMBER2F    PICTURE X.
    02  FILLER REDEFINES NUMBER2F.
      03 NUMBER2A    PICTURE X.
    02  NUMBER2I  PIC X(6).
    02  NAME2L    COMP  PIC  S9(4).
    02  NAME2F    PICTURE X.
    02  FILLER REDEFINES NAME2F.
      03 NAME2A    PICTURE X.
    02  NAME2I  PIC X(20).
    02  AMOUNT2L    COMP  PIC  S9(4).
    02  AMOUNT2F    PICTURE X.
    02  FILLER REDEFINES AMOUNT2F.
      03 AMOUNT2A    PICTURE X.
    02  AMOUNT2I  PIC X(8).
    02  NUMBER3L    COMP  PIC  S9(4).
    02  NUMBER3F    PICTURE X.
    02  FILLER REDEFINES NUMBER3F.
      03 NUMBER3A    PICTURE X.
    02  NUMBER3I  PIC X(6).
    02  NAME3L    COMP  PIC  S9(4).
    02  NAME3F    PICTURE X.
    02  FILLER REDEFINES NAME3F.
      03 NAME3A    PICTURE X.
    02  NAME3I  PIC X(20).
    02  AMOUNT3L    COMP  PIC  S9(4).
    02  AMOUNT3F    PICTURE X.
    02  FILLER REDEFINES AMOUNT3F.
      03 AMOUNT3A    PICTURE X.
    02  AMOUNT3I  PIC X(8).
    02  NUMBER4L    COMP  PIC  S9(4).
    02  NUMBER4F    PICTURE X.
    02  FILLER REDEFINES NUMBER4F.
      03 NUMBER4A    PICTURE X.
    02  NUMBER4I  PIC X(6).
```

DSECT generated by DFH$CGC (continued)

```
        02  NAME4L    COMP  PIC  S9(4).
        02  NAME4F    PICTURE X.
        02  FILLER REDEFINES NAME4F.
          03 NAME4A    PICTURE X.
        02  NAME4I  PIC X(20).
        02  AMOUNT4L    COMP  PIC  S9(4).
        02  AMOUNT4F    PICTURE X.
        02  FILLER REDEFINES AMOUNT4F.
          03 AMOUNT4A    PICTURE X.
        02  AMOUNT4I  PIC X(8).
        02  MSG0L    COMP  PIC  S9(4).
        02  MSG0F    PICTURE X.
    01  BROWSEI.
        02  FILLER REDEFINES MSG0F.
          03 MSG0A    PICTURE X.
        02  MSG0I  PIC X(39).
        02  MSG1L    COMP  PIC  S9(4).
        02  MSG1F    PICTURE X.
        02  FILLER REDEFINES MSG1F.
          03 MSG1A    PICTURE X.
        02  MSG1I  PIC X(39).
        02  MSG2L    COMP  PIC  S9(4).
        02  MSG2F    PICTURE X.
        02  FILLER REDEFINES MSG2F.
          03 MSG2A    PICTURE X.
        02  MSG2I  PIC X(39).
    01  BROWSEO REDEFINES BROWSEI.
        02  FILLER PIC X(12).
        02  FILLER PICTURE X(3).
        02  DIRO  PIC X(1).
        02  FILLER PICTURE X(3).
        02  NUMBER1O  PIC X(6).
        02  FILLER PICTURE X(3).
        02  NAME1O  PIC X(20).
        02  FILLER PICTURE X(3).
        02  AMOUNT1O  PIC X(8).
        02  FILLER PICTURE X(3).
        02  NUMBER2O  PIC X(6).
        02  FILLER PICTURE X(3).
        02  NAME2O  PIC X(20).
        02  FILLER PICTURE X(3).
        02  AMOUNT2O  PIC X(8).
        02  FILLER PICTURE X(3).
        02  NUMBER3O  PIC X(6).
        02  FILLER PICTURE X(3).
        02  NAME3O  PIC X(20).
        02  FILLER PICTURE X(3).
        02  AMOUNT3O  PIC X(8).
        02  FILLER PICTURE X(3).
        02  NUMBER4O  PIC X(6).
        02  FILLER PICTURE X(3).
        02  NAME4O  PIC X(20).
        02  FILLER PICTURE X(3).
        02  AMOUNT4O  PIC X(8).
        02  FILLER PICTURE X(3).
        02  MSG0O  PIC X(39).
        02  FILLER PICTURE X(3).
        02  MSG1O  PIC X(39).
        02  FILLER PICTURE X(3).
        02  MSG2O  PIC X(39).
```

DFH$CGC screen layout

```
             +FILE BROWSE

+NUMBER         +NAME         +AMOUNT
+XXXXXX +XXXXXXXXXXXXXXXXXXXX+XXXXXX
+XXXXXX +XXXXXXXXXXXXXXXXXXXX+XXXXXX
+XXXXXX +XXXXXXXXXXXXXXXXXXXX+XXXXXX
+XXXXXX +XXXXXXXXXXXXXXXXXXXX+XXXXXX




+PRESS CLEAR TO END BROWSE OPERATION
+PRESS PF1 OR TYPE F TO PAGE FORWARD
+PRESS PF2 OR TYPE B TO PAGE BACKWARD
```

DFH$CGD map definition

```
         TITLE 'FILEA - MAPSET FOR LOW BALANCE REPORT - COBOL'
DFH$CGD  DFHMSD TYPE=&SYSPARM,MODE=OUT,CTRL=(FREEKB,FRSET),            *
               LANG=COBOL,STORAGE=AUTO,EXTATT=MAPONLY,COLOR=BLUE
LINE     DFHMDI SIZE=(1,40),COLOR=GREEN
NUMBER   DFHMDF POS=(1,1),LENGTH=6
NAME     DFHMDF POS=(1,9),LENGTH=20
AMOUNT   DFHMDF POS=(1,30),LENGTH=8
HEADING  DFHMDI SIZE=(3,40),HEADER=YES
         DFHMDF POS=(1,5),LENGTH=18,INITIAL='LOW BALANCE REPORT',      *
               HILIGHT=UNDERLINE
         DFHMDF POS=(1,30),LENGTH=4,INITIAL='PAGE'
PAGEN    DFHMDF POS=(1,35),LENGTH=3
         DFHMDF POS=(3,1),LENGTH=6,INITIAL='NUMBER'
         DFHMDF POS=(3,17),LENGTH=4,INITIAL='NAME'
         DFHMDF POS=(3,32),LENGTH=6,INITIAL='AMOUNT'
FOOTING  DFHMDI SIZE=(2,40),TRAILER=YES,JUSTIFY=LAST
         DFHMDF POS=(2,1),LENGTH=38,                                   *
               INITIAL='PRESS CLEAR AND TYPE P/N TO SEE PAGE N'
FINAL    DFHMDI SIZE=(2,40),TRAILER=YES,JUSTIFY=LAST
         DFHMDF POS=(2,10),LENGTH=14,INITIAL='END OF REPORT.'
         DFHMSD TYPE=FINAL
         END
```

The symbolic storage definition produced as a result of the above statements would be as follows:

DSECT generated by DFH$CGD

```
    01  LINEO.
        02  FILLER PIC X(12).
        02  FILLER PICTURE X(2).
        02  NUMBERA    PICTURE X.
        02  NUMBERO  PIC X(6).
        02  FILLER PICTURE X(2).
        02  NAMEA    PICTURE X.
        02  NAMEO  PIC X(20).
        02  FILLER PICTURE X(2).
        02  AMOUNTA    PICTURE X.
        02  AMOUNTO  PIC X(8).
    01  HEADINGO.
        02  FILLER PIC X(12).
        02  FILLER PICTURE X(2).
        02  PAGENA    PICTURE X.
        02  PAGENO  PIC X(3).
    01  FOOTINGO.
        02  FILLER PIC X(12).
    01  FINALO.
        02  FILLER PIC X(12).
```

DFH$CGD screen layout

```
    +LOW BALANCE REPORT      +PAGE+XXX

+NUMBER         +NAME          +AMOUNT
+XXXXXX+XXXXXXXXXXXXXXXXXXXX +XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX +XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX +XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX +XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX +XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX +XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX +XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX +XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX +XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX +XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX +XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX +XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX +XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX +XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX +XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX +XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX +XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX +XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX +XXXXXXXX

+PRESS CLEAR AND TYPE P/N TO SEE PAGE N
```

DFH$CGK map definition

```
         TITLE 'FILEA - MAP FOR ORDER ENTRY - COBOL'
DFH$CGK  DFHMSD TYPE=&SYSPARM,MODE=INOUT,CTRL=(FREEKB,FRSET),          *
               TIOAPFX=YES,LANG=COBOL,EXTATT=MAPONLY
ORDER    DFHMDI SIZE=(12,40)
         DFHMDF POS=(01,10),LENGTH=11,ATTRB=(BRT,ASKIP),               *
               INITIAL='ORDER ENTRY',COLOR=BLUE,HILIGHT=UNDERLINE
MSG1     DFHMDF POS=(03,04),LENGTH=26,ATTRB=(DRK,ASKIP),               *
               INITIAL='NUMBER NOT FOUND - REENTER',                   *
               COLOR=RED,HILIGHT=BLINK
MSG2     DFHMDF POS=(04,04),LENGTH=22,ATTRB=(DRK,ASKIP),               *
               INITIAL='DATA ERROR - REENTER',                         *
               COLOR=RED,HILIGHT=BLINK
         DFHMDF POS=(05,04),LENGTH=09,ATTRB=PROT,                      *
               INITIAL='NUMBER  :'
CUSTNO   DFHMDF POS=(05,14),LENGTH=06,ATTRB=(IC,NUM)
         DFHMDF POS=(05,21),LENGTH=01
         DFHMDF POS=(06,04),LENGTH=09,ATTRB=PROT,COLOR=BLUE,           *
               INITIAL='PART NO :'
PARTNO   DFHMDF POS=(06,14),LENGTH=06,ATTRB=NUM
         DFHMDF POS=(06,21),LENGTH=01
         DFHMDF POS=(07,04),LENGTH=09,ATTRB=PROT,COLOR=BLUE,           *
               INITIAL='QUANTITY:'
QUANT    DFHMDF POS=(07,14),LENGTH=06,ATTRB=NUM
         DFHMDF POS=(07,21),LENGTH=01
         DFHMDF POS=(09,01),LENGTH=38,ATTRB=ASKIP,COLOR=BLUE,          *
               INITIAL='PRESS ENTER TO CONTINUE,CLEAR TO QUIT'
         DFHMSD TYPE=FINAL
         END
```

The symbolic storage definition produced as a result of the above statements would be as follows:

DSECT generated by DFH$CGK

```
01  ORDERI.
    02  FILLER PIC X(12).
    02  MSG1L    COMP  PIC  S9(4).
    02  MSG1F    PICTURE X.
    02  FILLER REDEFINES MSG1F.
      03 MSG1A    PICTURE X.
    02  MSG1I  PIC X(26).
    02  MSG2L    COMP  PIC  S9(4).
    02  MSG2F    PICTURE X.
    02  FILLER REDEFINES MSG2F.
      03 MSG2A    PICTURE X.
    02  MSG2I  PIC X(22).
    02  CUSTNOL    COMP  PIC  S9(4).
    02  CUSTNOF    PICTURE X.
    02  FILLER REDEFINES CUSTNOF.
      03 CUSTNOA    PICTURE X.
    02  CUSTNOI  PIC X(6).
    02  PARTNOL    COMP  PIC  S9(4).
    02  PARTNOF    PICTURE X.
    02  FILLER REDEFINES PARTNOF.
      03 PARTNOA    PICTURE X.
    02  PARTNOI  PIC X(6).
    02  QUANTL    COMP  PIC  S9(4).
    02  QUANTF    PICTURE X.
    02  FILLER REDEFINES QUANTF.
      03 QUANTA    PICTURE X.
    02  QUANTI  PIC X(6).
01  ORDERO REDEFINES ORDERI.
    02  FILLER PIC X(12).
    02  FILLER PICTURE X(3).
    02  MSG1O  PIC X(26).
    02  FILLER PICTURE X(3).
    02  MSG2O  PIC X(22).
    02  FILLER PICTURE X(3).
    02  CUSTNOO  PIC X(6).
    02  FILLER PICTURE X(3).
    02  PARTNOO  PIC X(6).
    02  FILLER PICTURE X(3).
    02  QUANTO  PIC X(6).
```

DFH$CGK screen layout

```
        +ORDER ENTRY

   +NUMBER NOT FOUND  REENTER
   +DATA ERROR  REENTER
   +NUMBER  :+XXXXXX+
   +PART NO :+XXXXXX+
   +QUANTITY:+XXXXXX+

 +PRESS ENTER TO CONTINUE, CLEAR TO QUIT
```

DFH$CGL map definition

```
         TITLE 'FILEA - MAP FOR ORDER ENTRY QUEUE PRINT - COBOL'
DFH$CGL  DFHMSD TYPE=&SYSPARM,MODE=OUT,                                *
               TIOAPFX=YES,LANG=COBOL
PRINT    DFHMDI SIZE=(05,80)
TITLE    DFHMDF POS=(01,01),LENGTH=43,                                 *
               INITIAL='NUMBER     NAME                    ADDRESS'
NUMB     DFHMDF POS=(02,01),LENGTH=06
NAM      DFHMDF POS=(02,12),LENGTH=20
ADDR     DFHMDF POS=(02,37),LENGTH=20
         DFHMDF POS=(03,01),LENGTH=09,                                 *
               INITIAL='PART NO :'
PART     DFHMDF POS=(03,11),LENGTH=06
         DFHMDF POS=(04,01),LENGTH=09,                                 *
               INITIAL='QUANTITY:'
QUANT    DFHMDF POS=(04,11),LENGTH=06
         DFHMDF POS=(05,01),LENGTH=1,                                  *
               INITIAL=' '
         DFHMSD TYPE=FINAL
         END
```

The symbolic storage definition produced as a result of the above statements would be as follows:

DSECT generated by DFH$CGL

```
       01  PRINTO.
           02  FILLER PIC X(12).
           02  FILLER PICTURE X(2).
           02  TITLEA    PICTURE X.
           02  TITLEO  PIC X(43).
           02  FILLER PICTURE X(2).
           02  NUMBA    PICTURE X.
           02  NUMBO  PIC X(6).
           02  FILLER PICTURE X(2).
           02  NAMA    PICTURE X.
           02  NAMO  PIC X(20).
           02  FILLER PICTURE X(2).
           02  ADDRA    PICTURE X.
           02  ADDRO  PIC X(20).
           02  FILLER PICTURE X(2).
           02  PARTA    PICTURE X.
           02  PARTO  PIC X(6).
           02  FILLER PICTURE X(2).
           02  QUANTA    PICTURE X.
           02  QUANTO  PIC X(6).
```

DFH$CGL print layout

```
+NUMBER     NAME                    ADDRESS
+XXXXXX    +XXXXXXXXXXXXXXXXXXXX    +XXXXXXXXXXXXXXXXXXXX
+PART NO :+XXXXXX
+QUANTITY:+XXXXXX
+X
```

Record descriptions for COBOL sample programs

FILEA record description

The FILEA record description is used by the sample programs. It is defined in copy code DFH$CFIL and has the following format:

```
02  FILEREC.
  03  STAT             PIC X.
  03  NUMB             PIC X(6).
  03  NAME             PIC X(20).
  03  ADDRX            PIC X(20).
  03  PHONE            PIC X(8).
  03  DATEX            PIC X(8).
  03  AMOUNT           PIC X(8).
  03  COMMENT          PIC X(9).
```

LOGA record description

The LOGA record description is used by the sample programs when an audit trail is written to a transient data file. It is defined in copy code DFH$CLOG and has the following format:

```
02  LOGHDR.
 03  LDAY             PIC S9(7) COMP-3.
 03  LTIME            PIC S9(7) COMP-3.
 03  LTERML           PIC X(4).
02  LOGREC.
 03  LSTAT            PIC X.
 03  LNUMB            PIC X(6).
 03  LNAME            PIC X(20).
 03  LADDR            PIC X(20).
 03  LPHONE           PIC X(8).
 03  LDATE            PIC X(8).
 03  LAMOUNT          PIC X(8).
 03  LCOMMENT         PIC X(9).
```

L86O record description

The L86O record description is used by the Order Entry Queue Print sample program when it writes to the transient data queue 'L86O'. It is defined in copy code DFH$CL86 and has the following format:

```
02  ITEM.
  03  CUSTNO           PIC X(6).
  03  PARTNO           PIC X(6).
  03  QUANTITY         PIC X(6).
  03  TERMID           PIC X(4).
```

Appendix F. Sample programs (PL/I)

The PL/I sample programs described in this appendix are included, in source form, on the CICS distribution tape. The *CICS/MVS Installation Guide* describes how these sample programs, and associated resources, can be defined to CICS and how the programs can be executed online.

This appendix describes six CICS sample application programs, written in PL/I, as follows:

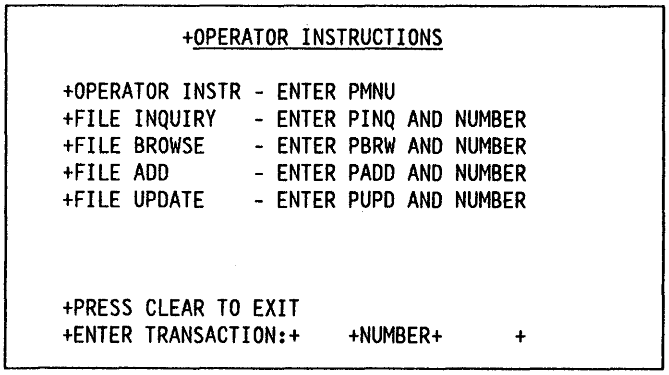
- Operator instruction
- Inquiry/update
- Browse
- Order entry
- Order entry queue print
- Low balance report.

These programs illustrate basic applications (such as inquire, browse, add, and update) that can serve as a framework for your installation's first programs. The programs operate using a VSAM file, known as FILEA, consisting of records containing details of individual customer accounts. Each program has a short description of what the program does, a listing of its source code, and a series of program notes. Numbered coding lines in the source listing correspond to the numbered program notes.

All the sample programs are for use with the IBM 3270 Information Display System.

The sample BMS maps include examples of how the COLOR, EXTATT, and HILIGHT attributes are specified in the map definition macros. However, due to production limitations, the associated screen layouts do not show you all the effects of these attributes.

You can add attributes without changing the application program by specifying EXTATT=MAPONLY in the DFHMSD map set definition macro. If you include an attribute that specifies a facility not available at the terminal, it will be ignored.



```
                +OPERATOR INSTRUCTIONS

    +OPERATOR INSTR - ENTER PMNU
    +FILE INQUIRY   - ENTER PINQ AND NUMBER
    +FILE BROWSE    - ENTER PBRW AND NUMBER
    +FILE ADD       - ENTER PADD AND NUMBER
    +FILE UPDATE    - ENTER PUPD AND NUMBER



    +PRESS CLEAR TO EXIT
    +ENTER TRANSACTION:+    +NUMBER+      +
```

The statements listed are those of the sample programs supplied with the initial release of CICS. Sample programs shipped with subsequent program temporary fixes (PTFs) may differ from these listings.

The BMS maps (which are unaligned) and the file record descriptions used by these sample programs are included at the end of the appendix.

Once CICS is running, type PMNU onto a clear screen and press the enter key. The PMNU transaction identifier invokes the 'Operator Instruction' sample program, which is a short program that produces a menu containing the transaction identifiers for two of the other sample programs, namely 'Inquiry/Update' and 'Browse'.

If you clear the screen, remember to reenter the transaction identifier, as no data is accepted from an unformatted screen.

You can run the sample programs using EDF but, because the CEDF transaction is defined with RSLC=YES, you must first sign on to CICS as an operator with an appropriate resource security level key.

The menu, on a screen that is 40 characters wide by 12 lines deep, is as shown in the box above. The plus (+) sign in this and subsequent displays shows the position of the attribute byte. In an actual display, this position contains a blank.

To invoke any of the transactions PMNU, PINQ, PBRW, PADD, or PUPD, do as instructed, entering the four-character transaction identifier and, when necessary, the six-digit account number in the fields highlighted in the bottom line of the display. These specific account numbers include the sequence 100000, 111111, 200000, 222222 ... , 999999.

These transaction identifiers give you access to the inquiry, add, and update functions of the 'Inquiry/Update' program, and access to the 'Browse' program.

You can invoke the three remaining sample programs 'Order Entry', 'Order Entry Queue Print', and 'Low Balance Report' separately by entering their transaction identifiers (PORD, PORQ, and PREP respectively) onto a clear screen.

Operator instruction program (PL/I)

Description

The operator instruction sample program displays map DFH$PGA in response to the EXEC CICS SEND MAP command.

The map displays a menu that lists the transaction identifiers associated with two of the sample programs, 'Inquiry/Update', and 'Browse', and gives instructions for the operator.

Source listing

```
/*********************************************************************/
/*      DFH$PMNU   CICS/VS SAMPLE FILEA OPERATOR INSTRUCTION MENU   */
/*********************************************************************/
   MENU: PROC OPTIONS(MAIN);
1    EXEC CICS SEND MAP('DFH$PGA') MAPONLY ERASE;
2    EXEC CICS RETURN;
     END;
```

Program notes

1. The BMS command erases the screen and displays map DFH$PGA.
2. The RETURN command ends the program.

Inquiry/update sample program (PL/I)

Description

The inquiry/update sample program lets you make an inquiry about, add to, or update records in a file. You can select one of these by entering the appropriate transaction identifier (PINQ, PADD, or PUPD) in the menu that is displayed when you start operations by entering PMNU.

To make an inquiry, enter PINQ and an account number into the menu. The program maps in the account number and reads the record from FILEA. The required fields from the file area, and a title 'FILE INQUIRY' are moved to the map dsect for DFH$PGB. DFH$PGB, containing the record fields, is displayed at your screen.

To add a record, enter PADD and the account number into the menu. The account number and a title 'FILE ADD' are moved to the map area of DFH$PGB. DFH$PGB, containing empty data fields, is displayed at your screen. The data fields entered are mapped into DFH$PGB and moved to the file record area which is then written to FILEA. The addition is recorded on an update log (LOGA), which is a transient data queue. The operator instruction screen is displayed with the message 'RECORD ADDED'.

To update a record, enter PUPD and the account number into the menu, as before. The program reads and displays the requested FILEA record. Modified data fields are mapped in to DFH$PGB and edited. The sample program only suggests the type of editing you might wish to do. Insert editing steps needed to ensure valid changes to the file. Those fields that have been changed are moved to the data record and the record is rewritten to FILEA. The update is recorded on LOGA. The message 'RECORD UPDATED' is moved to the dsect for DFH$PGA, the operator instruction menu map, which is then displayed at your screen.

This program is an example of pseudoconversational programming, in which control is returned to CICS together with a transaction identifier whenever a response is requested from the operator. Associated with each return of control to CICS is a storage area containing details associated with the previous invocation of this transaction.

Source listing for DFH$PALL

```
/********************************************************************/
/*      DFH$PALL - CICS/VS SAMPLE FILEA INQUIRY/UPDATE - PL/I       */
/********************************************************************/
UPDATE: PROC(COMPOINT)  OPTIONS(MAIN);
        DCL MESSAGES CHAR(39);
        DCL COMLEN FIXED BIN(15);
        DCL KEYNUM PICTURE '(6)9';
        %INCLUDE DFH$PGA;
        %INCLUDE DFH$PGB;
        %INCLUDE DFH$PFIL;
        %INCLUDE DFH$PLOG;
        %INCLUDE DFHBMSCA;
        DCL CHSTR CHAR(256) BASED;
        DCL COMPOINT PTR;
        DCL COMMAREA LIKE FILEA BASED(COMPOINT);
1       IF EIBCALEN¬=0 THEN GO TO READ_INPUT;
2       EXEC CICS HANDLE CONDITION ERROR(ERRORS) MAPFAIL(MFAIL);
        ALLOCATE COMMAREA;
3       EXEC CICS RECEIVE MAP('DFH$PGA');
4       IF KEYL=0 THEN GO TO BADLENG;
        IF VERIFY(KEYI,'0123456789')¬=0 THEN GOTO BADCHARS;
                                 /* KEYI CONTAINS 6 NUMERIC DIGITS */
        KEYNUM=KEYI;
        SUBSTR(ADDR(DFH$PGBO)->CHSTR,1,STG(DFH$PGBO))
               =LOW(STG(DFH$PGBO));
        SELECT(EIBTRNID);
          WHEN('PADD') DO;
5                        TITLEO='FILE ADD';
                         MSG3O='ENTER DATA AND PRESS ENTER KEY';
                         NUMBO,COMMAREA.NUMB=KEYI;
                         AMOUNTA=DFHBMUNN;
                         AMOUNTO='$0000.00';
                         COMLEN =7;
6                        CALL MAP_SEND;
                         GO TO CICS_CONTROL;
                       END;
          WHEN('PINQ',
               'PUPD') DO;
7                        EXEC CICS HANDLE CONDITION NOTFND(NOTFOUND);
8                        EXEC CICS READ DATASET('FILEA') INTO(FILEA)
                                                      RIDFLD(KEYNUM);
                         IF EIBTRNID='PINQ' THEN
                           DO;
9                            TITLEO='FILE INQUIRY';
                             MSG3O ='PRESS ENTER TO CONTINUE';
                             CALL MAP_BUILD;
                                 /* PROTECT ALL FIELDS ON MAP */
10                               NAMEA,ADDRA,PHONEA,DATEA,AMOUNTA,
                                                 COMMENTA = DFHBMPRO;
11                           CALL MAP_SEND;
                             EXEC CICS RETURN TRANSID('PMNU');
                           END;
                         ELSE DO;
12                           TITLEO='FILE UPDATE';
                             MSG3O ='CHANGE FIELDS AND PRESS ENTER';
13                           COMMAREA.FILEREC=FILEA.FILEREC;
```

Source listing for DFH$PALL (continued)

```
14                            CALL MAP_BUILD;
                              CALL MAP_SEND;
15                            COMLEN=80;
                              GO TO CICS_CONTROL;
                            END;
                        END;
16         OTHERWISE GO TO ERRORS;
         END;
 MAP_BUILD:   PROC;
17       NUMBO    = FILEA.NUMB;
         NAMEO    = FILEA.NAME;
         ADDRO    = FILEA.ADDRX;
         PHONEO   = FILEA.PHONE;
         DATEO    = FILEA.DATEX;
         AMOUNTO  = FILEA.AMOUNT;
         COMMENTO = FILEA.COMMENT;
         RETURN;
 END;
 MAP_SEND:    PROC;
18       EXEC CICS SEND MAP('DFH$PGB') ERASE;
         RETURN;
 END;
 READ_INPUT:
19
20       EXEC CICS HANDLE CONDITION MAPFAIL(NOTMODF) DUPREC(DUPREC)
                                    ERROR(ERRORS) NOTFND(NOTFOUND);
21       EXEC CICS RECEIVE MAP('DFH$PGB');
         SELECT(EIBTRNID);
           WHEN('PUPD')
            DO;
22            EXEC CICS READ UPDATE DATASET('FILEA') INTO(FILEA)
                                                RIDFLD(COMMAREA.NUMB);
              IF STRING(FILEA.FILEREC)¬=STRING(COMMAREA.FILEREC) THEN
                 DO;
23                 MSG1O='RECORD UPDATED BY OTHER USER, TRY AGAIN';
                   MSG1A=DFHBMASB;
                   MSG3A=DFHPROTN;
                   CALL MAP_BUILD;
                   EXEC CICS SEND MAP('DFH$PGB') DATAONLY;
                   COMMAREA.FILEREC=FILEA.FILEREC;
                   COMLEN=80;
                   GO TO CICS_CONTROL;
                 END;
              ELSE
                 DO;
24                 FILEA.STAT='U';
                   MESSAGES='RECORD UPDATED';
                 END;
            END;
           WHEN('PADD') DO;
25                        FILEA.STAT='A';
                          MESSAGES='RECORD ADDED';
                        END;
26         OTHERWISE GO TO ERRORS;
         END;
27       IF NAMEL    = 0 &
            ADDRL    = 0 &
            PHONEL   = 0 &
            DATEL    = 0 &
```

Source listing for DFH$PALL (continued)

```
            AMOUNTL  = 0 &
            COMMENTL = 0 THEN
            GO TO NOTMODF;
28       SELECT(EIBTRNID);
           WHEN('PADD') IF
               VERIFY(NAMEI,'ABCDEFGHIJKLMNOPQRSTUVWXYZ .-''')¬=0 THEN
               GO TO DATA_ERROR;
           WHEN('PUPD')
                        DO;
            IF NAMEL¬=0 THEN
            IF VERIFY(NAMEI,'ABCDEFGHIJKLMNOPQRSTUVWXYZ .-''')¬=0
                                                THEN
               GO TO DATA_ERROR;
            IF AMOUNTL¬=0 THEN
            IF VERIFY(AMOUNTI,'0123456789.$¢')¬=0 THEN
               GO TO DATA_ERROR;
                        END;
           OTHERWISE;
         END;
29       IF EIBTRNID='PADD' THEN
            FILEA.NUMB=COMMAREA.NUMB;
         IF NAMEL   ¬= 0 THEN FILEA.NAME=NAMEI;
         IF ADDRL   ¬= 0 THEN FILEA.ADDRX=ADDRI;
         IF PHONEL  ¬= 0 THEN FILEA.PHONE=PHONEI;
         IF DATEL   ¬= 0 THEN FILEA.DATEX=DATEI;
         IF AMOUNTL ¬= 0 THEN FILEA.AMOUNT=AMOUNTI;
         ELSE IF EIBTRNID = 'PADD' THEN FILEA.AMOUNT='$0000.00';
         IF COMMENTL¬=0 THEN FILEA.COMMENT=COMMENTI;
30       LOGREC=FILEA.FILEREC;
         LDAY  =EIBDATE;
         LTIME=EIBTIME;
         LTERML=EIBTRMID;
31       EXEC CICS WRITEQ TD QUEUE('LOGA') FROM(LOGA) LENGTH(92);
         IF EIBTRNID='PUPD' THEN
32          EXEC CICS REWRITE DATASET('FILEA') FROM(FILEA);
         ELSE
33          EXEC CICS WRITE DATASET('FILEA') FROM(FILEA)
                                             RIDFLD(COMMAREA.NUMB);
         GO TO PMNU;
 DATA_ERROR:
34       MSG3A=DFHBMASB;
         MSG3O='DATA ERROR - CORRECT AND PRESS ENTER';
   /* PRESERVE CONTENTS OF SCREEN BY SETTING MODIFIED DATA TAG*/
   /* AMOUNT IS MADE NUMERIC AND MODIFIED*/
         AMOUNTA=DFHUNNUM;
35       NAMEA, ADDRA, PHONEA, DATEA, COMMENTA=DFHBMFSE;
36       EXEC CICS SEND MAP('DFH$PGB') DATAONLY;
37       IF EIBTRNID='PADD' THEN COMLEN=7;
         ELSE COMLEN=80;
 CICS_CONTROL:
38       EXEC CICS RETURN TRANSID(EIBTRNID) COMMAREA(COMMAREA)
                                            LENGTH(COMLEN);
 NOTMODF:
```

Source listing for DFH$PALL (continued)

```
39       MESSAGES='RECORD NOT MODIFIED';
         GO TO PMNU;
 DUPREC:
         MESSAGES='DUPLICATE RECORD';
         GO TO PMNU;
 BADLENG:
         MESSAGES='PLEASE ENTER AN ACCOUNT NUMBER';
         GOTO PMNU;
 BADCHARS:
         MESSAGES='ACCOUNT NUMBER MUST BE NUMERIC';
         GO TO PMNU;
 NOTFOUND:
         MESSAGES='INVALID NUMBER - PLEASE REENTER';
         GO TO PMNU;
 MFAIL:
         MESSAGES='PRESS CLEAR TO EXIT';
         GOTO PMNU;
 ERRORS:
40       EXEC CICS DUMP DUMPCODE('ERRS');
         MESSAGES='TRANSACTION TERMINATED';
 PMNU:
41       SUBSTR(ADDR(DFH$PGAO)->CHSTR,1,STG(DFH$PGAO))
                  =LOW(STG(DFH$PGAO));
         MSGA=DFHBMASB;
         MSGO=MESSAGES;
42       EXEC CICS SEND MAP('DFH$PGA') ERASE;
43       EXEC CICS RETURN;
 END;
```

Program notes

1. The length of the COMMAREA is tested. If not zero then this is the validation stage of an add or update.
2. The program exits are set up.
3. The menu map DFH$PGA is received. The account number, if entered, is mapped into KEYI in the dsect for DFH$PGA.
4. The account number is validated and saved.
5. If the program is invoked by PADD, a title and command message are moved to the map area. The record key is moved to the map area and saved in COMMAREA. The amount field has the attribute byte set to numeric.
6. The add screen is displayed and the program terminates to await a reply from the terminal.
7. For an inquiry or update the exit for the record-not-found condition is set up.
8. The file control READ command reads the file record into the file area.
9. If the program is invoked by PINQ, a title and command message are moved to the map area. The file record fields are moved to the map area by a subroutine.
10. All field attributes are set to protected.
11. The inquiry screen is displayed and the program terminates. The TRANSID of PMNU causes the operator instruction program to be invoked when the next response is received from the terminal.
12. If the program is invoked by PUPD, a title and command message are moved to the map area.
13. The file record is saved in COMMAREA.
14. Data is moved to the map dsect and displayed.
15. The length of the COMMAREA to be returned is set up and control is returned to CICS.
16. An unknown transaction identifier is treated as an error.
17. This subroutine moves fields from the FILEA record to the map dsect for DFH$PGB ready for display.
18. MAP_SEND sends the map DFH$PGB to the screen specifying that the screen is to be erased before the map is displayed.
19. Control is passed here when the test of EIBCALEN, at the beginning of the program, finds that a COMMAREA has been received. This part of the program maps in data for an add or update request, performs validation and updates FILEA.
20. The error exits are set up.
21. The RECEIVE MAP command maps in the variables from the screen.

22. If this is an update request a file control READ UPDATE command reads the existing record using the number stored in COMMAREA by the last invocation of this program.

23. If the current file record is not the same as the one saved in COMMAREA then another user has updated the record. A warning message is displayed, with fields from the record read from FILEA, for reentry of the updates.

24. The update flag is set in the record area and the message 'RECORD UPDATED' is moved to the message area ready for display on the operator instruction screen.

25. If this is an add request the add flag is set in the new record and the message 'RECORD ADDED' is moved to the message area ready for display on the operator instruction screen.

26. An unknown transaction identifier is treated as an error.

27. If all length fields in the input map are zero then no data has been entered on the screen.

28. Any required editing steps should be inserted here. A suitable form of editing should be used to ensure valid records are placed on the file.

29. This code creates or updates the account record. Any field which has been entered is moved to the account record.

30. The record fields, the date, the time, and the termid are moved to the update log record area.

31. The record is written to the update log which is a transient data queue.

32. For an update request the updated account record is rewritten to FILEA.

33. For an add request the new account record is written to the file.

34. When a data error is detected the screen is redisplayed for errors to be corrected. An error message is moved to the map area and highlighted.

35. The modified data tag is set on for all the data fields so that all data is received at the next RECEIVE MAP.

36. The contents of map DFH$PGB are sent to the screen. The constant information on the screen is not refreshed as a result of the use of the DATAONLY option.

37. The size of the COMMAREA is set to 7 for an add request or to 80 for an update request.

38. After the FILE ADD or FILE UPDATE screen has been displayed the program branches here to return to CICS awaiting a response from the terminal. The RETURN gives CICS the transaction identifier for the next transaction at this terminal together with a COMMAREA containing all information that the program needs to continue the update. The COMMAREA is passed to the next invocation of this program, see note 1 on page 439.

39. These short error routines set up an error message in MESSAGES and branch to PMNU to display the message in the operator instruction menu DFH$PGA.

40. If a CICS command fails with the ERROR condition or if an unknown transaction identifier is used to invoke this program, a dump is taken and the message 'TRANSACTION TERMINATED' is moved to MESSAGES for display on the operator instruction screen.

41. This code gets control when an add or update is complete. An information or error message is in MESSAGES. The operator instruction map area is cleared. The message is moved to the map area and highlighted.

42. The operator instruction map DFH$PGA is displayed on an erased screen.

43. The program terminates by returning to CICS. No transaction identifier or COMMAREA is specified.

Browse sample program (PL/I)

Description

The browse program sequentially retrieves a page or set of records for display, starting at a point in a file specified by the terminal operator.

To start a browse, type PBRW and an account number into the menu and press the enter key. If you omit the account number browsing begins at the start of the file.

Depressing the PF1 key or typing F causes retrieval of the next page or paging forward. If you wish to re-examine the previous records displayed, press PF2 or type B. This lets you page backward.

The browse program uses READNEXT to forward page to the end of the file and READPREV to backward page to the start of the file.

Source listing for DFH$PBRW

```
/*********************************************************************/
/*      DFH$PBRW - CICS/VS SAMPLE FILEA BROWSE - PL/I               */
/*********************************************************************/
BROWSE: PROC OPTIONS(MAIN);
  %INCLUDE DFHBMSCA;                    /*STANDARD ATTRIBUTE CHARACTERS*/
  %INCLUDE DFH$PFIL;                    /*FILEA RECORD DESCRIPTION     */
  %INCLUDE DFH$PGA;                     /*'GENERAL MENU'=MAP'A'        */
  %INCLUDE DFH$PGC;                     /*'BROWSE FILEA'=MAP'B'        */

  DCL                                   /*BUILT IN FUNCTIONS           */
    (ADDR,
     HIGH,
     LOW,
     STG,
     SUBSTR,
     VERIFY) BUILTIN;

  DCL I         FIXED BIN(15);

  DCL (RID,                                    /*USED AS RIDFLD PARAM  */
       RIDB,                                   /*FOR BUILDING PREV PAGE*/
       RIDF)                                   /*FOR BUILDING NEXT PAGE*/
                PIC'999999'  INIT(0);

  DCL (CURROP,                             /* NOTE CURRENT OPERATION   */
       LASTOP,                             /*      LAST OPERATION      */
                                           /*  F = GOING FORWARDS,     */
                                           /*  B = GOING BACKWARDS.    */
       STATUS) CHAR(1) INIT('');           /*STATUS H = AT TOP OF FILE */
                                           /*        L = AT BOTTOM.    */

  DCL MESSAGES  CHAR(39)     INIT('');
  DCL STRING    CHAR(256)    BASED;

1 EXEC CICS HANDLE AID CLEAR(SMSG)
                       PF1(PAGE_FORWARD)
                       PF2(PAGE_BACKWARD);

2 EXEC CICS HANDLE CONDITION ERROR(ERRORS)
                             MAPFAIL(SMSG)
                             NOTFND(NOTFOUND);

3 EXEC CICS RECEIVE MAP('DFH$PGA');            /*READ FIRST A/C NO.
*/
/*********************************************************************/
/*                                    SIMPLE CHECKS OF INPUT DATA    */
/*********************************************************************/
  SELECT(KEYL);
    WHEN(0)  DO;                                   /*DEFAULT=000000  */
4              RID =000000;
               RIDF=000000;
             END;
    OTHERWISE
             DO;
               IF VERIFY(KEYI,'0123456789')=0 THEN
                 DO;
5                  RID =KEYI;                      /* NUMERIC A/C NO.*/
                   RIDF=KEYI;
                   RIDB=KEYI;
```

Source listing for DFH$PBRW (continued)

```
                  END;
                ELSE
                  DO;
                    MESSAGES='ACCOUNT NUMBER MUST BE NUMERIC';
                    GOTO PMNU;
                  END;
              END;
   END;

 6 EXEC CICS STARTBR DATASET('FILEA') RIDFLD(RID); /*ESTABLISH 'START'*/
   IF RID¬=999999 THEN
                       GOTO PAGE_FORWARD;
 7    STATUS='H';
      GOTO PAGE_BACKWARD;
 /*********************************************************************/
 /*                                    HANDLE PAGING REQUESTS         */
 /*********************************************************************/
 PAGE_FORWARD:
   CURROP='F';
 8 EXEC CICS HANDLE CONDITION ENDFILE(TOOHIGH);

   SUBSTR(ADDR(DFH$PGCO)->STRING,1,STG(DFH$PGCO)) = LOW(STG(DFH$PGCO));
                                       /*RESET FIELDS + ATTRB IN MAP C*/

                                       /*IF LAST REQUEST=BACKPAGE THEN*/
   RID =RIDF;                          /*NEED RIDF FOR FORWARD PAGING */
 9 CALL BUILDNEXT;
   RIDF=RID;                           /*USE RIDF FOR NEXT PAGE       */
10 EXEC CICS SEND MAP('DFH$PGC') ERASE ;
   GOTO RECEIVE;

 PAGE_BACKWARD:
11 CURROP='B';
   EXEC CICS HANDLE CONDITION ENDFILE(TOOLOW);

   SUBSTR(ADDR(DFH$PGCO)->STRING,1,STG(DFH$PGCO)) = LOW(STG(DFH$PGCO));
                                       /*RESET FIELDS + ATTRB IN MAP C*/

   RID =RIDB;                          /*USE RIDB FOR BACKWARD PAGING */
   RIDF=RIDB;                          /*SAVE RIDF FOR FORWARD PAGING */
   IF LASTOP='B' THEN
                      GOTO PREVLINE;
   IF STATUS='H' THEN
                      GOTO PREVLINE;

 PREVXTRA:
   EXEC CICS READPREV INTO(FILEA) DATASET('FILEA') RIDFLD(RID);

 PREVLINE:
   CALL BUILDPREV;
   RIDB=RID;                           /*SAVE RIDB FOR PREVIOUS PAGE  */
   EXEC CICS SEND MAP('DFH$PGC') ERASE ;

 RECEIVE:
   LASTOP=CURROP;
12 EXEC CICS RECEIVE MAP('DFH$PGC');
   SELECT(DIRI);
     WHEN('F') GOTO PAGE_FORWARD;
```

Source listing for DFH$PBRW (continued)

```
     WHEN('B') GOTO PAGE_BACKWARD;
     OTHERWISE
       DO;
         EXEC CICS SEND MAP('DFH$PGC');
         GOTO RECEIVE;
       END;
   END;
 /*********************************************************************/
 /*                                    HANDLE END OF FILE CONDITIONS  */
 /*********************************************************************/
 TOOHIGH:
13 STATUS='H';
   RIDF  = RID;
   RIDB  = RID;
   DIRO  = ' ';
   MSG1O='HI-END OF FILE';
   MSG1A=DFHBMASB;
   EXEC CICS SEND MAP('DFH$PGC') ERASE;
   GOTO RECEIVE;
 TOOLOW:
14 STATUS='L';
   RIDF  = 000000;
   RIDB  = 000000;
   DIRO  = ' ';
   MSG2O='LO-END OF FILE';
   MSG2A=DFHBMASB;
   EXEC CICS SEND MAP('DFH$PGC') ERASE;
   GOTO RECEIVE;

 /*********************************************************************/
 /*                                    HANDLE GENERAL CONDITIONS      */
 /*********************************************************************/

 NOTFOUND:
15 MESSAGES='END OF FILE - PLEASE RESTART';
   EXEC CICS ENDBR DATASET('FILEA');
   GOTO PMNU;

 SMSG:
16 MESSAGES='PRESS CLEAR TO EXIT';
   GOTO PMNU;

 ERRORS:
17 EXEC CICS DUMP DUMPCODE('ERRS');
   MESSAGES='TRANSACTION TERMINATED';
 /*********************************************************************/
 /*                                     DISPLAY GENERAL MENU THEN EXIT*/
 /*********************************************************************/

 PMNU:
18 SUBSTR(ADDR(DFH$PGAO)->STRING,1,STG(DFH$PGAO)) = LOW(STG(DFH$PGAO));
   MSGA=DFHBMASB;
   MSGO=MESSAGES;
   EXEC CICS SEND MAP('DFH$PGA') ERASE;
19 EXEC CICS RETURN;

 /*********************************************************************/
 /* INTERNAL PROCEDURES                MOVE REQUIRED FIELDS TO MAP    */
 /*********************************************************************/
```

Source listing for DFH$PBRW (continued)

```
 BUILDNEXT: PROC;
20
   DO I=1 TO 4;
21   EXEC CICS READNEXT INTO(FILEA) DATASET('FILEA') RIDFLD(RID);
     SELECT(I);
       WHEN(1) DO;
22               NUMBER10 = NUMB;
                 NAME10   = NAME;
                 AMOUNT10 = AMOUNT;
                 RIDB     = RID;      /*RIDB NEEDS AN EXISTING A/C NO.*/
               END;
       WHEN(2) DO;
23               NUMBER20 = NUMB;
                 NAME20   = NAME;
                 AMOUNT20 = AMOUNT;
               END;
       WHEN(3) DO;
                 NUMBER30 = NUMB;
                 NAME30   = NAME;
                 AMOUNT30 = AMOUNT;
               END;
       WHEN(4) DO;
                 NUMBER40 = NUMB;
                 NAME40   = NAME;
                 AMOUNT40 = AMOUNT;
               END;
     END;
   END;
 END BUILDNEXT;
 BUILDPREV: PROC;
   DO I=1 TO 4;
24   EXEC CICS READPREV INTO(FILEA) DATASET('FILEA') RIDFLD(RID);
     SELECT(I);
       WHEN(4) DO;                     /*PUT FIELDS IN ASCENDING ORDER*/
                 NUMBER10 = NUMB;
                 NAME10   = NAME;
                 AMOUNT10 = AMOUNT;
               END;
       WHEN(3) DO;
                 NUMBER20 = NUMB;
                 NAME20   = NAME;
                 AMOUNT20 = AMOUNT;
               END;
       WHEN(2) DO;
                 NUMBER30 = NUMB;
                 NAME30   = NAME;
                 AMOUNT30 = AMOUNT;
               END;
       WHEN(1) DO;
                 NUMBER40 = NUMB;
                 NAME40   = NAME;
                 AMOUNT40 = AMOUNT;
               END;
     END;
   END;
 END BUILDPREV;

 END BROWSE;
```

Program notes

1. The exits for CLEAR, PF1 and PF2 are set up.
2. The error exits are set up.
3. This command maps in the account number from the operator instruction screen.
4. If no account number is entered browsing begins at the start of the file.
5. If the format of the account number is valid the number is used to set the program's browse pointers, otherwise an error message is displayed on the operator instruction menu.
6. The STARTBR command establishes the browse starting point.
7. Entering the maximum value (999999) for the account number begins a backward browse from the end of the file.
8. The forward browse end of file exit is set up.
9. A subroutine builds a page for display.
10. The screen is erased and the full page is displayed at the terminal.
11. The backward browse procedure is similar to the forward browse. Note the need for an extra READPREV when changing from forward to backward browsing.
12. When the RECEIVE command executes control will go to one of the HANDLE AID exits (see note 1) if CLEAR, PF1 or PF2 is pressed. The program explicitly tests for F or B if no exit is taken. Any other terminal response is ignored.
13. If the end of file is reached, on a READNEXT, any records read to that point are displayed together with a highlighted message 'HI-END OF FILE'.
14. If the start of file is reached on a READPREV (backward browse) then the ENDFILE condition occurs and TOOLOW gets control. Any records read up to that point are displayed, together with a highlighted message 'LO-END OF FILE'.
15. If the NOTFND condition occurs at the start browse (note 6) the message 'END OF FILE - PLEASE RESTART' is moved to MESSAGES for display on the operator instruction screen.
16. If the CLEAR key is pressed or when a MAPFAIL occurs a message 'PRESS CLEAR TO EXIT' is moved to MESSAGES for display on the operator instruction screen.
17. In some error situations a dump is taken and the message 'TRANSACTION TERMINATED' is moved to MESSAGES for display on the operator instruction screen.
18. This code displays the operator instruction menu with a message which has been stored in MESSAGES.
19. The program terminates by returning to CICS.
20. BUILDNEXT browses forward through FILEA building a screen, or page, of accounts for display.
21. The READNEXT reads the first record, and subsequently the next record, into the file area.
22. The account number, name, and amount are moved to the first line of the browse map area.
23. The same basic commands are repeated to read and set up the next three lines. The same file area is used for each read.
24. Backward browsing uses the READPREV command to read the previous record and stores records in the map area starting at the bottom line.

Order entry sample program (PL/I)

Description

The order entry sample application program provides a data entry facility for customer orders for parts from a warehouse. Orders are recorded on a transient data queue which 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 PORD onto a blank screen and press ENTER. The order entry program displays the map DFH$PGK on the screen requesting the operator to enter order details, that is, customer number, part number, and the quantity of that part required. The customer number must be valid, that is, it must exist on FILEA. 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. A definition of the transient data queue is included in the sample destination control table listed in the *CICS/MVS Installation Guide*. The TRANSID specified in the DCT entry for L86O must be changed from AORQ to PORQ for the PL/I program to be triggered.

The trigger level may be changed using the CEMT command, as follows:

```
CEMT SET QUEUE(L860) TRIGGER(n)
```

where n is the destination trigger level (any integer from 0 through 32767).

Source listing for DFH$PREN

```
/*******************************************************************/
/*      DFH$PREN - CICS/VS SAMPLE FILEA ORDER ENTRY - PL/I         */
/*******************************************************************/
ORDER: PROC OPTIONS(MAIN);

%INCLUDE DFHBMSCA;                   /*STANDARD ATTRIBUTE CHARACTERS*/
%INCLUDE DFH$PFIL;                   /*COLLECTION OF ACCOUNTS       */
%INCLUDE DFH$PL86;                   /*RECORD DESCRIPTION FOR L860  */
%INCLUDE DFH$PGK;                    /*MAP DEFINITION               */

DCL                                  /*BUILT IN FUNCTIONS           */
   (ADDR,
    LOW,
    STG,
    SUBSTR,
    VERIFY) BUILTIN;

DCL
    CHSTR       CHAR(256) BASED;
DCL
    ERROR_FLAG  BIT(1)    INIT('0'B);
DCL PRESMSG CHAR(20) STATIC
      INIT('PROCESSING COMPLETED');

1 EXEC CICS HANDLE AID CLEAR(ENDPORD);         /*EXIT FOR 'CLEAR'   */
2 EXEC CICS HANDLE CONDITION MAPFAIL(MAPFAIL)  /*EXITS FOR ERRORS   */
                             ERROR  (ERRORS)
                             NOTFND (NOTFOUND);

/********************************************************************/
/*                                       ZEROIZE PLI STRUCTURE=DFH$PGK*/
/********************************************************************/

  SUBSTR(ADDR(DFH$PGKO)->CHSTR,1,STG(DFH$PGKO))=LOW(STG(DFH$PGKO));

3 EXEC CICS SEND MAP('DFH$PGK') ERASE;  /*ERASE SCREEN + DISPLAY MAP*/

  RECEIVE:
4 EXEC CICS RECEIVE MAP('DFH$PGK');  /*MAP IN CUSTNO,PARTNO & QUANT */

  ERROR_FLAG='0'B;
  CUSTNOA,PARTNOA,QUANTA=DFHBMFSE;   /*MDT=1 IN CASE NEED TO REINPUT*/

/********************************************************************/
/*                                       SIMPLE VALIDATION OF DATA   */
/********************************************************************/

5 IF VERIFY(CUSTNOI,'1234567890')¬=0 THEN
    DO;
      CUSTNOA    = DFHUNINT;
      ERROR_FLAG = '1'B;
    END;

  IF VERIFY(PARTNOI,'1234567890')¬=0 THEN
    DO;
      PARTNOA    = DFHUNINT;
      ERROR_FLAG = '1'B;
    END;
```

Source listing for DFH$PREN (continued)

```
   IF VERIFY(QUANTI,'1234567890')¬=0 THEN
     DO;
       QUANTA     = DFHUNINT;
       ERROR_FLAG = '1'B;
     END;

   IF ERROR_FLAG THEN
     DO;
 6     MSG2A=DFHBMASB;                             /*DATA ERROR-REENTER */
       EXEC CICS SEND MAP('DFH$PGK') ERASE;
       GOTO RECEIVE;
     END;

 /*********************************************************************/
 /*                                    READ RECORD,CHECK CUSTNO EXISTS*/
 /*********************************************************************/

 7 EXEC CICS READ DATASET('FILEA') INTO(FILEA) RIDFLD(CUSTNOI);
 8 CUSTNO   = CUSTNOI;
   PARTNO   = PARTNOI;
   QUANTITY = QUANTI;
   TERMID   = EIBTRMID;

 /*********************************************************************/
 /*                                    WRITE VALID ORDER TO TD QUEUE  */
 /*********************************************************************/

 9 EXEC CICS WRITEQ TD QUEUE ('L860') FROM (L860) LENGTH(22);

10 EXEC CICS SEND MAP('DFH$PGK') MAPONLY ERASEAUP;
   GOTO RECEIVE;

 /*********************************************************************/
 /*                                    HANDLE ERRORS THEN RESTART     */
 /*********************************************************************/

11 NOTFOUND:                                       /*INVALID ACCOUNT NO */
   CUSTNOA = DFHUNINT;
   MSG1A   = DFHBMASB;                             /*NUMBER NOT FOUND   */
   EXEC CICS SEND MAP('DFH$PGK');
   GOTO RECEIVE;

12 MAPFAIL:                                        /*NO DATA IN FIELDS  */

   SUBSTR(ADDR(DFH$PGKO)->CHSTR,1,STG(DFH$PGKO))=LOW(STG(DFH$PGKO));

   MSG2A=DFHBMASB;                                 /*DATA ERROR -REENTER*/
   EXEC CICS SEND MAP('DFH$PGK');
   GOTO RECEIVE;

 /*********************************************************************/
 /*                                    EXIT FROM PROGRAM              */
 /*********************************************************************/

   ERRORS:                                         /*GENERAL ERROR COND */
13 MSG2O='TRANSACTION TERMINATED';
   MSG2A=DFHBMASB;                                 /*DATA ERROR -REENTER*/
```

Source listing for DFH$PREN (continued)

```
   EXEC CICS SEND MAP('DFH$PGK');
   EXEC CICS DUMP DUMPCODE('ERRS');
   GOTO EXIT;

   ENDPORD:                                  /*EXIT-'CLEAR'WAS HIT*/
14 EXEC CICS SEND TEXT FROM(PRESMSG) ERASE;
   EXEC CICS SEND CONTROL FREEKB;            /*SET INPUT-INHIB OFF*/
   EXIT:
   EXEC CICS RETURN;
   END;
```

Program notes

1. The CLEAR key exit is set up.
2. The error exits are set up.
3. The screen is erased and the order entry map is displayed at the terminal.
4. This RECEIVE MAP causes a read from the terminal and maps in the customer number, part number, and quantity. The program remains in virtual storage until the terminal response is received. Compare this technique with that used in the pseudoconversational inquiry/update sample program. If no data is received CICS branches to the MAPFAIL exit (note 2).
5. The order details are checked, invalid orders are redisplayed for correction. Error fields are highlighted and have MDT set on. The user should add further editing steps necessary to ensure only valid orders are accepted.
6. The error message 'DATA ERROR – REENTER' is a constant in the map load module and is sent to the terminal, with any other constant information, unless DATAONLY is specified on the SEND MAP. The message is normally dark (non-display). This instruction overrides the dark attribute and the message appears in high intensity when the SEND MAP command is executed.
7. The file control READ command attempts to read the customer record from FILEA. If no record exists for the customer CICS branches to the NOTFND exit (note 2).
8. The order details are moved from the input map to the queue area.
9. The WRITEQ TD command writes the order record to a sequential file, a transient data queue.
10. The order entry map is redisplayed ready for the next order. Only the map load module is used to build the screen display, MAPONLY causes the data in the map dsect area to be ignored. ERASEAUP erases all the unprotected data on the screen, that is, the customer number, part number, and quantity.
11. If there is no record for the customer on FILEA, CICS raises the NOTFND condition and branches here. The attribute for the customer number field is set to high intensity with MDT on and an error message 'NUMBER NOT FOUND - REENTER' is set to display in high intensity (see note 6). The order is redisplayed for correction.
12. If no fields are entered, the MAPFAIL condition occurs. The message 'DATA ERROR-REENTER' is displayed in high intensity (see note 6).
13. If an error occurs a dump is taken, and the message 'TRANSACTION TERMINATED' is displayed in high intensity in the data error message area. The program terminates leaving the order entry screen displayed.
14. When the CLEAR key is pressed the program terminates. The message 'PROCESSING COMPLETED' is displayed on a blank screen, the keyboard is freed and control is returned to CICS.

Order entry queue print sample program (PL/I)

Description

The order entry queue print sample program sends customer orders to a printer terminal at the warehouse. The order entry sample program, described earlier, records customer orders on a transient data queue which is read by this program.

The queue print transaction can be invoked in one of three ways:

- You can type the transaction identifier PORQ onto 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 COMPLETED' is displayed and your terminal is available for other work.
- One hour after you enter PORQ, the queue print transaction is automatically invoked by CICS interval control. In this case the terminal identifier, specified by the START, is L86O so the program prints the orders at the warehouse.
- The queue print transaction is "triggered" 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 will print the orders. The TRANSID specified in the DCT entry for L86O must be changed from AORQ to PORQ for the PL/I program to be triggered. The trigger level may be changed using the command:

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

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/update sample program. When all the orders have been processed, or if there were no orders to process, the message 'ORDER QUEUE IS EMPTY' is printed at the warehouse.

Source listing for DFH$PCOM

```
/*****************************************************************/
/*      DFH$PCOM - CICS/VS SAMPLE FILEA ORDER ENTRY QUEUE PRINT     */
/*****************************************************************/
QPRINT: PROC OPTIONS(MAIN);
        %INCLUDE DFH$PFIL;
        %INCLUDE DFH$PL86;
        %INCLUDE DFH$PGL;
        DCL Q_LENGTH FIXED BIN(15) INIT(22);
        DCL 1 LOGORD,
              2 LOGTIME,
                3 LDATE FIXED DEC(7,0),
                3 LTIME FIXED DEC(7,0),
              2 LITEM CHAR(22),
              2 COMMENT CHAR(11) INIT('ORDER ENTRY'),
              2 FILLER CHAR(51) INIT(' ');
        DCL CHSTR CHAR(256) BASED;
  DCL PRESMSG CHAR(20) STATIC
      INIT('PROCESSING COMPLETED');
1       EXEC CICS HANDLE CONDITION ERROR(ERRORS) QZERO(ENDA);
2       IF EIBTRMID¬='L860' THEN
           GO TO TIME;
        SUBSTR(ADDR(DFH$PGLO)->CHSTR,1,STG(DFH$PGLO))
                 =LOW(STG(DFH$PGLO));
Q_READ:
3       EXEC CICS READQ TD INTO(L860) LENGTH(Q_LENGTH) QUEUE('L860');
MAP_BUILD:
4       EXEC CICS READ DATASET('FILEA') INTO(FILEA) RIDFLD(CUSTNO);
5       IF AMOUNT>'$0100.00' THEN DO;
6          ADDRO  = ADDRX;
           PARTO  = PARTNO;
           NAMO   = NAME;
           NUMBO  = CUSTNO;
           QUANTO = QUANTITY;
7          EXEC CICS SEND MAP('DFH$PGL') ERASE PRINT L80;
        END;
        ELSE DO;
8          LDATE = EIBDATE;
           LTIME = EIBTIME;
           LITEM = STRING(ITEM);
9          EXEC CICS WRITEQ TD QUEUE('LOGA') FROM(LOGORD) LENGTH(92);
        END;
        GO TO Q_READ;
ERRORS:
10      EXEC CICS DUMP DUMPCODE('ERRS');
        GO TO FIN;
ENDA:
        SUBSTR(ADDR(DFH$PGLO)->CHSTR,1,STG(DFH$PGLO))
                 =LOW(STG(DFH$PGLO));
11      TITLEO='ORDER QUEUE IS EMPTY';
        EXEC CICS SEND MAP('DFH$PGL') DATAONLY ERASE PRINT L80;
TIME:
                                 /* IF THE COMMENT DELIMITERS ARE   */
                                 /* REMOVED FROM THE NEXT TWO PL/I  */
                                 /* STATEMENTS, THE APPLICATION WILL*/
                                 /* BE RESTARTED IN AN HOUR IF THE  */
                                 /* TIME OF DAY RIGHT NOW IS NOT    */
                                 /* LATER THAN 1400 HRS. IF THE     */
                                 /* CODE IS LEFT UNCHANGED THE      */
                                 /* APPLICATION WILL BE RESTARTED   */
```

Source listing for DFH$PCOM (continued)

```
                                  /* UNCONDITIONALLY AFTER AN HOUR   */
                                  /* HAS ELAPSED                     */
     /*  EXEC CICS ASKTIME;       */
     /*  IF EIBTIME¬>140000 THEN  */
 12         EXEC CICS START TRANSID('PORQ')  INTERVAL(10000)
                                             TERMID('L860');
 FIN:
 13 EXEC CICS SEND TEXT FROM(PRESMSG) ERASE;
   EXEC CICS SEND CONTROL FREEKB;              /*SET INPUT-INHIB OFF*/

         EXEC CICS RETURN;
 END;
```

Program notes

1. The error exits are set up.
2. The termid is tested to see whether this transaction is started from a terminal or at the printer.
3. A queue item (customer order) is read into the program.
4. The file control READ command reads the record into a record area so that the amount may be checked.
5. The amount (bank balance) is tested. If it is over $100 then the order is acceptable, otherwise the order is rejected. This test is only a suggestion; a suitable form of editing should be inserted here to ensure valid orders are sent to the warehouse.
6. The order details are moved to the map area for DFH$PGL.
7. The order map is sent to the printer terminal at the warehouse.
8. The current date and time, and details of the rejected order, are moved to a log record area.
9. The WRITEQ TD command writes details of the rejected order to LOGA, a transient data queue.
10. If the ERROR condition occurs on any CICS command a dump is taken and the program terminates.
11. When the queue is empty, the message 'ORDER QUEUE IS EMPTY' is moved to the map area which is then sent to the printer terminal at the warehouse.
12. The START command starts the PORQ transaction (this program), after a one hour delay, with a terminal identifier of L86O. (The time interval could be changed, for demonstration purposes, by changing the INTERVAL value.) If the comment delimiters are removed from the two preceding statements, EIBTIME is refreshed and, if the time is before 1400 hours, the transaction is started in one hour. If the comment delimiters are not removed, the transaction is started unconditionally in one hour.
13. The message 'PROCESSING COMPLETED' is sent to the terminal associated with this invocation of PORQ, either the printer at the warehouse or the screen on which PORQ was entered. The program terminates by returning control to CICS.

Low balance report sample program (PL/I)

Description

The low balance report sample program produces a report that lists all entries in the data set FILEA for which the amount is less than or equal to $50.00.

The program illustrates page building techniques and the use of the terminal paging facilities of BMS.

The transaction is invoked by entering PREP onto a clear screen. The program does a sequential scan through the file selecting each entry that obeys the search criterion.

The pages are built from four maps which comprise map set DFH$PGD, using the paging option so that the data is not displayed immediately but instead is stored for later retrieval. The HEADING map is inserted at the head of each page. The detail map (DFH$PGD) is written repeatedly until the OVERFLOW condition occurs. The FOOTING map is then written at the foot of the page and the HEADING map written at the top of the next page. The command to write the detail map that caused overflow is then repeated. When all the data has been written the FINAL map is written at the bottom of the last page and the transaction terminated.

The terminal operator then enters paging commands to display the data, clearing the screen before entering each paging command.

Source listing for DFH$PREP

```
 /********************************************************************/
 /*      DFH$PREP - CICS/VS SAMPLE FILEA LOW BALANCE INQUIRY - PL/I  */
 /********************************************************************/
  REPORT: PROC OPTIONS(MAIN);
   DCL LOWLIM CHAR(8) INIT('$0050.00');
   DCL KEYNUM PIC'999999' INIT(0);
   DCL PAGEN PIC'999' INIT(1);
   DCL OPINSTR CHAR(52) STATIC
       INIT('PRESS THE ENTER KEY AND FOLLOW WITH PAGING COMMANDS.');
   DCL TERM_DATA CHAR(1);
   DCL TERM_LENG FIXED BIN(15);
   DCL STRING CHAR(256) BASED;
   %INCLUDE DFH$PGD;
   %INCLUDE DFH$PFIL;
 1 EXEC CICS HANDLE CONDITION ERROR(ERRORS) OVERFLOW(OFLOW)
                           ENDFILE(ENDFILE) LENGERR(END_TASK);
   PAGENA=LOW(1);
 2 PAGENO=PAGEN;
 3 EXEC CICS SEND MAP('HEADING') MAPSET('DFH$PGD') ACCUM PAGING ERASE;
 4 EXEC CICS STARTBR DATASET('FILEA') RIDFLD(KEYNUM);
   REPEAT:
 5 EXEC CICS READNEXT INTO(FILEA) DATASET('FILEA') RIDFLD(KEYNUM);
 6    IF AMOUNT<=LOWLIM THEN
      DO;
         SUBSTR(ADDR(DFH$PGDO)->STRING,1,STG(DFH$PGDO))=
                                                  LOW(STG(DFH$PGDO));
 7       AMOUNTO = AMOUNT;
         NUMBERO = NUMB;
         NAMEO   = NAME;
 8        EXEC CICS SEND MAP('DFH$PGD') MAPSET('DFH$PGD') ACCUM PAGING;
      END;
   GOTO REPEAT;

  ENDFILE:
 9 EXEC CICS SEND MAP('FINAL') MAPSET('DFH$PGD') MAPONLY ACCUM PAGING;
10 EXEC CICS SEND PAGE;
11 EXEC CICS SEND TEXT FROM(OPINSTR) ERASE;
12 EXEC CICS ENDBR DATASET('FILEA');
                                  /* A RECEIVE IS ISSUED TO GIVE THE
                                     TERMINAL OPERATOR A CHANCE TO
                                     READ THE PROMPTING MESSAGE.
                                     THE TRANSACTION WILL TERMINATE
                                     WHEN THE OPERATOR PRESSES THE
                                     ENTER KEY                       */
                                  /* NO HARM DONE IF OPERATOR TYPES IN
                                     DATA IN ADDITION TO PRESSING THE
                                     ENTER KEY                       */
   TERM_LENG=1;
13 EXEC CICS RECEIVE INTO(TERM_DATA) LENGTH(TERM_LENG);
  END_TASK:
14 EXEC CICS RETURN;
  ERRORS:
15 EXEC CICS HANDLE CONDITION ERROR;
   EXEC CICS PURGE MESSAGE;
   EXEC CICS ABEND ABCODE('ERRS');
  OFLOW:
```

Source listing for DFH$PREP (continued)

```
16 EXEC CICS SEND MAP('FOOTING') MAPSET('DFH$PGD')
                       MAPONLY ACCUM PAGING;
   PAGEN  = PAGEN+1;
   PAGENA = LOW(1);
   PAGENO = PAGEN;
17 EXEC CICS SEND MAP('HEADING') MAPSET('DFH$PGD')
               ACCUM PAGING ERASE;
18 EXEC CICS SEND MAP('DFH$PGD') MAPSET('DFH$PGD') ACCUM PAGING;
   GOTO REPEAT;
   END;
```

Program notes

1. The program exits are set up.
2. A page number of 1 is moved to the heading map.
3. This BMS command sets up the heading in the page build operation. BMS builds the pages in temporary storage.
4. The STARTBR command sets up the file browse to begin at the first record with a key equal to or greater than the RIDFLD, in this case the first record on file.
5. This command reads the next customer record from FILEA.
6. The search criterion for creating the report is that the customer has a bank balance which is $50 or less.
7. Fields are moved from the selected customer record to the map area for the detail line.
8. The customer detail map is set up for subsequent paging.
9. When the ENDFILE condition is raised, the last map is sent to BMS.
10. The SEND PAGE command makes all the pages of the report available for paging, at the terminal, when the current transaction terminates.
11. A message is sent to the terminal. This message will be displayed before the pages of the low balance report.
12. The file browse is terminated.
13. This RECEIVE MAP command reads from the terminal and allows the terminal operator to read the prompting message before the first page of the report is displayed.
14. The program ends, the first page of the report will now be displayed.
15. If the ERROR condition occurs on a CICS command this routine gains control. Handling of the ERROR condition is suppressed, any data sent to BMS so far is purged and the program terminates abnormally with a transaction dump.
16. If the OVERFLOW condition occurs, when a detail line is sent to BMS, CICS branches here. This routine completes the current page and starts the next one. This BMS command sets up the footing for the current page.
17. This BMS command sets up the heading for the next page.
18. This BMS command resends the detail line which caused the OVERFLOW condition.

Maps and screen layouts for PL/I sample programs

The preceding sample programs assume that the following map sets have been cataloged with names the same as the map names.

The names of the source maps are all of the form DFH$PMx, whereas output generated by the assembly of maps is in the form DFH$PGx. Differing names are required for the map source and the generated dsect only if you wish to store both in the same source library.

DFH$PGA map definition

```
         TITLE 'FILEA - MAP FOR OPERATOR INSTRUCTIONS - PL/I'
MAPSETA  DFHMSD TYPE=&SYSPARM,MODE=INOUT,CTRL=(FREEKB,FRSET),LANG=PLI, *
               STORAGE=AUTO,TIOAPFX=YES,EXTATT=MAPONLY,COLOR=BLUE
DFH$PGA  DFHMDI SIZE=(12,40)
         DFHMDF POS=(1,10),LENGTH=21,INITIAL='OPERATOR INSTRUCTIONS',  *
               HILIGHT=UNDERLINE
         DFHMDF POS=(3,1),LENGTH=29,INITIAL='OPERATOR INSTR - ENTER PMN*
               U'
         DFHMDF POS=(4,1),LENGTH=38,INITIAL='FILE INQUIRY   - ENTER PIN*
               Q AND NUMBER'
         DFHMDF POS=(5,1),LENGTH=38,INITIAL='FILE BROWSE    - ENTER PBR*
               W AND NUMBER'
         DFHMDF POS=(6,1),LENGTH=38,INITIAL='FILE ADD       - ENTER PAD*
               D AND NUMBER'
         DFHMDF POS=(7,1),LENGTH=38,INITIAL='FILE UPDATE    - ENTER PUP*
               D AND NUMBER'
MSG      DFHMDF POS=(11,1),LENGTH=39,INITIAL='PRESS CLEAR TO EXIT'
         DFHMDF POS=(12,1),LENGTH=18,INITIAL='ENTER TRANSACTION:'
         DFHMDF POS=(12,20),LENGTH=4,ATTRB=IC,COLOR=GREEN,             *
               HILIGHT=REVERSE
         DFHMDF POS=(12,25),LENGTH=6,INITIAL='NUMBER'
KEY      DFHMDF POS=(12,32),LENGTH=6,ATTRB=NUM,COLOR=GREEN,            *
               HILIGHT=REVERSE
         DFHMDF POS=(12,39),LENGTH=1
         DFHMSD TYPE=FINAL
         END
```

The symbolic storage definition produced as a result of the above statements would be as follows:

DSECT generated by DFH$PGA

```
  DECLARE 1 DFH$PGAI AUTOMATIC UNALIGNED,
       2  DFHMS1 CHARACTER (12),
       2  MSGL    FIXED BINARY (15,0),
       2  MSGF    CHARACTER (1),
       2  MSGI  CHARACTER (39),
       2  KEYL    FIXED BINARY (15,0),
       2  KEYF    CHARACTER (1),
       2  KEYI  CHARACTER (6),
       2  FILL0055  CHARACTER (1);
  DECLARE 1 DFH$PGAO BASED(ADDR(DFH$PGAI)) UNALIGNED,
       2  DFHMS2 CHARACTER (12),
       2  DFHMS3 FIXED BINARY (15,0),
       2  MSGA    CHARACTER (1),
       2  MSGO  CHARACTER (39),
       2  DFHMS4 FIXED BINARY (15,0),
       2  KEYA    CHARACTER (1),
       2  KEYO  CHARACTER (6),
       2  FILL0055  CHARACTER (1);
 /* END OF MAP DEFINITION */
```

DFH$PGA screen layout

```
          +OPERATOR INSTRUCTIONS

+OPERATOR INSTR - ENTER PMNU
+FILE INQUIRY   - ENTER PINQ AND NUMBER
+FILE BROWSE    - ENTER PBRW AND NUMBER
+FILE ADD       - ENTER PADD AND NUMBER
+FILE UPDATE    - ENTER PUPD AND NUMBER



+PRESS CLEAR TO EXIT
+ENTER TRANSACTION:+XXXX+NUMBER+XXXXXX+
```

DFH$PGB map definition

```
         TITLE 'FILEA - MAP FOR FILE INQUIRY/UPDATE - PL/I'
MAPSETB  DFHMSD TYPE=&SYSPARM,MODE=INOUT,CTRL=(FREEKB,FRSET),LANG=PLI, *
               STORAGE=AUTO,TIOAPFX=YES,EXTATT=MAPONLY
DFH$PGB  DFHMDI SIZE=(12,40)
TITLE    DFHMDF POS=(1,15),LENGTH=12
         DFHMDF POS=(3,1),LENGTH=8,INITIAL='NUMBER:',COLOR=BLUE
NUMB     DFHMDF POS=(3,10),LENGTH=6
         DFHMDF POS=(3,17),LENGTH=1
         DFHMDF POS=(4,1),LENGTH=8,INITIAL='NAME:   ',COLOR=BLUE
NAME     DFHMDF POS=(4,10),LENGTH=20,ATTRB=(UNPROT,IC)
         DFHMDF POS=(4,31),LENGTH=1
         DFHMDF POS=(5,1),LENGTH=8,INITIAL='ADDRESS:',COLOR=BLUE
ADDR     DFHMDF POS=(5,10),LENGTH=20,ATTRB=UNPROT
         DFHMDF POS=(5,31),LENGTH=1
         DFHMDF POS=(6,1),LENGTH=8,INITIAL='PHONE:   ',COLOR=BLUE
PHONE    DFHMDF POS=(6,10),LENGTH=8,ATTRB=UNPROT
         DFHMDF POS=(6,19),LENGTH=1
         DFHMDF POS=(7,1),LENGTH=8,INITIAL='DATE:    ',COLOR=BLUE
DATE     DFHMDF POS=(7,10),LENGTH=8,ATTRB=UNPROT
         DFHMDF POS=(7,19),LENGTH=1
         DFHMDF POS=(8,1),LENGTH=8,INITIAL='AMOUNT:  ',COLOR=BLUE
AMOUNT   DFHMDF POS=(8,10),LENGTH=8,ATTRB=NUM
         DFHMDF POS=(8,19),LENGTH=1
         DFHMDF POS=(9,1),LENGTH=8,INITIAL='COMMENT:',COLOR=BLUE
COMMENT  DFHMDF POS=(9,10),LENGTH=9,ATTRB=UNPROT
         DFHMDF POS=(9,20),LENGTH=1
MSG1     DFHMDF POS=(11,1),LENGTH=39
MSG3     DFHMDF POS=(12,1),LENGTH=39
         DFHMSD TYPE=FINAL
         END
```

The symbolic storage definition produced as a result of the above statements would be as follows:

DSECT generated by DFH$PGB

```
DECLARE 1 DFH$PGBI AUTOMATIC UNALIGNED,
     2  DFHMS1 CHARACTER (12),
     2  TITLEL    FIXED BINARY (15,0),
     2  TITLEF    CHARACTER (1),
     2  TITLEI  CHARACTER (12),
     2  NUMBL    FIXED BINARY (15,0),
     2  NUMBF    CHARACTER (1),
     2  NUMBI  CHARACTER (6),
     2  NAMEL    FIXED BINARY (15,0),
     2  NAMEF    CHARACTER (1),
     2  NAMEI  CHARACTER (20),
     2  ADDRL    FIXED BINARY (15,0),
     2  ADDRF    CHARACTER (1),
     2  ADDRI  CHARACTER (20),
     2  PHONEL    FIXED BINARY (15,0),
     2  PHONEF    CHARACTER (1),
     2  PHONEI  CHARACTER (8),
     2  DATEL    FIXED BINARY (15,0),
     2  DATEF    CHARACTER (1),
     2  DATEI  CHARACTER (8),
     2  AMOUNTL    FIXED BINARY (15,0),
     2  AMOUNTF    CHARACTER (1),
     2  AMOUNTI  CHARACTER (8),
     2  COMMENTL    FIXED BINARY (15,0),
     2  COMMENTF    CHARACTER (1),
     2  COMMENTI  CHARACTER (9),
     2  MSG1L    FIXED BINARY (15,0),
     2  MSG1F    CHARACTER (1),
     2  MSG1I  CHARACTER (39),
     2  MSG3L    FIXED BINARY (15,0),
     2  MSG3F    CHARACTER (1),
     2  MSG3I  CHARACTER (39),
     2  FILL0092  CHARACTER (1);
DECLARE 1 DFH$PGBO BASED(ADDR(DFH$PGBI)) UNALIGNED,
     2  DFHMS2 CHARACTER (12),
     2  DFHMS3 FIXED BINARY (15,0),
     2  TITLEA    CHARACTER (1),
     2  TITLEO  CHARACTER (12),
     2  DFHMS4 FIXED BINARY (15,0),
     2  NUMBA    CHARACTER (1),
     2  NUMBO  CHARACTER (6),
     2  DFHMS5 FIXED BINARY (15,0),
     2  NAMEA    CHARACTER (1),
     2  NAMEO  CHARACTER (20),
     2  DFHMS6 FIXED BINARY (15,0),
     2  ADDRA    CHARACTER (1),
     2  ADDRO  CHARACTER (20),
     2  DFHMS7 FIXED BINARY (15,0),
     2  PHONEA    CHARACTER (1),
     2  PHONEO  CHARACTER (8),
     2  DFHMS8 FIXED BINARY (15,0),
     2  DATEA    CHARACTER (1),
     2  DATEO  CHARACTER (8),
     2  DFHMS9 FIXED BINARY (15,0),
     2  AMOUNTA    CHARACTER (1),
     2  AMOUNTO  CHARACTER (8),
     2  DFHMS10 FIXED BINARY (15,0),
```

DSECT generated by DFH$PGB (continued)

```
          2  COMMENTA    CHARACTER (1),
          2  COMMENTO  CHARACTER (9),
          2  DFHMS11 FIXED BINARY (15,0),
          2  MSG1A    CHARACTER (1),
          2  MSG1O  CHARACTER (39),
          2  DFHMS12 FIXED BINARY (15,0),
          2  MSG3A    CHARACTER (1),
          2  MSG3O  CHARACTER (39),
          2  FILL0092  CHARACTER (1);
    /* END OF MAP DEFINITION */
```

DFH$PGB screen layout

```
          +XXXXXXXXXXXX

+NUMBER: +XXXXXX+
+NAME:    +XXXXXXXXXXXXXXXXXXXX+
+ADDRESS:+XXXXXXXXXXXXXXXXXXXX+
+PHONE:   +XXXXXXXX+
+DATE:    +XXXXXXXX+
+AMOUNT: +XXXXXXXX+
+COMMENT:+XXXXXXXXX+

+XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
+XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

DFH$PGC map definition

```
         TITLE 'FILEA - MAP FOR FILE BROWSE - PL/I'
MAPSETC  DFHMSD TYPE=&SYSPARM,MODE=INOUT,CTRL=(FREEKB,FRSET),LANG=PLI, *
               STORAGE=AUTO,TIOAPFX=YES,EXTATT=MAPONLY
DFH$PGC  DFHMDI SIZE=(12,40)
DIR      DFHMDF POS=(1,1),LENGTH=1,ATTRB=IC
         DFHMDF POS=(1,3),LENGTH=1
         DFHMDF POS=(1,15),LENGTH=11,INITIAL='FILE BROWSE',            *
               COLOR=BLUE,HILIGHT=UNDERLINE
         DFHMDF POS=(3,1),LENGTH=6,INITIAL='NUMBER',COLOR=BLUE
         DFHMDF POS=(3,17),LENGTH=4,INITIAL='NAME',COLOR=BLUE
         DFHMDF POS=(3,32),LENGTH=6,INITIAL='AMOUNT',COLOR=BLUE
NUMBER1  DFHMDF POS=(4,1),LENGTH=6
NAME1    DFHMDF POS=(4,9),LENGTH=20
AMOUNT1  DFHMDF POS=(4,30),LENGTH=8
NUMBER2  DFHMDF POS=(5,1),LENGTH=6
NAME2    DFHMDF POS=(5,9),LENGTH=20
AMOUNT2  DFHMDF POS=(5,30),LENGTH=8
NUMBER3  DFHMDF POS=(6,1),LENGTH=6
NAME3    DFHMDF POS=(6,9),LENGTH=20
AMOUNT3  DFHMDF POS=(6,30),LENGTH=8
NUMBER4  DFHMDF POS=(7,1),LENGTH=6
NAME4    DFHMDF POS=(7,9),LENGTH=20
AMOUNT4  DFHMDF POS=(7,30),LENGTH=8
MSG0     DFHMDF POS=(10,1),LENGTH=39,COLOR=BLUE,                      *
               INITIAL='PRESS CLEAR TO END BROWSE OPERATION'
MSG1     DFHMDF POS=(11,1),LENGTH=39,COLOR=BLUE,                      *
               INITIAL='PRESS PF1 OR TYPE F TO PAGE FORWARD'
MSG2     DFHMDF POS=(12,1),LENGTH=39,COLOR=BLUE,                      *
               INITIAL='PRESS PF2 OR TYPE B TO PAGE BACKWARD'
         DFHMSD TYPE=FINAL
         END
```

The symbolic storage definition produced as a result of the above statements would be as follows:

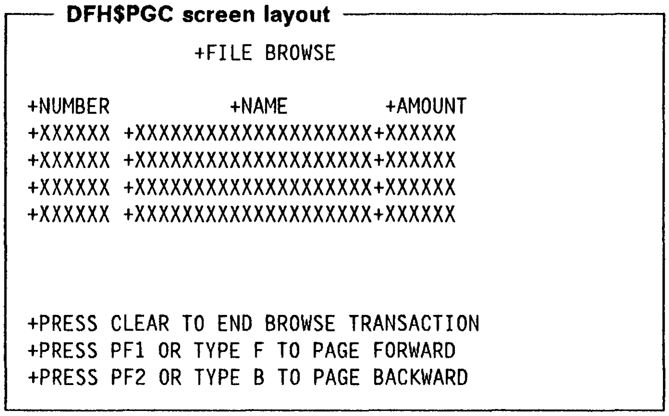
DSECT generated by DFH$PGC

```
DECLARE 1 DFH$PGCI AUTOMATIC UNALIGNED,
     2  DFHMS1 CHARACTER (12),
     2  DIRL    FIXED BINARY (15,0),
     2  DIRF    CHARACTER (1),
     2  DIRI  CHARACTER (1),
     2  NUMBER1L    FIXED BINARY (15,0),
     2  NUMBER1F    CHARACTER (1),
     2  NUMBER1I  CHARACTER (6),
     2  NAME1L    FIXED BINARY (15,0),
     2  NAME1F    CHARACTER (1),
     2  NAME1I  CHARACTER (20),
     2  AMOUNT1L    FIXED BINARY (15,0),
     2  AMOUNT1F    CHARACTER (1),
     2  AMOUNT1I  CHARACTER (8),
     2  NUMBER2L    FIXED BINARY (15,0),
     2  NUMBER2F    CHARACTER (1),
     2  NUMBER2I  CHARACTER (6),
     2  NAME2L    FIXED BINARY (15,0),
     2  NAME2F    CHARACTER (1),
     2  NAME2I  CHARACTER (20),
     2  AMOUNT2L    FIXED BINARY (15,0),
     2  AMOUNT2F    CHARACTER (1),
     2  AMOUNT2I  CHARACTER (8),
     2  NUMBER3L    FIXED BINARY (15,0),
     2  NUMBER3F    CHARACTER (1),
     2  NUMBER3I  CHARACTER (6),
     2  NAME3L    FIXED BINARY (15,0),
     2  NAME3F    CHARACTER (1),
     2  NAME3I  CHARACTER (20),
     2  AMOUNT3L    FIXED BINARY (15,0),
     2  AMOUNT3F    CHARACTER (1),
     2  AMOUNT3I  CHARACTER (8),
     2  NUMBER4L    FIXED BINARY (15,0),
     2  NUMBER4F    CHARACTER (1),
     2  NUMBER4I  CHARACTER (6),
     2  NAME4L    FIXED BINARY (15,0),
     2  NAME4F    CHARACTER (1),
     2  NAME4I  CHARACTER (20),
     2  AMOUNT4L    FIXED BINARY (15,0),
     2  AMOUNT4F    CHARACTER (1),
     2  AMOUNT4I  CHARACTER (8),
     2  MSG0L    FIXED BINARY (15,0),
     2  MSG0F    CHARACTER (1),
     2  MSG0I  CHARACTER (39),
     2  MSG1L    FIXED BINARY (15,0),
     2  MSG1F    CHARACTER (1),
     2  MSG1I  CHARACTER (39),
     2  MSG2L    FIXED BINARY (15,0),
     2  MSG2F    CHARACTER (1),
     2  MSG2I  CHARACTER (39),
     2  FILL0084  CHARACTER (1);
DECLARE 1 DFH$PGCO BASED(ADDR(DFH$PGCI)) UNALIGNED,
     2  DFHMS2 CHARACTER (12),
     2  DFHMS3 FIXED BINARY (15,0),
     2  DIRA    CHARACTER (1),
     2  DIRO  CHARACTER (1),
     2  DFHMS4 FIXED BINARY (15,0),
     2  NUMBER1A    CHARACTER (1),
     2  NUMBER1O  CHARACTER (6),
```

DSECT generated by DFH$PGC (continued)

```
      2  DFHMS5 FIXED BINARY (15,0),
      2  NAME1A    CHARACTER (1),
      2  NAME1O  CHARACTER (20),
      2  DFHMS6 FIXED BINARY (15,0),
      2  AMOUNT1A    CHARACTER (1),
      2  AMOUNT1O  CHARACTER (8),
      2  DFHMS7 FIXED BINARY (15,0),
      2  NUMBER2A    CHARACTER (1),
      2  NUMBER2O  CHARACTER (6),
      2  DFHMS8 FIXED BINARY (15,0),
      2  NAME2A    CHARACTER (1),
      2  NAME2O  CHARACTER (20),
      2  DFHMS9 FIXED BINARY (15,0),
      2  AMOUNT2A    CHARACTER (1),
      2  AMOUNT2O  CHARACTER (8),
      2  DFHMS10 FIXED BINARY (15,0),
      2  NUMBER3A    CHARACTER (1),
      2  NUMBER3O  CHARACTER (6),
      2  DFHMS11 FIXED BINARY (15,0),
      2  NAME3A    CHARACTER (1),
      2  NAME3O  CHARACTER (20),
      2  DFHMS12 FIXED BINARY (15,0),
      2  AMOUNT3A    CHARACTER (1),
      2  AMOUNT3O  CHARACTER (8),
      2  DFHMS13 FIXED BINARY (15,0),
      2  NUMBER4A    CHARACTER (1),
      2  NUMBER4O  CHARACTER (6),
      2  DFHMS14 FIXED BINARY (15,0),
      2  NAME4A    CHARACTER (1),
      2  NAME4O  CHARACTER (20),
      2  DFHMS15 FIXED BINARY (15,0),
      2  AMOUNT4A    CHARACTER (1),
      2  AMOUNT4O  CHARACTER (8),
      2  DFHMS16 FIXED BINARY (15,0),
      2  MSG0A    CHARACTER (1),
      2  MSG0O  CHARACTER (39),
      2  DFHMS17 FIXED BINARY (15,0),
      2  MSG1A    CHARACTER (1),
      2  MSG1O  CHARACTER (39),
      2  DFHMS18 FIXED BINARY (15,0),
      2  MSG2A    CHARACTER (1),
      2  MSG2O  CHARACTER (39),
      2  FILL0084  CHARACTER (1);
/* END OF MAP DEFINITION */
```

DFH$PGC screen layout



```
              +FILE BROWSE

+NUMBER          +NAME        +AMOUNT
+XXXXXX +XXXXXXXXXXXXXXXXXXXX+XXXXXX
+XXXXXX +XXXXXXXXXXXXXXXXXXXX+XXXXXX
+XXXXXX +XXXXXXXXXXXXXXXXXXXX+XXXXXX
+XXXXXX +XXXXXXXXXXXXXXXXXXXX+XXXXXX




+PRESS CLEAR TO END BROWSE TRANSACTION
+PRESS PF1 OR TYPE F TO PAGE FORWARD
+PRESS PF2 OR TYPE B TO PAGE BACKWARD
```

DFH$PGD map definition

```
         TITLE 'FILEA - MAPSET FOR LOW BALANCE REPORT - PL/I'
MAPSETD  DFHMSD TYPE=&SYSPARM,MODE=OUT,CTRL=(FREEKB,FRSET),LANG=PLI,   *
               STORAGE=AUTO,TIOAPFX=YES,EXTATT=MAPONLY,COLOR=BLUE
DFH$PGD  DFHMDI SIZE=(1,40),COLOR=GREEN
NUMBER   DFHMDF POS=(1,1),LENGTH=6
NAME     DFHMDF POS=(1,9),LENGTH=20
AMOUNT   DFHMDF POS=(1,30),LENGTH=8
HEADING  DFHMDI SIZE=(3,40),HEADER=YES
         DFHMDF POS=(1,5),LENGTH=18,INITIAL='LOW BALANCE REPORT',      *
               HILIGHT=UNDERLINE
         DFHMDF POS=(1,30),LENGTH=4,INITIAL='PAGE'
PAGEN    DFHMDF POS=(1,35),LENGTH=3
         DFHMDF POS=(3,1),LENGTH=6,INITIAL='NUMBER'
         DFHMDF POS=(3,17),LENGTH=4,INITIAL='NAME'
         DFHMDF POS=(3,32),LENGTH=6,INITIAL='AMOUNT'
FOOTING  DFHMDI SIZE=(2,40),TRAILER=YES,JUSTIFY=LAST
         DFHMDF POS=(2,1),LENGTH=38,                                   *
               INITIAL='PRESS CLEAR AND TYPE P/N TO SEE PAGE N'
FINAL    DFHMDI SIZE=(2,40),TRAILER=YES,JUSTIFY=LAST
         DFHMDF POS=(2,10),LENGTH=14,INITIAL='END OF REPORT.'
         DFHMSD TYPE=FINAL
         END
```

The symbolic storage definition produced as a result of the above statements would be as follows:

DSECT generated by DFH$PGD

```
 DECLARE 1 DFH$PGDO AUTOMATIC UNALIGNED,
      2  DFHMS1 CHARACTER (12),
      2  DFHMS2 FIXED BINARY (15,0),
      2  NUMBERA    CHARACTER (1),
      2  NUMBERO  CHARACTER (6),
      2  DFHMS3 FIXED BINARY (15,0),
      2  NAMEA    CHARACTER (1),
      2  NAMEO  CHARACTER (20),
      2  DFHMS4 FIXED BINARY (15,0),
      2  AMOUNTA    CHARACTER (1),
      2  AMOUNTO  CHARACTER (8),
      2  FILL0022  CHARACTER (1);
/* END OF MAP DEFINITION */
 DECLARE 1 HEADINGO AUTOMATIC UNALIGNED,
      2  DFHMS5 CHARACTER (12),
      2  DFHMS6 FIXED BINARY (15,0),
      2  PAGENA    CHARACTER (1),
      2  PAGENO  CHARACTER (3),
      2  FILL0043  CHARACTER (1);
/* END OF MAP DEFINITION */
 DECLARE 1 FOOTINGO AUTOMATIC UNALIGNED,
      2  DFHMS7 CHARACTER (12),
      2  FILL0049  CHARACTER (1);
/* END OF MAP DEFINITION */
 DECLARE 1 FINALO AUTOMATIC UNALIGNED,
      2  DFHMS8 CHARACTER (12),
      2  FILL0057  CHARACTER (1);
/* END OF MAP DEFINITION */
```

DFH$PGD screen layout

```
     +LOW BALANCE REPORT       +PAGE+XXX

+NUMBER          +NAME           +AMOUNT
+XXXXXX+XXXXXXXXXXXXXXXXXXXX   XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX   XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX   XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX   XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX   XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX   XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX   XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX   XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX   XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX   XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX   XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX   XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX   XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX   XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX   XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX   XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX   XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX   XXXXXXXX
+XXXXXX+XXXXXXXXXXXXXXXXXXXX   XXXXXXXX

+PRESS CLEAR AND TYPE P/N TO SEE PAGE N
```

DFH$PGK map definition

```
         TITLE 'FILEA - MAP FOR ORDER ENTRY - PL/I'
MAPSETK  DFHMSD TYPE=&SYSPARM,MODE=INOUT,CTRL=(FREEKB,FRSET),LANG=PLI, *
               STORAGE=AUTO,TIOAPFX=YES,EXTATT=MAPONLY
DFH$PGK  DFHMDI SIZE=(12,40)
         DFHMDF POS=(01,10),LENGTH=11,ATTRB=(BRT,ASKIP),               *
               INITIAL='ORDER ENTRY',COLOR=BLUE,HILIGHT=UNDERLINE
MSG1     DFHMDF POS=(03,04),LENGTH=26,ATTRB=(DRK,ASKIP),               *
               INITIAL='NUMBER NOT FOUND - REENTER',                   *
               COLOR=RED,HILIGHT=BLINK
MSG2     DFHMDF POS=(04,04),LENGTH=22,ATTRB=(DRK,ASKIP),               *
               INITIAL='DATA ERROR - REENTER',                         *
               COLOR=RED,HILIGHT=BLINK
         DFHMDF POS=(05,04),LENGTH=09,ATTRB=PROT,                      *
               INITIAL='NUMBER  :'
CUSTNO   DFHMDF POS=(05,14),LENGTH=06,ATTRB=(IC,NUM)
         DFHMDF POS=(05,21),LENGTH=01
         DFHMDF POS=(06,04),LENGTH=09,ATTRB=PROT,COLOR=BLUE,           *
               INITIAL='PART NO :'
PARTNO   DFHMDF POS=(06,14),LENGTH=06,ATTRB=NUM
         DFHMDF POS=(06,21),LENGTH=01
         DFHMDF POS=(07,04),LENGTH=09,ATTRB=PROT,COLOR=BLUE,           *
               INITIAL='QUANTITY:'
QUANT    DFHMDF POS=(07,14),LENGTH=06,ATTRB=NUM
         DFHMDF POS=(07,21),LENGTH=01
         DFHMDF POS=(09,01),LENGTH=38,ATTRB=ASKIP,COLOR=BLUE,          *
               INITIAL='PRESS ENTER TO CONTINUE,CLEAR TO QUIT'
         DFHMSD TYPE=FINAL
         END
```

The symbolic storage definition generated as a result of the above statements would be as follows:

DSECT generated by DFH$PGK

```
 DECLARE 1 DFH$PGKI AUTOMATIC UNALIGNED,
      2  DFHMS1 CHARACTER (12),
      2  MSG1L    FIXED BINARY (15,0),
      2  MSG1F    CHARACTER (1),
      2  MSG1I  CHARACTER (26),
      2  MSG2L    FIXED BINARY (15,0),
      2  MSG2F    CHARACTER (1),
      2  MSG2I  CHARACTER (22),
      2  CUSTNOL    FIXED BINARY (15,0),
      2  CUSTNOF    CHARACTER (1),
      2  CUSTNOI  CHARACTER (6),
      2  PARTNOL    FIXED BINARY (15,0),
      2  PARTNOF    CHARACTER (1),
      2  PARTNOI  CHARACTER (6),
      2  QUANTL    FIXED BINARY (15,0),
      2  QUANTF    CHARACTER (1),
      2  QUANTI  CHARACTER (6),
      2  FILL0061  CHARACTER (1);
 DECLARE 1 DFH$PGKO BASED(ADDR(DFH$PGKI)) UNALIGNED,
      2  DFHMS2 CHARACTER (12),
      2  DFHMS3 FIXED BINARY (15,0),
      2  MSG1A    CHARACTER (1),
      2  MSG1O  CHARACTER (26),
      2  DFHMS4 FIXED BINARY (15,0),
      2  MSG2A    CHARACTER (1),
      2  MSG2O  CHARACTER (22),
      2  DFHMS5 FIXED BINARY (15,0),
      2  CUSTNOA    CHARACTER (1),
      2  CUSTNOO  CHARACTER (6),
      2  DFHMS6 FIXED BINARY (15,0),
      2  PARTNOA    CHARACTER (1),
      2  PARTNOO  CHARACTER (6),
      2  DFHMS7 FIXED BINARY (15,0),
      2  QUANTA    CHARACTER (1),
      2  QUANTO  CHARACTER (6),
      2  FILL0061  CHARACTER (1);
/* END OF MAP DEFINITION */
```

DFH$PGK screen layout

```
         +ORDER ENTRY

    +NUMBER NOT FOUND  REENTER
    +DATA ERROR  REENTER
    +NUMBER  :+XXXXXX+
    +PART NO :+XXXXXX+
    +QUANTITY:+XXXXXX+

+PRESS ENTER TO CONTINUE, CLEAR TO QUIT
```

DFH$PGL map definition

```
         TITLE 'FILEA - MAP FOR ORDER ENTRY QUEUE PRINT - PL/I'
MAPSETL  DFHMSD TYPE=&SYSPARM,MODE=OUT,LANG=PLI,                       *
               STORAGE=AUTO,TIOAPFX=YES
DFH$PGL  DFHMDI SIZE=(05,80)
TITLE    DFHMDF POS=(01,01),LENGTH=43,                                 *
               INITIAL='NUMBER     NAME                     ADDRESS'
NUMB     DFHMDF POS=(02,01),LENGTH=06
NAM      DFHMDF POS=(02,12),LENGTH=20
ADDR     DFHMDF POS=(02,37),LENGTH=20
         DFHMDF POS=(03,01),LENGTH=09,                                 *
               INITIAL='PART NO :'
PART     DFHMDF POS=(03,11),LENGTH=06
         DFHMDF POS=(04,01),LENGTH=09,                                 *
               INITIAL='QUANTITY:'
QUANT    DFHMDF POS=(04,11),LENGTH=06
         DFHMDF POS=(05,01),LENGTH=1,                                  *
               INITIAL=' '
         DFHMSD TYPE=FINAL
         END
```

The symbolic storage definition generated as a result of the above statements would be as follows:

DSECT generated by DFH$PGL

```
 DECLARE 1 DFH$PGLO AUTOMATIC UNALIGNED,
      2  DFHMS1 CHARACTER (12),
      2  DFHMS2 FIXED BINARY (15,0),
      2  TITLEA    CHARACTER (1),
      2  TITLEO  CHARACTER (43),
      2  DFHMS3 FIXED BINARY (15,0),
      2  NUMBA    CHARACTER (1),
      2  NUMBO  CHARACTER (6),
      2  DFHMS4 FIXED BINARY (15,0),
      2  NAMA    CHARACTER (1),
      2  NAMO  CHARACTER (20),
      2  DFHMS5 FIXED BINARY (15,0),
      2  ADDRA    CHARACTER (1),
      2  ADDRO  CHARACTER (20),
      2  DFHMS6 FIXED BINARY (15,0),
      2  PARTA    CHARACTER (1),
      2  PARTO  CHARACTER (6),
      2  DFHMS7 FIXED BINARY (15,0),
      2  QUANTA    CHARACTER (1),
      2  QUANTO  CHARACTER (6),
      2  FILL0039  CHARACTER (1);
/* END OF MAP DEFINITION */
```

DFH$PGL print format

```
+NUMBER     NAME                    ADDRESS
+xxxxxx    +xxxxxxxxxxxxxxxxxxxx    +xxxxxxxxxxxxxxxxxxxx
+PART NO :+xxxxxx
+QUANTITY:+xxxxxx
+x
```

Record descriptions for PL/I sample programs

FILEA record description

The FILEA record description is used by the sample programs. It is defined in copy code DFH$PFIL and has the following format:

```
DCL 1 FILEA,
      2 FILEREC,
        3 STAT CHAR(1),
        3 NUMB PIC'(6)9',
        3 NAME CHAR(20),
        3 ADDRX CHAR(20),
        3 PHONE CHAR(8),
        3 DATEX CHAR(8),
        3 AMOUNT CHAR(8),
        3 COMMENT CHAR(9);
```

LOGA record description

The LOGA record description is used by the sample programs when an audit trail is written to a transient data file. It is defined in copy code DFH$PLOG and has the following format:

```
DCL 1 LOGA,
      2 LOGHDR,
        3 LDAY FIXED DEC (7,0),
        3 LTIME FIXED DEC (7,0),
        3 LTERML CHAR(4),
        2 LOGREC,
        3 LSTAT CHAR(1),
        3 LNUMB CHAR(6),
        3 LNAME CHAR(20),
        3 LADDR CHAR(20),
        3 LPHONE CHAR(8),
        3 LDATE CHAR(8),
        3 LAMOUNT CHAR(8),
        3 LCOMMENT CHAR(9);
```

L86O record description

The L86O record description is used by the Order Entry Queue Print sample program when it writes to the transient data queue ‘L86O’. It is defined in copy code DFH$PL86 and has the following format:

```
DCL 1 L86O,
      2 ITEM,
        3 CUSTNO CHAR(6),
        3 PARTNO CHAR(6),
        3 QUANTITY CHAR(6),
        3 TERMID CHAR(4);
```

Index

A

ABCODE option 53, 315
ABEND command 314
abend user task (EDF) 61
ABEND (abnormal termination) exit 313
abnormal termination
  exceptional condition 315
  options 315
  reactivate an exit 313
  recovery 313
absolute expression 6
ABSTIME option 280
access to DL/I database
  DL/I CALL statement 117
  EXEC DL/I command 101
access to system information
  ADDRESS command 51
  ASSIGN command 52
  CICS storage areas 51
  EXEC interface block (EIB) 51
ACCOUNT option 321
ACCUM option 176, 213
activate an ABEND exit 313
active partition 162, 164
ACTPARTN option 167, 213
adding records
  to BDAM data set 90
  to BDI data set 266
  to VSAM data set 82
address
  cursor 232
  PCB 118
ADDRESS command 51
ADDRESS register, VS COBOL II 24
AID (see attention identifier)
AIX (alternate index) 78
ALARM option 152, 213
ALIGNED attribute (PL/I) 7
ALL option 213
ALLOCATE command 236, 238
alternate facility 221
alternate index path 78
alternate index (AIX) 78
alternate key 78
alternate screen size 142
ALTSCRN operand 205
ampersand (CL interpreter) 70
ANSI85 standards, VS COBOL II 36—40
ANSI85 translator option 14
ANYKEY option 155, 233
API (application programming interface) 51
APLG abend 41
APOST option 15
application data area of screen 142
application partition set 164, 167
application program logical levels 32, 35, 289
application program using commands and macros 13
application programming interface (API) 51
application-oriented information (LU6) 237
APPLID option 53
argument value
  assembler language 6
  COBOL 6
  PL/I 7
ASIS option
  basic mapping support 154, 213
  terminal control 257
ASKTIME command 275
Assembler
  DL/I CALL interface 20
assembler language
  argument value 6
  coding conventions 5
  LENGTH option default 6
  program exit 10
  programming techniques 19
  register contents 10
  restrictions 19
  sample programs 353
  translated code 10
assembling a map 146
assembling a partition set 166
ASSIGN command 52
asynchronous interrupt 223
asynchronous journal output 327
asynchronous page build 156
ATI (see automatic task initiation)
ATTACHID option 257
attention condition (SIGNAL) 228
attention identifier (AID)
  constants 208
  HANDLE AID command 154
  input without data 232
  list (DFHAID) 232
  3270 input operation 139
ATTRB operand 195, 205
attribute character 140, 150
attribute constants 150
attribute control character list (DFHBMSCA) 205, 233
attribute, extended 160
audible alarm (MSR) 170

Audio Response Unit (7770) 256
audio terminal (2721) 256
autoanswer transaction (3735) 251
autocall transaction (3735) 251
automatic task initiation (ATI) 301
AUTOPAGE option 174, 213
autoskip field 140
AUXILIARY option 307
auxiliary storage temporary data 305
auxiliary trace facility 319

B

background transparency 141
backout of resources 331
backout updates (DL/I) 103
base color 140
base locator for linkage (BLL)
  chained storage areas 22
  large storage areas 23
  OCCURS DEPENDING ON clauses 22
  storage addressing 21
BASE operand 196
base state 165
basic mapping support (BMS)
  an introduction 135
  assembling maps 146
  block data format 172
  BMS and GDDM 145
  cataloging maps 146
  CMSG message switching transaction 189
  completing a logical message 174
  coordinating BMS and another screen manager 153
  cumulative mapping 176
  cumulative output processing 176
  cursor position 153
  deleting a logical message 174
  determining the actual input partition (RECEIVE PARTN) 209
  device control options 152
  exceptional conditions 156, 217
  field data format 149
  field definition macro 145, 194
  field group 145
  floating maps 176
  full function BMS 173
  GDDM coordination 145, 153
  header and trailer maps 176
  input field suffix 149
  input partition (RECEIVE PARTN) 168
  invalid data 150
  loading a partition set 167
  logical device components 169
  logical message 173
  map definition macro 144, 193
  map positioning 177
basic mapping support (BMS) *(continued)*
  map set definition macro 143, 193
  map set suffixing 146
  map set termination 146
  map sets 135
  map size 19
  mapping input data 153, 209
  mapping output data 150
  maps 148
  minimum function BMS 139
  null map 144
  options 213
  outboard formatting 172
  output field suffixes 149
  page overflow 176, 180
  partition definition macro 166, 204
  partition set definition macro 165, 204
  physical map 146
  pregenerated versions 137
  printer support 160
  returning mapped data to a program 189
  routing a logical message (ROUTE) 184
  SCS and non-3270 printers 161
  sending a user defined data stream (SEND TEXT NOEDIT) 191
  sending data previously mapped by BMS (SEND TEXT MAPPED) 190
  sending data to a display 150
  sending device controls without data (SEND CONTROL) 153
  sending text data (SEND TEXT) 159
  standard function BMS 159
  symbolic map 146, 148
  terminal code table 190
  terminal operator paging 175
  3270 printer using NLEOM 161
  3270 printer without NLEOM 160
batch compilation, VS COBOL II (ANSI85) 37
batch data interchange 265
  add record to data set 266
  delete a record from data set 267
  destination identification 265
  exceptional conditions 269
  interrogate a data set 266
  options 268
  read record from data set 266
  request next record number 268
  send data to output device 268
  terminate data set 267
  update a record in data set 266
  wait for function completion 268
batch logical unit (3770) 253
batch mode application (3740) 252
BDAM
  browsing operations 89

BDAM *(continued)*
- data sets 78, 89, 90
- exclusive control 91

BIF DEEDIT (built-in function) 335
blank lines and 3270 printer 157
blank lines, VS COBOL II (ANSI85) 36
BLL (see base locator for linkage)
block data format (BMS) 172
block reference 89
blue parameter of COLOR operand 196
BMS logical message (see logical messages)
BMS (see basic mapping support)
Boolean expression 104
BOTTOM command (CEBR) 73
bracket protocol (LAST option) 227
bright intensity field 140
browse operation
- BDAM 89
- ending 96
- read next record during 95
- read previous record 96
- reset starting point 96
- specify starting point 95
- VSAM 80

browse temporary storage (EDF) 61
browse transaction (CEBR) 73
BTAM programmable device 229
BTRANS option 53
BUFFER option 257
BUFSZE operand 205
BUILD ATTACH command 236
built-in function (BIF DEEDIT) 335

C

CALL statement 20
- in VS COBOL II 28, 32–36

CALLDLI macro 117
CANCEL command 280
CANCEL option 315
CARD option 268
cataloging a map 146
cataloging a partition set 166
CBIDERR condition 262
CBL statement (COBOL) 13
CBUFF option 257
CEBR (browse transaction) 73
CECI (see command level interpreter) 67
CECS (see command level interpreter) 67
CEDF transaction 58
cell size for partitions 168
chained storage area, COBOL 22
chaining of data 226
character attribute 160
character cell size 165, 168
CHARSZE operand 205
checking a DL/I call 120
checkout, program 57
CICS and the XRF environment 53
CICS option 14
CLEAR key 155, 162
CLEAR option 155, 233
CLEAR PARTITION key 162
CLRPARTN option 233
CMSG message switching transaction 189
COBOL
- argument value 6
- base locator for linkage (BLL) 21
- compilers supported 21
- program segments 24
- restrictions 20
- sample programs 393
- translated code 13

COBOL2 option 14
CODEREG argument 12
coding conventions 5
COLOR operand 196
COLOR option 53
COLUMN command (CEBR) 73
COLUMN operand 177, 197
comma and semicolon as delimiters, VS COBOL II (ANSI85) 40
command
- argument values 5
- end-of-command delimiter 5
- execution (CL interpreter) 69
- format 5
- macro equivalent 351
- syntax check 68

command language translator
- data set 9
- optional facilities 13
- translated code 10

command level interpreter (CECI/CECS)
- an introduction 67
- command input area 67
- information area 68
- installing 72
- invoking 67
- PF key values area 71
- program control 72
- screen layout 67
- security rules 72
- status area 68
- terminal sharing 72
- variables 70

COMMAREA
- length of 19

COMMAREA option 296

common buffer, output to (2980) 243
communication area (DFHCOMMAREA) 21
communication line, relinquishing 223
compiler options 14
compilers supported
  COBOL 21
  PL/I 42
COMPLETE option 324
conditions (exceptional conditions) 43
CONFIRM option 257
CONNECT PROCESS command 238
CONSOLE option 268
constants
  AID values (DFHAID) 208
  attribute values (DFHBMSCA) 205
  for examining EIBAID field 155, 208
  for MSR control values 208
  for printer format controls 205
  for 3270 attributes 205
  MSR control (DFHMSRCA) 208
  printer control values (DFHBMSCA) 205
  3270 attributes 150
control
  exclusive, BDAM 91
  exclusive, VSAM 87
  pass with return 289
  pass without return 290
  return 291
  trace 320
control area, VSAM 78
control interval, VSAM 78
CONVERSE command 223
CONVERSE option 257
converse with terminal or LU 223
CONVID option 257
copy
  displayed information 232
  symbolic description map 148
copy book
  DFHAID 208
  DFHBMSCA 150, 205
  DFHEIBLK 10
  DFHMSRCA 208
copybook DFHAID 155
create a journal record 328
CSA option 51
CTLCHAR option 257
CTRL operand 197
cumulative output processing 176
cumulative text processing 181
CURRENT option 213
CURSOR option 152, 213
cursor position
  basic mapping support 153
  terminal control 232
CURSOR SELECT key 155
CWA option 51
CWALENG option 53

D

data
  chaining 226
  temporary storage 305
data communication operations 133
data comparison 163
data definition 17
data fields on screen 142
data initialization 17
data integrity 78
data interchange (see batch data interchange)
DATA operand 197
DATA option 300
data records
data set
  access from CICS application programs 79
  batch data interchange 265
  BDAM 89
  blocked 78
  identification 79
  translator 9
  VSAM 87
data-area argument 5
data-value argument 5
DATAONLY option 151, 213
DATAREG argument 12
DATASTR option 257
data, passing to other program 292
date field of EIB 51
DATE option 280
DATEFORM option 280
DATESEP option 280
DAYCOUNT option 280
DAYOFMONTH option 280
DAYOFWEEK option 280
DCT option 324
DDMMYY option 281
deadlock prevention 87
DEBKEY option 97
deblocking argument 89
DEBREC option 97
DEBUG option 14
debugging 57, 309
default action for conditions 43
default screen size 142
deferred journal output 327
deferred light pen field 155
defining a map set 143
defining partition sets 165
definite response protocol
  batch data interchange 265

definite response protocol *(continued)*
  terminal control 227
DEFRESP option
  batch data interchange 268
  terminal control 227, 258
DELAY command 276
delay processing of task 276
DELETE command 94
delete lock 87
DELETEQ TD command 303
DELETEQ TS command 307
deleting
  batch data interchange record 267
  file control record 94
  loaded program 291
  temporary storage queue 307
  transient data queue 303
DELIMITER option 53
delimiter, end-of-command 5
DEQ command 285
dequeue from resource 285
DEST option 258
DESTCOUNT option 53, 189, 212
DESTID option
  ASSIGN command 53
  batch data interchange 268
DESTIDLENG option
  ASSIGN command 53
  batch data interchange 268
destination
  extrapartition 301
  identification 265
  indirect 301
  intrapartition 301
detect an attention condition 228
device control options (BMS) 152, 183
device dependent data stream 135
DFHAID copy book 208, 232
DFHBMSCA copy book 150, 205, 233
DFHCOMMAREA (communication area) 21
DFHEAI interface processor 10
DFHECALL macro 10
DFHEIBLK copy book 10
DFHEICAL macro – use DFHECALL 10
DFHEIEND macro 10
DFHEIENT macro 10
  CODEREG 12
  DATAREG 12
  defaults 12
  EIBREG 12
DFHEIPLR symbolic register 12
DFHEIRET macro 10
DFHEISTG macro 10
DFHMDF field definition macro 145, 194
DFHMDI map definition macro 144, 193
DFHMSD map set definition macro 143, 193
DFHMSRCA copy book 170, 208
DFHPDI partition definition macro 166, 204
DFHPSD partition set definition macro 204
DFHRESP builtin function 43
DFH2980 structure 243
DIB (DL/I interface block) 105
direct terminal 186
DISABLED condition 98
disconnect a switched line 223
display device operations 230
  attention identifier list (DFHAID) 232
  attention identifier (AID) 232
  copy displayed information 232
  cursor address 232
  erase all unprotected fields 232
  input operation without data 232
  pass control on receipt of an AID 232
  print displayed information 231
  standard attribute/printer control character list (DFHBMSCA) 205, 233
display partitioning 162
display register (EDF) 61
display trigger field 171
disposition and message routing 186
distributed transaction processing (DTP) 221
DLI option 14
DL/I
  access scheduling 117
  and EDF 64
  call check 120
  CALL statement 117
  database access 117
  database call 119
  EXEC DLI command 101
  interface block (DIB) 105
  response codes 120
  restriction in IMS batch 129
  sync points 331
  work area 119
DSATTS operand 198
DSIDERR condition 98
DSSTAT condition 269
DTP (distributed transaction processing) 221
dual screen mode (EDF) 58
DUMP command 323
dump control 323
DUMPCODE option 324
DUPKEY condition 99
DUPREC condition 99
dynamic call 28
dynamic program 18
dynamic storage 10

E

ECADDR option 281
EDF option 14, 74
EDF (see execution diagnostic facility)
EI option 321
EIB (see EXEC interface block)
EIBAID field 155, 339
  examining contents 208
EIBATT field 339
EIBCALEN field 339
EIBCOMPL field 339
EIBCONF field 339
EIBCPOSN field 339
EIBDATE field 275, 339
EIBDS field 339
EIBEOC field 339
EIBERR field 339
EIBERRCD field 339
EIBFMH field 340
EIBFN field 44
EIBFREE field 341
EIBNODAT field 341
EIBRCODE field 44, 341
EIBRECV field 343
EIBREG argument 12
EIBREQID field 344
EIBRESP field 344
EIBRESP2 field 345
EIBRLDBK field 345
EIBRSRCE field 345
EIBSIG field 345
EIBSYNC field 345
EIBSYNRB field 345
EIBTASKN field 345
EIBTIME field 275, 345
EIBTRMID field 345
EIBTRNID field 345
end browse operation 96
end of message (EOM) order 161
END-EXEC delimiter (COBOL) 5
end-of-command delimiter 5
ENDBR command 96
ENDDATA condition 283
ENDFILE condition 99
ENDINPT condition 262
ENQ command 285
ENQBUSY condition 286
enqueue upon resource 285
ENTER command 320
ENTER key 71, 155
ENTER option 155, 233
ENTRY option 296
entry point, trace 317
entry to assembler program 10
entry-sequenced data set (ESDS) 77
ENTRYNAME option 321
ENVDEFERR condition 283
EOC condition
  basic mapping support 218
  terminal control 262
EODS condition
  basic mapping support 218
  batch data interchange 270
  terminal control 262
EOF condition 262
EOM (end of message) order 161
EPILOG option 14
EQUAL option 97
equated symbols 6
erase all unprotected fields 232
ERASE option 152, 213
  terminal control 258
ERASEAUP option 152, 214
ERROR condition 44
error handling 17
ERRTERM option 214
ESDS (entry-sequenced data set) 77
establish a sync point 331
event
  control area, timer 276
  monitoring point 317
  waiting for 277
exceptional conditions
  abnormal termination recovery 315
  basic mapping support 156, 217
  batch data interchange 269
  description 43
  file control 98
  HANDLE CONDITION command 45
  IGNORE CONDITION command 45
  interval control 283
  journal control 330
  list of 46
  partitions 169
  program control 297
  storage control 300
  task control 286
  temporary storage control 308
  terminal control 262
  trace control 321
  transient data control 303
exclusive control
  BDAM 91
  releasing (UNLOCK) 95
  VSAM 87
EXEC CICS command format 5
EXEC DLI command 64, 101
EXEC interface block (EIB)
  description 51

EXEC interface block (EIB) *(continued)*
  fields 339
EXEC interface stubs 19
execution diagnostic facility (EDF)
  CEBR initiation 73
  CEDF transaction 58
  displays 59, 63
  dual screen mode 58
  EXEC DLI command 64, 101
  functions 57
  installing 58
  invoking 58
  program labels 64
  pseudoconversational program 64
  security rules 58
  single screen mode 58
  terminal sharing 60
exit from ASM program 10
exit (see abnormal termination recovery)
expanded area (CL interpreter) 71
expiration time
  notification when reached 276
  specifying 275
EXPIRED condition 283
EXTATT operand 198
EXTDS option 53
extended attribute 160
extended color 140
EXTRACT ATTACH command 236
EXTRACT LOGONMSG command 229
EXTRACT PROCESS command 238
EXTRACT TCT command 236
extrapartition destination 301

F

FACILITY option 53
facility, alternate 221
facility, principal 221
FCI option 54
FCT option 324
FE option 14
field concepts, 3270 140
field data format (BMS) 149
field definition macro (BMS) 145, 194
field edit built-in function 335
field group 145
field of EIB 51
FIELD option 335
field outlining 141
field separator operand 198
fields, EIB 339
file control
  an overview 77
  BDAM data sets 89
  data set identification 79

file control *(continued)*
  deleting VSAM records 94
  end browse operation 96
  exceptional conditions 98
  options 97
  read a record 93
  read next record 95
  read previous record 96
  release exclusive control 95
  reset start for browse 96
  specify start for browse 95
  update a record 94
  VSAM data sets 87
  writing new record (WRITE) 93
FILE option 97
FILENOTFOUND condition 99
FIND command (CEBR) 73
flag byte, route list 188
FLAG option 14
FLDSEP operand 198
FLENGTH option 222, 296, 300, 324
floating maps 176
FMH option 258, 281
FMH (see function management header)
FMHPARM option 214
form feed control (BMS) 157
format
  command 5
  data 135
  trace table 318
FORMATTIME option 276
FORMFEED option 152, 161, 214
FREE command 229, 236
free main storage 299
FREEKB option 152, 214
FREEMAIN command 299
FREEMAIN restriction 18
FROM option
  basic mapping support 214
  batch data interchange 268
  dump control 324
  file control 97
  interval control 281
  journal control 329
  temporary storage control 307
  terminal control 258
  trace control 321
  transient data control 303
FROMFLENGTH option 222
FROMLENGTH option 258
FRSET option 152, 214
full function BMS 173
full function logical unit (3790) 254
fullword length option 222

FUNCERR condition 270
function management header (FMH) 227
function that is unsupported 44

G

GCHARS option 54
GCODES option 54
GDDM coordination (BMS)
  GDDM GSFLD call 153
  GDDM PSRSRV call 145
  graphic hole 153
  restriction with partitions 169
GDS option 15
General Banking Terminal System (see 2980)
generic key 80
GENERIC option 97
GET command (CEBR) 73
get main storage 299
GETMAIN command 299
global variables, VS COBOL II (ANSI85) 40
graphic hole 153
green parameter of COLOR operand 196
GRPNAME operand 145, 198
GTEQ option 97

H

HANDLE ABEND command 313
HANDLE AID command 154, 172, 232
HANDLE CONDITION command 45
hardware print key 156
HEADER operand 177
  map definition macros 198
HEADER option 181, 214
hhmmss argument 5
highlighting 141
HILIGHT operand 198
HILIGHT option 54
HOLD option 296
HONEOM option 214
horizontal picture element 165
host command processor LU (3650/3680) 249
host conversational (3270) LU (3650) 249
host conversational (3653) LU (3650)
hpel (horizontal picture element) 165
HTAB operand 198

I

IC attribute 153
identification
  BDAM record 89
  data set 79
  destination 265
  VSAM record 87
IGNORE CONDITION command 45
IGREQCD condition
  basic mapping support 218
  batch data interchange 270
  terminal control 262
IGREQID condition 218
ILLOGIC condition 99
immediate light pen field 155
INBFMH condition 262
inbound FMH 227
index, alternate (AIX) 78
indicator lights (MSR) 170
indirect destination 301
INITIAL operand 198
initialize main storage 299
initiate a task (see start a task)
INITIMG option 300
INPARTN option 54, 212, 214
input data
  chaining of 226
  unsolicited 227
input data set 9
input operation without data 232
input operations 139
input partition 164, 167
INQUIRE command 51
inquiry logical unit (3790) 254
insert-cursor indicator 141
installing EDF 58
installing the CL interpreter 72
integrity of data 78
interactive logical units 252
interface processor DFHEAI 10
interface stubs, EXEC 19
interleaving conversation with message routing 186
interpreter
  installation 72
  invoking 67
  screen layout 67
  security rules 72
  variables 70
interpreter logical unit (3650) 250
interrogate a data set 266
interval control
  cancel interval control command 280
  delay processing of task 276
  exceptional conditions 283
  expiration time 275
  format of date and time 276
  notification when specified time expires 276
  options 280
  request current time of day 275
  retrieve data stored for task 279
  specifying request identifier 275
  start a task 277

interval control *(continued)*
- wait for event to occur 277

INTERVAL option
- basic mapping support 214
- interval control 281

INTO option
- basic mapping support 214
- batch data interchange 269
- file control 97
- interval control 281
- temporary storage control 307
- terminal control 258
- transient data control 303

intrapartition destination 301
INVERRTERM condition 218
INVITE option 226, 258
INVLDC condition 218
INVMPSZ condition 218
invoking EDF 58
invoking the CL interpreter
- CECI transaction 67
- CECS transaction 67

INVPARTN condition 218
INVPARTNSET operand 218
INVREQ condition
- basic mapping support 218
- file control 99
- interval control 283
- journal control 330
- program control 297
- temporary storage control 308
- terminal control 262
- trace control 321

INVTSREQ condition 283
IOERR condition
- file control 99
- interval control 283
- journal control 330
- temporary storage control 308
- transient data control 303

ISCINVREQ condition
- file control 99
- temporary storage control 308
- transient data 304

ISSUE ABEND command 238
ISSUE ABORT command 267
ISSUE ADD command 266
ISSUE CONFIRMATION command 239
ISSUE COPY command 232
ISSUE DISCONNECT command 223, 228
ISSUE END command 267
ISSUE ENDFILE command 252
ISSUE ENDOUTPUT command 252
ISSUE EODS command 250
ISSUE ERASE command 267
ISSUE ERASEAUP command 232
ISSUE ERROR command 239
ISSUE LOAD command 250
ISSUE NOTE command 268
ISSUE PASS command 228
- effects of NOTIFY 228

ISSUE PREPARE command 228
ISSUE PRINT command 156, 231
ISSUE QUERY command 266
ISSUE RECEIVE command 266
ISSUE REPLACE command 266
ISSUE RESET command 223
ISSUE SEND command 268
ISSUE SIGNAL command 223
ISSUE WAIT command 268
ITEM option 307
ITEMERR condition 308
IUTYPE option 258
I/O work area in DL/I 119

J

JFILEID option 329
JIDERR condition 330
JOURNAL command 328
journal control
- create a journal record 328
- exceptional conditions 330
- journal records 327
- options 329
- output synchronization 327

journal records
JTYPEID option 329
JUSFIRST option 214
JUSLAST option 215
JUSTIFY operand
- effect on map positioning 177
- map definition macros 199

JUSTIFY option 215

K

KATAKANA option 54
key
- alternate (secondary) 78
- generic 80
- physical 89

key-sequenced data set (KSDS) 77
KEYLENGTH option
- batch data interchange 269
- file control 97
- remote data set 85

KEYNUMBER option 269
keystroke overlapping 163

keyword fields on screen 142
keyword length 222
KSDS (key-sequenced data set) 77

L

label argument 5
LABEL option 315
LANG operand 199
LANGLVL option 15
LAST option 215
  bracket protocol 227
  terminal control 259
LDC operand 200
LDC option
  basic mapping support 215
  description of 248
  terminal control 259
LDCMNEM option 54, 212
LDCNUM option 54, 212
LEAVEKB option 259
LENGERR condition 218, 286
  batch data interchange 270
  file control 99
  interval control 283
  journal control 330
  program control 297
  storage control 300
  temporary storage control 308
  terminal control 262
  transient data control 304
length of 19
length of passed area 18
LENGTH operand 200
LENGTH option
  basic mapping support 215
  batch data interchange 269
  built-in function 335
  default (assembler language) 6
  default (PL/I) 7
  dump control 324
  file control 97
  interval control 281
  journal control 329
  program control 296
  storage control 300
  task control 286
  temporary storage control 307
  terminal control 259
  transient data control 303
LENGTH register, VS COBOL II 24
levels, application program logical 289
light pen
  detectable field 141
  handling in program 155
light pen detectable field 155
LIGHTPEN option 155, 233
LINE command (CEBR) 73
LINE operand 177, 200
line width for printer 157
LINEADDR option 259
LINECOUNT option 15
line, communication 223
LINK command 289
  in VS COBOL II 32—36
link to program anticipating return 289
LIST option
  basic mapping support 215
listing data set 9
literal constant 6
load a map set 146
load a program, table, or map 291
LOAD command 291
load module size 19
local copy key 156
locality of reference 17
locate-mode
  SERVICE RELOAD statement 23
  to minimize the working set 18
lockout (see deadlock prevention)
logical device code (LDC option) 248
logical device components
  basic support 169
  page overflow 180
logical levels, application program 32, 35, 289
logical messages (BMS)
  completing a logical message 174
  cumulative text processing 181
  device controls 183
  direct terminal output 174
  example of use 181
  floating maps 177
  introduction 173
  map positioning 177
  message recovery 174
  page overflow 180
  PAGING output 175
  purging a logical message 174
logical record presentation 226
logical unit of work (LUW) 331
logical units
  batch 253
  conversing with (CONVERSE) 223
  facilities for 225
  interactive 252
  LUTYPE6.1 235
  LUTYPE6.2 237
  pipeline 247
  reading data from 266
    terminal control 222

logical units *(continued)*
  writing data to 266
    terminal control 223
  3270 Information Display System 245
  3270 SCS Printer 245
  3270-Display (LUTYPE2) 246
  3600 Pipeline 247
  3600 (3601) 248
  3600 (3614) 248
  3650 Host Conversational (3270) 249
  3650 Host Conversational (3653) 250
  3650 Interpreter 250
  3650 Pipeline 247
  3650/3680 Host Command Processor 249
  3767 Interactive 252
  3770 Batch 253
  3770 Interactive 252
  3790 Full Function 254
  3790 Inquiry 254
  3790 SCS printer 255
  3790 (3270-Display) 255
  3790 (3270-Printer) 256
look aside query 163
lowercase characters, VS COBOL II (ANSI85) 36
LUNAME option 259
LUTYPE2 (3270-Display LU) 246
LUTYPE4
  batch data interchange 265
  logical record presentation 226
  logical unit 235
LUTYPE6.1 Logical Unit 235
LUTYPE6.2 Logical Unit 237
LUW (logical unit of work) 331
L40 option 215
L64 option 215
L80 option 215

M

macro global bit (&DFHEIMX) 13
macro instruction
  command equivalent 351
  field definition, DFHMDF 145
  map definition, DFHMDI 144
  map set definition, DFHMSD 143
  partition definition (DFHPDI) 166
  partition set definition (DFHPSD) 165
  used with commands 13
macros
  used with commands 18
magnetic slot reader (MSR)
  audible alarm 170
  DFHMSRCA copy book 170
  DFHMSRCA set of constants 170
  indicator lights 170
  MSRCONTROL option 170
MAIN option 307
main storage
  dumping (DUMP) 323
  initialize 299
  obtain 299
  releasing (FREEMAIN) 299
  temporary data 305
map definition macro (BMS) 193
map definition macro, BMS 144
MAP option 215
map positioning 177
map set definition macro 143
map set definition macro (BMS) 193
map set loading 146
map set name 146
map set suffixing 146
map set suffixing and partitions 164
map size 19
MAPATTS operand 200
MAPCOLUMN option 54, 212
MAPFAIL condition 156, 219
MAPHEIGHT option 54, 212
MAPLINE option 54, 212
MAPONLY option 151, 215
mapping input data (RECEIVE MAP) 153
maps
  ASM sample programs 378
  assembling 146
  cataloging 146
  COBOL sample programs 417
  copying symbolic description 148
  physical 135
  PL/I sample programs 457
  symbolic 135
  temporary modification 149
MAPSET option 215
MAPSFX operand 205
MAPWIDTH option 54, 212
MARGINS option 15
MASSINSERT option 98
MAXFLENGTH option 222
MAXLENGTH option 259
MDT (modified data tag) 141
message
  teletypewriter 230
  title 186
message area of screen 142
message length, teletypewriter 230
message recovery (BMS) 174
message routing 184
message switching transaction, CMSG 189
minimum function BMS 139
mixed addressing mode transaction 293
mixed mode application program 13, 18

MMDDYY option
MODE operand 200
modified data tag (MDT) 141
modular program 18
MONITOR option 321
monitoring point (ENTER command) 317
MONTHOFYEAR option 281
move-mode 18
MSR option 216
MSR (magnetic slot reader) 170
MSRCONTROL option 54, 170
multiple base registers 12
multithreading 17
MVS/XA restrictions 19
  VS COBOL II programs 30
MVS/XA transaction 293, 299

N

name argument 5
naming restriction 13
nested programs, VS COBOL II (ANSI85) 38
NETNAME option 54, 259
neutral parameter of COLOR operand 196
NEXT option 307
NLEOM option 216
NOAUTOPAGE option 174, 216
NOCHECK option 281
NODEBUG option 14
NOEDF option 14
NOEPILOG option 14
NOHANDLE option 43
NOJBUFSP condition 330
nondisplay field 141
NONUM option 15
NONVAL condition 262
NOOPSEQUENCE option 15
NOOPT option 15
NOOPTIONS option 15
NOPASSBKRD condition 262
NOPASSBKWR condition 262
NOPROLOG option 15
NOQUEUE option 259
normal intensity field 140
NOSEQ option 15
NOSEQUENCE option 15
NOSOURCE option 16
NOSPACE condition
  file control 100
  temporary storage control 308
  transient data control 304
NOSPIE option 16
NOSTART condition occurs if the 3651 is unable to initiate 263
NOSTG condition 300
NOSUSPEND option 44, 259
  ENQ command 286
  GETMAIN command 300
  JOURNAL command 329
  READQ TD command 303
  WRITEQ TS command 307
NOTALLOC condition 263
NOTAUTH condition 44
  file control 100
  interval control 283
  journal control 330
  program control 297
  temporary storage control 308
  transient data control 304
NOTFND condition
  file control 100
  interval control 283
NOTOPEN condition
  file control 100
  journal control 330
  transient data control 304
NOTRUNC compiler option 24
NOTRUNCATE option 259
NOVBREF option 16
NOWAIT option 269
NOXREF option 16
null lines and 3270 printer 157
null map 144
NUM option 15
numeric-only field (3270 attribute character) 140
NUMITEMS option 307
NUMREC option
  batch data interchange 269
  file control 98
NUMTAB option 54

O

OBFMT operand 200
OCCURS operand 201
OPCLASS option
  ASSIGN command 55
  basic mapping support 216
operator class codes 187
operator identification card reader 155
OPERID option 155, 233
OPERKEYS option 55
OPERPURGE option 174, 216
OPID option 55
OPMARGINS option 15
OPSECURITY option 55
OPSEQUENCE option 15
OPT option 15
option length 222
options
  abnormal termination recovery 315

options *(continued)*
- ADDRESS command 51
- ASSIGN command 53
- basic mapping support 213
- batch data interchange 268
- dump control 323
- execution time (PL/I STAE) 42
- file control 97
- HANDLE AID command 233
- HANDLE CONDITION command 45
- interval control 280
- journal control 329
- program control 296
- STAE (PL/I) 42
- storage control 300
- task control 286
- temporary storage control 307
- terminal control 257
- trace control 321
- transient data control 303
- translator 13

OPTIONS option 15
OPTIONS(MAIN) specification 13, 42
OS/VS COBOL
- mixing with VS COBOL II 32

outboard formatting
- basic mapping support 172
- effect of partitions 169

outbound FMH 227
OUTLINE operand 200
OUTLINE option 55
output control (2980) General Banking Terminal System 243
output data set 9
output data, chaining of 226
output operations 139
output partition 164
output to common buffer (2980) 243
OVERFLOW condition 180, 219
overlapping keystroking 163
overlays 18
overtyping EDF displays 63

P

PA option 155, 233
PA (program access) key 155
page buffer (3270) 160
page fault 17
page overflow example 181
page overflow (BMS) 180
page width for printer 157
PAGENUM option 55, 213
PAGING option 175, 216
paging (see terminal operator paging) 175
partition definition macro (BMS) 166, 204
PARTITION JUMP key 162
partition set 163, 164
partition set definition macro (BMS) 165, 204
partition set suffixing 146
partitions
- active partition 162
- base state 165
- character cell 168
- cumulative processing 184
- cursor 162
- description 162
- display control 162
- ENTER key 162
- exceptional conditions 169
- GDDM restriction 169
- outboard format 184
- outboard formatting 169
- page overflow 180
- PARTITION JUMP key 162
- partition set 163
- presentation space 164
- terminal sharing 184
- transaction partitions 163
- unpartitioned state 165
- viewport 164
- window 164
- within a map set 166

PARTN operand 201
PARTN option
- basic mapping support 216

PARTNFAIL condition 219
PARTNPAGE option 55, 213
PARTNS option 55
PARTNSET option 55, 213, 216
PASSBK option 259
passbook control (2980) 243
passing
- session 228

passing control
- anticipating return (LINK) 289
- on receipt of an AID (HANDLE AID) 232
- without return (XCTL) 290

passing data
- to new tasks 279
- to other program 292

PCB address 118
PCB (program communication block) 117
PCT option 324
pel (picture element) 168
PERFORM option 321
PF option 155, 233
PF (program function) key
- BMS 155
- CL interpreter 71

PF (program function) key *(continued)*
  EDF 59
PF (program function) key (CECI/CECS) 71
PFXLENG option 329
PGMIDERR condition
  abnormal termination recovery 315
  program control 297
physical key 89
physical map 135
PICIN operand 201
PICOUT operand 201
picture element (pel) 168
pink parameter of COLOR operand 196
pipeline logical unit 247
PIPLENGTH option 260
PIPLIST option 260
PL/I
  argument value 7
  compilers supported 42
  DL/I CALL interface 41
  LENGTH option default 7
  OPTIONS(MAIN) specification 42
  PROCEDURE statement 13
  program segments 42
  restrictions 41
  sample programs 433
  STAE option 42, 314
  translated code 13
POINT command 236
pointer-ref argument 5
pointer-value argument 5
POP HANDLE command 46
POS operand 146, 202
POST command 276
posting timer event control area 276
PPT option 324
PREFIX option 329
pregenerated modules 137
presentation space 164
principal facility 221
PRINSYSID option 55
print displayed information 231
PRINT option 152
  basic mapping support 216
  batch data interchange 269
printer and BMS text 161
printer control character list (DFHBMSCA) 205, 233
printers
  basic mapping support (BMS) 160
  device independence 161
  FORMFEED option 161
  NLEOM option 161
  printing displayed data 156
  SCS and non-3270 printers 161
  starting a printer task 156
printers *(continued)*
  3270 printer page width 156
  3270 printer using NLEOM 161
  3270 printer without NLEOM 160
  3270 printers and blank lines 156
printing contents of screen 156
PROCESS option 260
PROCLENGTH option 260
PROCNAME option 260
PROFILE option 260
program access (PA) key 155
program communication block (PCB) 117
program control
  CL interpreter 72
  deleting loaded program 291
  exceptional conditions 297
  linking to another program 289
  load a program, table, or map 291
  options 296
  passing data to other program 292
  program logical levels 289
  returning program control 291
  transfer program control 290
program function key (see PF key)
program labels in EDF 64
PROGRAM option
  abnormal termination recovery 315
  dump control 324
  program control 296
  terminal control 260
program segments
  COBOL 24
  PL/I 42
program size 19
program specification block (PSB) 117
program testing 57
programmed symbols 141
programming techniques
  COBOL 20
  general 17
  PL/I 41
programs
  checking out pseudoconversational 64
programs, sample (ASM) 353
programs, sample (COBOL) 393
programs, sample (PL/I) 433
PROLOG option 15
protected field 140
PROTECT, option
  START 281
PS operand 202
PS option 55
PSB release 119
PSB scheduling 118

PSB (program specification block) 117
PSEUDOBIN option 260
pseudoconversational programming 64
PURGE command (CEBR) 73
PURGE MESSAGE command 174
PUSH HANDLE command 46
PUT command (CEBR) 74

Q

QBUSY condition 304
QIDERR condition
  temporary storage control 308
  transient data control 304
QNAME option 55
quasi-reenterability 17
query structured field 53
question mark (CL interpreter) 67
QUEUE command (CEBR) 73
QUEUE option 260
  interval control 281
  temporary storage control 307
  transient data control 303
queue, temporary storage 305
QUOTE option 15
QZERO condition 304

R

RBA option 98
RBA (relative byte address) 77
RDATT condition
  basic mapping support 219
  terminal control 263
reactivate an ABEND exit 313
read ahead queueing feature 225
read attention 241
READ command 93
reading
  batch data interchange record 266
  data from a display (RECEIVE MAP) 153
  data from temporary storage queue 306
  data from terminal or LU 222
  data from transient data queue 302
  file control record 93
  next record when browsing 95
  previous record in VSAM browse 96
READNEXT command 95
READPREV command 96
READQ TD command 302
READQ TS command 306
RECEIVE command 222
RECEIVE MAP command 153, 209
RECEIVE PARTN command 168, 209
RECFM option 260
record
  deleting VSAM 94
  identification 87, 89
  journal 327
  reading 93, 266
  requesting next number 268
  updating 94, 266
  writing new (adding) 93, 266
record descriptions
  ASM sample programs 392
  COBOL sample programs 431
  PL/I sample programs 471
recovery
  abnormal termination 313
  and debugging 309
  sequential terminal support 311
  sync point 331
red parameter of COLOR operand 196
reenterability 17
reference modification, VS COBOL II (ANSI85) 40
register contents in ASM 10
relative byte address (RBA) 77
relative record data set (RRDS) 77
relative record number (RRN) 77
RELEASE command 291
RELEASE option 174, 217
releasing
  a PSB 119
  area of main storage 299
  exclusive control (UNLOCK) 95
relinquish communication line 223
RELOAD operand of DFHPPT macro 291
relocatable expression 6
remote data set, KEYLENGTH option 85
REPLACE statement, VS COBOL II (ANSI85) 36
REQID option 174
  basic mapping support 217
  file control 98
  interval control 281
  journal control 329
request/response unit (RU) 226
RESET option 315
reset start for browse 96
RESETBR command 96
RESOURCE option 261, 286, 321
resource scheduling 285
RESP option 43
RESP values 344
response codes (DL/I) 120
RESP2 option 43
RESP2 values 345
RESTART option 55
restrictions
  assembler language 19
  COBOL 20

restrictions *(continued)*
  IMS batch programs 129
  MVS/XA 19
  PL/I 41
  31-bit mode 19
RETAIN option 174, 217
RETPAGE condition 219
RETRIEVE command 279
retrieve data stored for task 279
RETURN CODE register, VS COBOL II 25
RETURN command 291
return facility to CICS 229
return program control 291
returning mapped data to a program 189
REWRITE command 94
REWRITE option 308
RIDFLD option
  batch data interchange 269
  file control 98
ROLLBACK option 331
ROLLEDBACK condition 332
ROUTE command 184
route list (LIST option) 187, 188
routing messages (ROUTE) 186
routing terminal 186
RPROCESS option 261
RRDS (relative record data set) 77
RRESOURCE option 261
RRN option
  batch data interchange 269
  file control 98
RTEFAIL condition 219
RTERMID option 282
RTESOME condition 219
RTRANSID option 282
RU (request/response unit) 226
run unit in VS COBOL II 32, 35

S

SA (set attribute) order 160
sample program
  browse (ASM) 361
  browse (COBOL) 401
  browse (PL/I) 441
  inquiry/update (ASM) 354
  inquiry/update (COBOL) 395
  inquiry/update (PL/I) 435
  low balance report (ASM) 375
  low balance report (COBOL) 414
  low balance report (PL/I) 454
  operator instruction (ASM) 354
  operator instruction (COBOL) 394
  operator instruction (PL/I) 434
  order entry queue print (ASM) 372
  order entry queue print (COBOL) 411

sample program *(continued)*
  order entry queue print (PL/I) 451
  order entry (ASM) 368
  order entry (COBOL) 407
  order entry (PL/I) 447
SBA (set buffer address) order 161
schedule a PSB 118
schedule access (DL/I) 117
schedule use of resource by task 285
Screen Definition Facility/CICS (SDF/CICS) 136
screen layout design
  application data area 142
  data fields 142
  input operations 139
  keyword fields 142
  message area 142
  output operations 139
  requirements 142
  stopper fields 142
  title area 142
screen layout (CL interpreter)
  command input area 67
  information area 68
  status area 68
screen partitioning 162
screen size 142
SCRNHT option 55
SCRNWD option 56
scrolling 162
SCS printer logical unit (3790) 255
SDF/CICS (Screen Definition Facility/CICS) 136
secondary key 78
security rules
  CL interpreter 72
  EDF 58
segment search argument (SSA) 118
segments, program
  COBOL 24
  PL/I 42
SELNERR condition 270
semicolon delimiter (PL/I) 5
send asynchronous interrupt 223
SEND command 223
SEND CONTROL command 153, 211
send data to output device 268
SEND MAP command 150, 210
SEND PAGE command 174
SEND PARTNSET command 167
SEND TEXT command 159, 161, 210
SEND TEXT MAPPED command 190
SEND TEXT NOEDIT command 191
SEND/RECEIVE mode 225
SEND/RECEIVE protocol 226
SEQ option 15

sequence numbers, VS COBOL II (ANSI85) 36
SEQUENCE option 15
sequential retrieval
  See browsing 93
sequential terminal support 311
SERVICE RELOAD elimination, VS COBOL II 25
SERVICE RELOAD statement (COBOL) 23
SESSBUSY condition 263
SESSION option 261
session-oriented information (LU6) 237
SESSIONERR condition 263
set attribute (SA) order 160
set buffer address (SBA) order 161
SET command 51
SET option 189
  basic mapping support 217
  batch data interchange 269
  file control 98
  interval control 282
  program control 296
  storage control 300
  temporary storage control 308
  terminal control 261
  transient data control 303
share option, VSAM 78
SHARED option
  storage control 300
SIGDATA option 56
SIGNAL condition 228
SIGNAL condition occurs when an inbound SIGNAL data-flow 263
SINGLE option 321
single screen mode (EDF) 58
single threading 17
SIT option 324
SIZE operand 202
skip-sequential processing 81
SOSI operand 202
SOSI option 56
SOURCE option 16
SO/SI creation 141
SPACE option 16
spanned records 78
SPIE option 16
SSA (segment search argument) 118
STAE option (PL/I) 42, 314
standard attribute/printer control character list (DFHBMSCA) 205, 233
standard CICS terminal support 234
standard function BMS 159
start a task 278
START command 277
STARTBR command 95
STARTCODE option 56
STARTIO option 329
static call 28
STATIONID option 56
status flag byte, route list 188
status of partition 165
stopper fields on screen 142
storage area length 53
storage control 299
STORAGE operand 202
STORAGE option 324
storage (see main storage)
STRFIELD option 261
stub (see interface processor)
stubs, EXEC interface 19
SUBADDR option 269
subprogram, calling from VS COBOL II 32–36
subroutines 17
SUFFIX operand 146, 203, 205
suffixing, map/partition sets 146
supplied constants (DFHMSRCA) 170
suspend a task 228, 285
SUSPEND command 285
switched line disconnection 223
symbol set 165
symbolic characters defined in program, VS COBOL II (ANSI85) 40
symbolic cursor positioning 153
symbolic description map
  block data format 172
  copying 148
  definition 135
  field data format 149
symbolic register DFHEIPLR 12
sync point 331
synchronization level 238
synchronizing
  journal output 327
  journal output (WAIT JOURNAL) 329
  terminal input/output 223
SYNCLEVEL option 261
SYNCPOINT command 331
syntax notation 3
syntax style 5
SYSBUSY condition 263
SYSID option
  ASSIGN command 56
  file control 98
  interval control 282
  temporary storage control 308
  terminal control 261
  transient data control 303
SYSIDERR condition 304
  EIBRCODE values 343
  file control 100
  interval control 284

SYSIDERR condition *(continued)*
  temporary storage control 308
  terminal control 263
system information, access to 51
SYSTEM option 321
system trace entry point 317
System/3 239
System/370 239
System/7 240

T

tab character 161
TABLES option 324
task control 285
task identification 223
task initiation (see start a task)
TASK option 324
task suspension 228
TCAM-supported logical units 229
TCAM-supported terminals 229
TCT option 325
TCTUA option 51
TCTUALENG option 56
techniques, programming 17
teletypewriter programming 230
TELLERID option 56
temporary storage
  auxiliary 305
  browse transaction (CEBR) 73
  exceptional conditions 308
  main 305
  options 307
  queue 305
TERM operand 203
TERMCODE option 56
TERMERR condition
  terminal control 263
TERMID option
  interval control 282
  terminal control 261
TERMIDERR condition
  interval control 284
  terminal control 263
terminal code table 190
TERMINAL command (CEBR) 73
terminal control
  an overview 221
  bracket protocol (LAST option) 227
  BTAM programmable device 229
  chaining of input data 226
  chaining of output data 226
  converse with terminal or LU 223
  definite response 227
  detecting attention condition (SIGNAL) 228
  disconnect a switched line 223
terminal control *(continued)*
  display device operations 230
  exceptional conditions 262
  facilities for logical units 225
  facilities for terminals 223
  facilities for terminals and LUs 222
  FMH, inbound 227
  FMH, outbound 227
  function management header (FMH) 227
  handle attention identifier 154
  interactive logical units 252
  logical record presentation 226
  LUTYPE2 (3270-Display LU) 246
  map input data (RECEIVE MAP) 153
  options 257
  passing a session 228
  pipeline logical unit 247
  print (ISSUE PRINT) 156
  read attention 241
  reading data from terminal or LU 222
  relinquish communication line 223
  standard CICS terminal support 234
  sync point processing 228
  synchronize terminal I/O 223
  System/3 239
  System/370 239
  System/7 240
  TCAM-supported logical units 229
  TCAM-supported terminals 229
  teletypewriter programming 230
  terminate a session 228
  unsolicited input 227
  VTAM logon data 229
  write break 242
  writing data to terminal or LU 223
  2260 Display Station 241
  2265 Display Station 241
  2741 Communication Terminal 241
  2770 Data Communication System 242
  2780 Data Transmission Terminal 242
  2980 General Banking Terminal System 243
  3270 field concept 140
  3270 Information Display System logical unit 245
  3270 SCS Printer Logical Unit 245
  3270 (BTAM or TCAM supported) 244
  3270-Display LU (LUTYPE2) 246
  3600 Pipeline Logical Unit 247
  3600 (3601) Logical Unit 248
  3600 (3614) Logical Unit 248
  3650 Host Conversational (3270) LU 249
  3650 Host Conversational (3653) LU 250
  3650 Interpreter Logical Unit 250
  3650 Pipeline Logical Unit 247
  3650/3680 Host Command Processor LU 249
  3660 251

terminal control *(continued)*
  3735 251
  3740 252
  3767 Interactive Logical Unit 252
  3770 Batch Logical Unit 253
  3770 Interactive Logical Unit 252
  3790 Full Function Logical Unit 254
  3790 Inquiry Logical Unit 254
  3790 SCS printer logical unit 255
  3790 (3270-Display) Logical Unit 255
  3790 (3270-Printer) Logical Unit 256
  4700 Finance Communication System 256
  7770 Audio Response Unit 256
terminal operator paging
  example of paging message 181
  initiating the paging transaction 174
  message recovery 174
  operator paging commands 176
  PAGING logical messages 175
TERMINAL option
  basic mapping support 217
  dump control 325
terminal sharing
  CL interpreter 72
  EDF 60
terminal-oriented task identification 223
terminating
  map set definition 146
  processing of data set 267
  session 228
  task abnormally (ABEND) 314
testing using sequential devices 311
text data format 135
THEN clause (COBOL) 5
time field of EIB 51
time of day, requesting (ASKTIME) 275
TIME option
  basic mapping support 217
  interval control 282
time-initiated transaction (3735) 251
timer event control area 276
TIMESEP option 282
TIOAPFX operand 203
TIOA, mixed mode programs 18
title area of screen 142
TITLE option 186, 217
title, message 186
TOFLENGTH option 222
TOLENGTH option 261
TOP command (CEBR) 73
TRACE command 320
trace control 317
  auxiliary trace facility 319
  controlling trace facility 320
  exceptional conditions 321
trace control *(continued)*
  options 321
  trace entry format 319
  trace entry point 317
  trace facility control 317
  trace flags 317
  trace table format 318
  user trace entry point 320
trace entry format 319
trace entry point 317
trace facility control 317
trace table format 318
TRACEID option 321
TRAILER operand 177, 203
TRAILER option 181, 217
transaction identifier (CEDF) 57
transaction partition 163
transfer program control 290
TRANSID option
  basic mapping support 217
  interval control 282
  program control 296
TRANSIDERR condition 284
transient data control
  automatic task initiation (ATI) 301
  delete intrapartition queue 303
  exceptional conditions 303
  extrapartition destination 301
  indirect destination 301
  intrapartition destination 301
  options 303
  read data from TD queue 302
  write data to TD queue 302
translated code
  ASM 10
  COBOL 13
  PL/I 13
translation tables for 2980 347
translator data set
  input 9
  listing 9
  output 9
translator options 13
translator options for 37
TRANSP operand 204
trigger field 171
TRIGGER option 233
TSIOERR condition 219
turquoise parameter of COLOR operand 196
TWA option 51
TWALENG option 56
TYPE operand 204

U

UIB (user interface block) 117
UNATTEND option 56
UNEXPIN condition
  basic mapping support 219
  batch data interchange 270
unit of compilation, VS COBOL II 36
  start of 38
  submitting to translator 38
UNLOCK command 95
unpartitioned state 165
unprotected field (3270 attribute character) 140
unsolicited input 227
unsupported function 44
update a record
  batch data interchange 266
  file control 94
update backout (DL/I) 103
update lock 87
UPDATE option 98
upgrade set 78
user interface block (UIB) 117
USER option 321
user trace entry point 317, 320
USERID option
  ASSIGN command 56

V

validation 141
VALIDATION option 56
validity of reference 17
VALIDN operand 171, 204
values of arguments 5
variable (CL interpreter) 70
VBREF option 16
vertical forms control 157
vertical picture element 165
viewport 164
VIEWPOS operand 205
VIEWSZE operand 205
virtual storage environment 17
VOLUME option 269
VOLUMELENG option 269
vpel (vertical picture element) 165
VS COBOL II 24—40
  addressing CICS data areas 25
  ANSI85 standards 36—40
    batch compilation 37
    blank lines 36
    comma and semicolon as delimiters 40
    global variables 40
    lowercase characters 36
    nested programs 38
    programming restrictions 40
    reference modification 40
VS COBOL II *(continued)*
  ANSI85 standards *(continued)*
    REPLACE statement 36
    sequence numbers 36
    symbolic characters defined in program 40
  BMS data structures 29
  CALL statement 28, 32—36
  calling subprograms 32—36
  compiler options not used under CICS 29
  debugging 28
  DL/I CALL interface 30
  LENGTH special register in CICS commands 26
  mixing with OS/VS COBOL 32
  restrictions in a CICS environment 29
  RETURN-CODE special register 27
  returning control 28
  run unit 32
VSAM
  data sets 87
VSAM share option 78
VTAB operand 204
VTAM logon data, access to 229

W

WAIT CONVID command 239
WAIT EVENT command 277
WAIT JOURNAL command 329
WAIT option
  basic mapping support 217
  interval control 283
  journal control 329
  of SEND command 223
  terminal control 223, 262
WAIT SIGNAL command 228
WAIT TERMINAL command 223
waiting
  batch data interchange 268
  for event to occur 277
  terminal control operation 223
WHERE clause in EXEC DLI command
  assembler language program 104
  Boolean expressions 104
  literal string 104
window 164
working set 17
WPMEDIA option 269
WRBRK condition
  basic mapping support 219
  terminal control 263
write break 242
WRITE command 93
WRITEQ TD command 302
WRITEQ TS command 306
writing
  batch data interchange record 266

writing *(continued)*
  data to temporary storage queue 306
  data to terminal or logical unit 223
  data to transient data queue 302
  file control record 93

X

XCTL command 290
XINIT operand 198
XOPTS keyword 14
XREF option 16
XRF and the APPLID option 53
XRF environment 53

Y

YEAR option 283
yellow parameter of COLOR operand 196
YYDDD option 283
YYDDMM option 283
YYMMDD option 283

Z

zero length field 142
zero parameter 199

Numerics

16-megabyte line 299
2260 Display Station 241
2265 Display Station 241
2721 Portable Audio Terminal 256
2741 Communication Terminal 241
2770 Data Communication System 242
2780 Data Transmission Terminal 242
2980 General Banking Terminal System 243
  DFH2980 structure 243
  output control 243
  output to common buffer 243
  passbook control 243
  translation tables 347
31-bit mode transaction 293
3270 Information Display System
  attribute character 140
  field concepts 140
  input operations 139
  logical unit 245
  screen sizes 142
  (BTAM or TCAM supported) 244
3600 Finance Communication System 246
  pipeline logical unit 247
  3601 Logical Unit 248
  3614 Logical Unit 248
3630 Plant Communication System 249
3650 Store System
  Host Conversational (3270) LU 249
  host conversational (3653) LU 250
  interpreter logical unit 250
  pipeline logical unit 251
3650/3680 Store System
  full function logical unit 251
  host command processor LU 249
3660 Supermarket Scanning System 251
3680 Programmable Store System
  host command processor LU 249
3735 Programmable Buffered Terminal 251
3740 Data Entry System 252
3767 Communication Terminal
  interactive logical unit 252
3770 Communication System
  batch logical unit 253
  full function logical unit 253
  interactive logical unit 253
3780 Communications Terminal 253
3790 Communication System
  full function logical unit 254
  inquiry logical unit 254
  SCS printer logical unit 255
  3270-Display Logical Unit 255
  3270-Printer Logical Unit 256
4700 Finance Communication System 256
7770 Audio Response Unit 256
8775 Display Terminal
  field validation attribute character 140
  partition support 162
  trigger validation attribute 171

Special Characters

& (CL interpreter) 70
&DFHEIMX (macro global bit) 13
*ASM statement (assembler language) 13
*PROCESS statement (PL/I) 13
? (CL interpreter) 67

Reader's Comments

CICS/MVS
Application Programmer's Reference
Version 2 Release 1 Modification 2

Publication No. SC33-0512-01

Use this form to tell us what you think about this manual. If you have found errors in it, or you want to express your opinion about it (such as organization, subject matter, appearance) or make suggestions for improvement, this is the form to use. To help us produce books that meet your needs, we have included a questionnaire at the front of the book. Whichever form you use, your comments will be sent to the author's department for review and appropriate action.

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.

Thank you for your time and effort. No postage stamp is necessary if mailed in USA. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.) Be sure to print your name and address below if you would like a reply.

Name

Address

Company or Organization

Phone No.

Reader's Comments
SC33-0512-01

Cut or
Along

Fold and Tape

Please do not staple

Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department 6R1H
180 KOST ROAD
MECHANICSBURG PA 17055-0786

Fold and Tape

Please do not staple

Fold and Tape

Cut or
Along l

IBM®

Application Programmer's Reference
SC33-0512-01

Version 2.1.2

CICS MVS

Program Number
5665-403

Printed in U.S.A.

SC33-0512-01