



CICS
CICS

CICS
CICS

CICS
CICS

Customer
Information
Control
System
CICS/DOS/VS

Licensed Program
Version 1.7

Program Number
5746-XX3

Application
Programmer's
Reference
Manual
(Command Level)

SC33-0077-5

Sixth Edition (July 1987)

This edition applies to Version 1 Release 7 (Version 1.7) of the IBM licensed program Customer Information Control System/ Disk Operating System/Virtual Storage (CICS/DOS/VS), program number 5746-XX3, and to all subsequent releases and modifications until otherwise indicated in new editions or technical newsletters.

This edition is based on the CICS/VS Version 1 Release 6 edition (SC33-0077-4), and changes from that edition are indicated by vertical lines to the left of the changes. The 1.6 edition remains applicable and current for users of Version 1.6.

Changes are made periodically to the information herein; before using this publication in connection with the operation of IBM systems, consult the latest *IBM System/370, 30xx, and 4300 Processors Bibliography*, GC20-0001, for the editions that are applicable and current.

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 in this publication is not intended to state or imply that only IBM's licensed program may be used. Any functionally equivalent program may be used instead.

Publications are not stocked at the addresses given below. Requests for IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed either 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.

IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

This publication contains sample programs. Permission is hereby granted to copy and store the sample programs into a data processing machine and to use the stored copies for study and instruction only. No permission is granted to use the sample programs for any other purpose.

No other part of this publication may be reproduced in any form or by any means, including storing in a data processing machine, without permission in writing from IBM.

THE PUBLICATION OF THE INFORMATION CONTAINED HEREIN IS NOT INTENDED TO AND DOES NOT CONVEY ANY RIGHTS OR LICENSES, EXPRESS OR IMPLIED, UNDER ANY IBM PATENTS, COPYRIGHTS, TRADEMARKS, MASK WORKS OR ANY OTHER INTELLECTUAL PROPERTY RIGHTS.

Preface

What This Book Is About

This book describes the IBM Customer Information Control System/Disk Operating System/Virtual Storage (CICS/DOS/VS) 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/DOS/VS (5746-XX3).

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

Who This Book Is For

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/DOS/VS Facilities and Planning Guide* is required. This facilities and planning 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 alphabetical 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 ACF/VTAME
- BTAM refers to BTAM-ES.
- ASM is used sometimes as the abbreviation for assembler language.

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 Data Bases” on page 73

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 127

Deals with communication with terminals, logical units, and subsystems in the telecommunications network to which the CICS system is connected.

A chapter is included about the report controller (a separately orderable feature) and the commands and options that can be selected to control the way it operates.

Another chapter is included that describes application programming using the Structured Query Language/Data System (SQL/DS).

“Part 4. Control Operations” on page 335

Describes the facilities for controlling the operation of application programs in the CICS system.

“Part 5. Recovery and Debugging” on page 379

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 407

Describes the one built-in function (BIF DEDIT) available with the command-level interface.

“Appendices” on page 411

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).

G. Report Controller Sample Programs

“Index” on page 557

Bibliography

CICS/DOS/VS Version 1 Release 7 Library

General

General Information GC33-0155	Release Guide GC33-0130	Messages and Codes SC33-0081
Library Guide GC33-0356	Master Index SC33-0095	

Planning

Facilities and Planning Guide SC33-0228	Installation and Operations Guide SC33-0070	Application Programming Primer SC33-0139	Problem Determination Guide SC33-0089
Intercommunication Facilities Guide SC33-0133	Resource Definition (Online) SC33-0238	Application Programmer's Reference Manual (Command Level) SC33-0077	Program Debugging Reference Summary SX33-6010
Recovery and Restart Guide SC33-0135	Resource Definition (Macro) SC33-0149	Application Programmer's Reference Summary (Command Level) GX33-6012	Data Areas LY33-6033
Performance Guide SC33-0134	Customization Guide SC33-0131	Application Programmer's Reference Manual (RPG II) SC33-0085	Diagnosis Reference LC33-0105
Performance Data SC33-0219	CICS-Supplied Transactions SC33-0080	Application Programmer's Reference Manual (Macro Level) SC33-0079	Remote Server Diagnosis LC33-0438
3270 Data Stream Device Guide SC33-0096			
3650/3680 Guide SC33-0073			
3767/3770/6670 Guide SC33-0074			
3790/3730/8100 Guide SC33-0075			
4700/3600/3630 Guide SC33-0072			

End User

Report Controller User's Guide SC33-0382

Books from Related Libraries

Systems Network Architecture

See the following Systems Network Architecture (SNA) manual for further information on SNA:

Sessions between Logical Units, GC20-1868

Structured Query Language/Data System (SQL/DS)

- | For information on executing SQL/DS in a CICS application program see the following manuals:
 - | *SQL/DS Application Programming*, SH20-5019
 - | *IBM DATABASE2 Application Programming Guide for CICS Users*, SC26-4080
 - | *IBM DATABASE2 Reference*, SC26-4078.

Other Products

You may also want to refer to the following IBM publications:

Distributed Processing Programming Executive/Distributed Presentation Services (DPPX/DPS):

DPPX/Distributed Presentation Services Version 2 System Programming Guide, SC33-0117

Screen Definition Facility/CICS SDF/CICS Program Reference Manual, SH19-6077

IBM System/360 Disk Operating System:

Subset American National Standard COBOL Compiler and Library Programmer's Guide, SC28-6439

Full American National Standard COBOL Compiler and Library Version 3, Programmer's Guide, SC28-6441

Full American National Standard COBOL Programmer's Guide, GC28-6398

DOS/VS COBOL Compiler and Library Programmer's Guide, SC28-6478

DOS PL/I Optimizing Compiler Programmer's Guide, SC33-0008

DL/I DOS/VS Application Programming: High Level Programming Interfaces, SH24-5009

DL/I DOS/VS Interactive Resource Definition and Utilities, SH24-5029

IBM 3274 Control Unit Reference Summary, GX20-1878

Contents

Part 1. Command Level Programming	1
Chapter 1.1. Introduction to Command Level Programming	
Programs	3
Commands Instead of Macros	3
Syntax Notation Used in This Manual	4
Chapter 1.2. Command Format and Argument Values	
Values	5
Command Format	5
Coding Conventions	5
Argument Values	6
Argument Values in Assembler Language	6
Argument Values in COBOL	7
Argument Values in PL/I	8
Chapter 1.3. Command Language Translator	
Translator Data Sets	11
Input Data Set	11
Output Data Set	11
Listing Data Set	12
Translated Code	12
Assembler Language	12
COBOL	15
PL/I	15
Translator Options	16
Chapter 1.4. Programming Techniques and Restrictions	
General Programming Techniques	21
CICS Macros Used With CICS Commands	23
Program Size	23
Entry Point Address	23
BMS Map Size	24
Assembler-Language Considerations	24
Restrictions	24
Commands Contained Within Macros and COPY Code	24
Invoking Assembler-Language Application Programs by a Call Statement	24
COBOL Considerations	24
Restrictions	24
Program Segments	28
PL/I Considerations	29
Restrictions	29
PL/I STAE Execution-Time Option	29
Compilers Supported	29
OPTIONS(MAIN) Specification	30
Program Segments	30
Base Locator for Linkage	25
NOTRUNC Compiler Option	28
Terminal Sharing Between Transaction and EDF	54
Functions of EDF	49
Security Rules	51
Installing EDF	51
Invoking EDF	51
EDF Displays	52
Diagnostic Facility	49
Terminal Sharing Between Transaction and EDF	54
Ignore Exceptional Conditions (IGNORE CONDITION)	35
Suspend Condition Handling (PUSH and POP)	35
List of Exceptional Conditions	35
Access to CICS Storage Areas (ADDRESS)	41
ADDRESS Command Options	42
Values Outside the Application Program (ASSIGN)	42
Example of ADDRESS and ASSIGN Commands	43
Assign Command Options	43
Functions of EDF	49
Security Rules	51
Installing EDF	51
Invoking EDF	51
EDF Displays	52
Diagnostic Facility	49
Terminal Sharing Between Transaction and EDF	54

Program Function (PF) Keys	54	Record Identification	87
Overtyping EDF Displays	57	CICS Locking of VSAM Records in Recoverable Files	88
Checking Pseudoconversational Programs	58		
Program Labels	58		
EDF and EXEC DLI Commands	59		
Chapter 1.8. Command Level Interpreter	61	Chapter 2.3. File Control - DAM Considerations	89
Invoking the Command Level Interpreter	61	Record Identification	89
Screen Layout	62	Browsing Records From DAM data sets	89
Command Input Area	62	Adding Records to DAM Data Sets	90
Status Area	62	DAM Exclusive Control	91
Information Area	63		
PF Key Values Area	66		
Terminal Sharing	67		
Program Control	67		
Security Rules	68		
Installing the Command Level Interpreter	68		
Chapter 1.9. Temporary Storage Browse	69	Chapter 2.4. File Control - Commands, Options, and Conditions	93
The Browse Transaction	69	Read a Record (READ)	93
Using the Transaction	69	Write a Record (WRITE)	94
CEBR Commands	69	Update a Record (REWRITE)	94
Resource Definition	71	Delete a Record (DELETE) - VSAM Only	95
Part 2. Files and Data Bases	73	Release Exclusive Control (UNLOCK)	95
Introduction to Files and Data Bases	74	Start Browse (STARTBR)	95
Chapter 2.1. General Description of File Control Facilities	75	Read Next Record During a Browse (READNEXT)	96
VSAM Data Sets	75	Read Previous Record During a Browse (READPREV) - VSAM Only	96
Key-Sequenced Data Set	75	Reset Start of Browse (RESETBR)	97
Entry-Sequenced Data Set	76	End Browse (ENDBR)	97
Relative Record Data Set	76	File Control Options	98
VSAM Data Set Organization	76	File Control Exceptional Conditions	100
VSAM Paths and Bases	76		
VSAM Share Options	77		
DAM Data Sets	77		
Data Set Identification	78		
Accessing Data Sets From CICS Application Programs	78		
Retrieving Records	78		
Updating Records	81		
Deleting Records	82		
Adding Records	82		
Review of File Control Command Options	83		
Preventing Transaction Deadlocks	84		
KEYLENGTH Option for Remote Data Sets	86		
Chapter 2.2. File Control - VSAM Considerations	87	Chapter 2.5. DL/I Services (EXEC DLI Command)	105
		EXEC DLI Command	105
		General Format of EXEC DLI Command	106
		Schedule the PSB	106
		Get One or More Segments	106
		Insert One or More Segments	106
		Load a Segment (Batch only)	107
		Replace One or More Segments	107
		Delete a Segment	107
		Terminate Access to the PSB	107
		Request a Basic Checkpoint	107
		General Rules and Conventions	107
		DL/I Interface Block (DIB)	108
		Example of DL/I Requests Using EXEC DLI	109
		Chapter 2.6. DL/I Services (DL/I CALL Statement)	115
		User Interface Block (UIB)	116
		Schedule the PSB and Obtain PCB Addresses	116
		Segment Search Arguments	117
		I/O Work Area for DL/I Segments	117
		Issue a DL/I Data Base Call	118

Terminate a PSB in the CICS Application Program	118	Chapter 3.2-3. Standard Function BMS	161
Check the Response to a DL/I CALL	119	Text Processing	162
Check the CICS-DL/I Response Codes in UIBRCODE	119	Character Attribute Control	162
Check the DL/I Function	120	Printer Support	163
Examples of DL/I Requests	120	3270 Printers Without the NLEOM Option	163
Part 3. Data Communication Operations	127	3270 Printers With the NLEOM Option	163
Chapter 3.1. Introduction to Data Communication Operations	129	SCS and Other Non-3270 Printers	164
Chapter 3.2-1. Introduction to Basic Mapping Support	131	FORMFEED Option	164
How BMS Affects Programming	131	Printers and BMS Text	164
BMS Maps	132	Printers and Device Independence	165
BMS Map Definition	132	Partition Support	165
Cataloging BMS Map Sets	133	Applications of Partitions Under CICS	166
BMS Commands	133	How Existing Programs can Use Partitions	166
Facilities Provided by BMS	134	How New Application Programs Can Use Partitions	166
Sample Programs	134	How CICS Manages Partitions	167
Chapter 3.2-2. Minimum Function BMS	135	Partitioning Concepts	168
IBM 3270 Information Display System	135	Summary of Implementation and Use of Partitions	169
Input Operations	135	Application Programming	169
Output Operations	136	Defining Partition Sets	170
Display Field Concepts	136	Assembling and Cataloging a Partition Set	171
Screen Layout Design	139	Specifying a Partition Within a Map Set	171
Defining BMS Maps	140	How You Code Programs to Manage Partitions	172
Defining a Map Set	141	Logical Device Components	175
Defining Maps Within a Map Set	142	Defining Logical Device Components	175
Defining Fields Within a BMS Map	143	Sending Data to a Logical Device Component	175
Terminating a Map Set Definition	144	10/63 Magnetic Slot Reader Control	176
Example of Map Set Definitions	145	Application Programming	176
Assembling and Cataloging BMS Maps	145	Trigger Fields	177
Writing Programs to Use BMS Services	147	Application Programming	177
Copying Symbolic Description Maps	147	Outboard Formatting	179
Processing Data Structures Under BMS	148	Block Data Format	179
Sending Data to a Display	150	Chapter 3.2-4. Full Function BMS	181
Sending Device Controls Without Display Data	153	Logical Message Handling	181
Cursor Positioning	153	Logical Messages for Direct Terminal Output	183
Accessing Data Outside the Program	154	Logical Messages With the SET Option	184
Receiving Data From a Display	154	Logical Messages for Terminal Operator Paging	184
Responding to Terminal Input	156	Cumulative Output Processing (ACCUM Option)	186
Exceptional Conditions	157	Cumulative Text Formatting	192
Printed Output	158	Cumulative Processing and Device Controls	194
		Cumulative Processing and Partitions	195
		Cumulative Processing and Logical Device Components	195
		Message Routing	195

Defining a ROUTE List	196	Synchronize Terminal I/O for a Transaction (WAIT TERMINAL)	242
Disposition and Message Routing	197	Converse With Terminal or Logical Unit (CONVERSE)	242
Interleaving Conversation with Message Routing	197	Send an Asynchronous Interrupt (ISSUE SIGNAL)	242
TITLE Option of the ROUTE Command	198	Relinquish a Communication Line (ISSUE RESET)	242
Route List and Operator Class Codes (LIST and OPCLASS)	198	Disconnect a Switched Line (ISSUE DISCONNECT)	242
Route List Format	199	Terminal-Oriented Task Identification	243
Routing and Page Overflow	201	Commands and Options for Logical Units	244
Message Switching Transaction (CMMSG)	201	Send/Receive Mode	244
Returning Mapped Data to a Program Before Output	201	Send/Receive Protocol (Invite Option)	244
SET Option	201	Chaining of Input Data	244
Terminal Code Table	202	Chaining of Output Data	246
SEND TEXT MAPPED Command	203	Logical Record Presentation	246
SEND TEXT NOEDIT Command	203	Definite Response	247
Chapter 3.2-5. BMS Macro and Command Reference Summary	205	Function Management Header (FMH)	247
Map Set, Map, and Field Definition	205	Unsolicited Input	248
Map Set Definition Macro (DFHMSD)	205	Bracket Protocol (LAST option)	248
Map Definition Macro (DFHMDI)	206	Suspend a Task (WAIT SIGNAL)	248
Field Definition Macro (DFHMDF)	206	Terminate a Session (ISSUE DISCONNECT)	248
Ending a Map Set Definition	207	VTAM Application Routing (ISSUE PASS)	249
Map Definition Macro Operand Summary	207	Sync Point Processing (ISSUE PREPARE)	249
Partition Set Definition	220	Receipt of VTAM Logon Data (EXTRACT LOGONMSG)	249
Partition Set Definition Macro (DFHPSD)	220	Return a Facility to CICS (FREE)	250
Partition Definition Macro (DFHPDI)	220	BTAM Programmable Terminals	250
Ending a Partition Set Definition	220	Teletypewriter Programming	250
Partition Definition Macro Operand Summary	221	Message Format	251
BMS Related Constants	222	Message Length	251
Standard Attribute and Printer Control		Connection Through VTAM	252
Character List (DFHBMSCA)	222	Display Device Operations	252
Attention Identifier Constants (DFHAID)	224	Print Displayed Information (ISSUE PRINT)	252
Magnetic Slot Reader (MSR) Control		Copy Displayed Information (ISSUE COPY)	253
Value Constants (DFHMSRCA)	224	Erase All Unprotected Fields (ISSUE ERASEAUP)	253
Input Commands	225	Input Operation Without Data (RECEIVE)	253
Output Commands	226	Standard Attention Identifier List (DFHAID)	254
BMS Related ASSIGN Options	229	Handling Attention Identifiers (HANDLE AID)	254
BMS Options	230	Standard Attribute and Printer Control	
BMS Exceptional Conditions	236	Character List (DFHBMSCA)	255
Chapter 3.3. Terminal Control	239	Standard CICS Terminal Support (BTAM)	257
Commands and Options for Terminals and Logical Units	241		
Fullword Lengths	241		
Read From Terminal or Logical Unit (RECEIVE)	241		
Write to Terminal or Logical Unit (SEND)	241		

LUTYPE4 Logical Unit	257	3770 Interactive Logical Unit	279
LUTYPE6.1 Logical Unit	258	3770 Full Function Logical Unit	279
Session Status Information	259	3780 Communications Terminal	279
Application-Oriented Information	259	3790 Full Function Logical Unit	279
Session-Oriented Information	259	3790 Inquiry Logical Unit	280
LUTYPE6.2 Logical Unit (VTAM Only)	260	3790 SCS Printer Logical Unit	280
Synchronization Levels	260	3790 (3270-Display) Logical Unit	281
Session-Oriented Information	261	3790 (3270-printer) Logical Unit	281
System/3	262	4700 Finance Communication System	282
System/370	262	7770 Audio Response Unit	282
System/7	262	Terminal Control Options	283
2260 Display Station	264	Terminal Control Exceptional Conditions	290
2265 Display Station	264		
2741 Communication Terminal	264	Chapter 3.4. Report Controller	293
Read Attention	264	Reports	294
2770 Data Communication System	265	Identifying a Report	295
2780 Data Transmission Terminal	266	Report Name	295
2980 General Banking Terminal System	266	Report Title	295
Passbook Control	266	Userdata	295
Output Control	267	Tokens	295
Output to a Common Buffer	267	Report Number	296
The DFH2980 Structure	267	Creating a Report	296
3270 Information Display System (BTAM)	268	Report Types	296
3270 Logical Unit	269	Opening a Report	298
3270 SCS Printer Logical Unit	269	Writing Data to a Report	298
3270-Display Logical Unit (LUTYPE2) and 3270-Printer Logical Unit (LUTYPE3)	270	Closing a Report	299
3600 Finance Communication System (BTAM)	270	Formatting a Report	300
Input	270	Report Formats	300
Output	271	NOCC, ASA, and MCC Formats	300
Resend Message	271	SCS and T3270 Formats	303
3600 Pipeline Logical Unit	272	BMS Mapped Reports	305
3600 (3601) Logical Unit	272	ESCAPE Format	305
3600 (3614) Logical Unit	273	JCL Format	306
3630 Plant Communication System	273	Printing a Report	307
3650/3680 Host Command Processor Logical Unit	273	Controlling Where Reports are Printed	307
3650 Host Conversational (3270) Logical Unit	274	Controlling When Reports are Printed	308
3650 Host Conversational (3653) Logical Unit	274	Controlling How Reports are Printed	309
3650 Interpreter Logical Unit	275	Security and Recovery	310
3650 Pipeline Logical Unit	275	Resource Security Level	310
3650/3680 Full Function Logical Unit	275	PRINTFAIL	310
3660 Supermarket Scanning System	275	Report Recovery	310
3735 Programmable Buffered Terminal	276	Sync Point	311
3735 Transactions - Autoanswer	276	Report Controller Commands	311
3735 Transactions - Autocall or Time-Initiated	276	Open ASA, MCC, or NOCC Report	311
3740 Data Entry System	277	Open MAP Format Report	311
3767 Interactive Logical Unit	278	Open SCS or T3270 Report	312
3770 Batch Logical Unit	278	Open ESC Format Report	312
		Open JCL Format Report	312
		Write to a Report (not MAP)	312
		Close a Report	312
		Reopen a Report	312

Write to a MAP Report	313	Starting Tasks with Terminals but Without Data	343
Report Controller Options	313	Starting Tasks with Terminals and Data	344
Report Controller Exceptional Conditions ..	320	Retrieve Data Stored for a Task (RETRIEVE)	344
Chapter 3.5. Batch Data Interchange	323	Cancel Interval Control Requests (CANCEL)	345
Destination Selection and Identification ..	323	Interval Control Options	346
Definite-Response	324	Interval Control Exceptional Conditions	350
Waiting for Function Completion	324		
Interrogate a Data Set (ISSUE QUERY)	324		
Read a Record From a Data Set (ISSUE RECEIVE)	324		
Add a Record to a Data Set (ISSUE ADD)	325		
Update a Record in a Data Set (ISSUE REPLACE)	325		
Delete a Record from a Data Set (ISSUE ERASE)	326		
End Processing of a Data Set (ISSUE END)	326		
End Processing of a Data Set Abnormally (ISSUE ABORT)	326		
Send Data to an Output Device (ISSUE SEND)	326		
Request Next Record Number (ISSUE NOTE)	327		
Wait for an Operation to be Completed (ISSUE WAIT)	327		
Batch Data Interchange Options	327		
Batch Data Interchange Exceptional Conditions	329		
 Chapter 3.6. Structured Query Language/Data System (SQL/DS)	331	Chapter 4.5. Storage Control	365
Embedded SQL Commands	331	Obtain and Initialize Main Storage (GETMAIN)	365
Preparing a Program	332	Release Main Storage (FREEMAIN)	365
Part 4. Control Operations	335	Storage Control Options	366
Chapter 4.1. Introduction to Control Operations	337	Storage Control Exceptional Conditions	366
Chapter 4.2. Interval Control	339	Chapter 4.6. Transient Data Control	367
Request Current Date and Time of Day (ASKTIME)	340	Write Data to Transient Data Queue (WRITEQ TD)	369
Select the Format of Date and Time (FORMATTIME)	340	Read Data from Transient Data Queue (READQ TD)	369
Delay Processing of a Task (DELAY)	340	Delete an Intrapartition Transient Data Queue (DELETEQ TD)	370
Request Notification when Specified Time has Expired (POST)	341	Transient Data Control Options	370
Wait for an Event to Occur (WAIT EVENT)	342	Transient Data Control Exceptional Conditions	371
Start a Task (START)	342		
Starting Tasks without Terminals	343		
		Chapter 4.7. Temporary Storage Control	373
		Write Data to a Temporary Storage Queue (WRITEQ TS)	374

Read Data from Temporary Storage Queue (READQ TS)	375	Sync Point Exceptional Condition	406
Delete Temporary Storage Queue (DELETEQ TS)	375	Part 6. The CICS Built-In Function Command	407
Temporary Storage Control Options	376	Chapter 6.1. The Field Edit Built-In Function (BIF DEEDIT) Command	409
Temporary Storage Control Exceptional Conditions	377		
Part 5. Recovery and Debugging ..	379		
Chapter 5.1. Introduction to Recovery and Debugging	381		
Sequential Terminal Support	381	Appendixes	411
Chapter 5.2. Abnormal Termination Recovery ..	385		
Handle an Abnormal Termination Exit (HANDLE ABEND)	385	Appendix A. EXEC Interface Block	413
Terminate Task Abnormally (ABEND)	387	EIB Fields	413
Abnormal Termination Recovery Options ..	387		
Abnormal Termination Recovery Exceptional Condition	387	Appendix B. Translation Tables for the 2980 ..	421
Chapter 5.3. Trace Control	389		
Trace Entry Points	389	Appendix C. CICS Macros and Equivalent Commands	425
Event Monitoring Points	389		
Trace Facility Control	390	Appendix D. Sample Programs (Assembler Language)	427
Trace Table Format	390	Operator Instruction Program (ASM)	428
CICS Auxiliary Trace Facility	392	Description	428
User Trace Entry Point and Event Monitoring Point (ENTER)	392	Source Listing	428
Control the CICS Trace Facility (TRACE ON, TRACE OFF)	393	Program Notes	428
Macro-Level Trace Facilities	393	Inquiry/Update Sample Program (ASM)	429
Trace Control Options	393	Description	429
Trace Control Exceptional Conditions	394	Program Notes	434
Chapter 5.4. Dump Control	395	Browse Sample Program (ASM)	436
Dump Main Storage (DUMP)	396	Description	436
Dump Control Options	396	Program Notes	441
Dump Control Exceptional Conditions	398	Order Entry Sample Program (ASM)	442
Chapter 5.5. Journal Control	399	Description	442
Create a Journal Record (JOURNAL)	401	Program Notes	444
Synchronize with Journal Output (WAIT JOURNAL)	402	Order Entry Queue Print Sample Program (ASM)	446
Journal Control Options	402	Description	446
Journal Control Exceptional Conditions	403	Program Notes	447
Chapter 5.6. Recovery (Sync Points)	405	Low Balance Report Sample Program (ASM)	449
Establish a Sync Point (SYNCPOINT)	406	Description	449
Sync Point Option	406	Program Notes	451
		Maps and Screen Layouts for ASM Sample Programs	452
		DFH\$AGA Screen Layout	453
		DFH\$AGB Screen Layout	456
		DFH\$AGC Screen Layout	460
		DFH\$AGD Screen Layout	463
		DFH\$AGK Screen Layout	466
		DFH\$AGL Print Format	468
		Record Descriptions for ASM Sample Programs	469

FILEA Record Description	469	Operator Instruction Program (PL/I)	510
LOGA Record Description	469	Description	510
L86O Record Description	469	Source Listing	510
Appendix E. Sample Programs (COBOL) ..	471	Inquiry/Update Sample Program (PL/I) ..	511
Operator Instruction Program (COBOL) ..	472	Description	511
Description	472	Program Notes	515
Source Listing	472	Browse Sample Program (PL/I) ..	517
Inquiry/Update Sample Program (COBOL) ..	473	Description	517
Description	473	Program Notes	521
Program Notes	477	Order Entry Sample Program (PL/I) ..	523
Browse Sample Program (COBOL) ..	479	Description	523
Description	479	Program Notes	525
Program Notes	483	Order Entry Queue Print Sample Program (PL/I) ..	527
Order Entry Sample Program (COBOL) ..	484	Description	527
Description	484	Program Notes	528
Program Notes	486	Low Balance Report Sample Program (PL/I) ..	530
Order Entry Queue Print Sample Program (COBOL) ..	488	Description	530
Description	488	Program Notes	531
Program Notes	490	Maps and Screen Layouts for PL/I Sample Programs ..	533
Low Balance Report Sample Program (COBOL) ..	491	DFH\$PGA Screen Layout	534
Description	491	DFH\$PGB Screen Layout	537
Program Notes	492	DFH\$PGC Screen Layout	541
Maps and Screen Layouts for COBOL Sample Programs ..	494	DFH\$PGD Screen Layout	543
DFH\$CGA Screen Layout	495	DFH\$PGK Screen Layout	545
DFH\$CGB Screen Layout	498	DFH\$PGL Print Format	547
DFH\$CGC Screen Layout	502	Record Descriptions for PL/I Sample Programs ..	548
DFH\$CGD Screen Layout	504	Description	548
DFH\$CGK SCREEN LAYOUT	506	FILEA Record Description	548
DFH\$CGL Print Layout	507	LOGA Record Description	548
Record Descriptions for COBOL Sample Programs ..	508	L86O Record Description	548
Description	508	Index ..	557
FILEA Record Description	508		
LOGA Record Description	508		
L86O Record Description	508		
Appendix F. Sample Programs (PL/I) ..	509		

Figures

1. Translated Code for a CICS Command	13	14. Many BMS Maps on One Page	191
2. Translated Code for User Variables	14	15. Overflow Processing	193
3. Typical EDF Display	53	16. Example of Paging	194
4. "Stop-Conditions" Display	56	17. Terminal Oriented Task Identification . .	245
5. First Page of Typical EXEC DLI Display	59	18. BTAM Programmable Terminal Programming	251
6. Second Page of Typical EXEC DLI Display	60	19. SQL commands embedded in application programs	332
7. "Command Syntax Check" Display	63	20. Application Program Logical Levels . .	357
8. "About to Execute Command" Display . .	64	21. ABEND Exit Processing	386
9. "Command Execution Complete" Display	65	22. 2980-1 Teller Station Character Set/Translate Table	422
10. Initial Display Produced By the Browse Transaction	70	23. 2980-2 Administrative Station Character Set/Translate Table	423
11. Examples of Record Identification	90	24. 2980-4 Teller Station Character Set/Translate Table	424
12. BMS Map Set Suffixing Logic	146		
13. MSR Control Byte Values	178		

Summary of Amendments

Amendments for Latest Edition (CICS/DOS/VS 1.7)

This book includes information about the new or enhanced facilities introduced by CICS/DOS/VS Version 1 Release 7.

Some of these facilities, listed below, are not described in this publication because they do not directly affect the application programmer; for more information on these, refer to the other CICS publications listed in the bibliography.

- Multiregion performance improvements
- Temporary storage performance improvements
- Transient data performance improvements
- CICS monitoring facility improvements
- RDO for terminals and sessions
- Improved restart for CICS
- Task control improvements
- File control enhancements
- ISC enhancements
- Security enhancements
- Monitoring enhancements
- EXEC API enhancements
- VTAM interface enhancements
- BMS enhancements and query support
- Report controller
- System spooling interface
- Serviceability enhancements.

In particular, note that file control is now covered in four chapters, which have been designed to make file control easier to understand and use. The first three of these chapters have been completely rewritten and do not contain revision bars; they should be read in their entirety.

A new chapter (Chapter 3.4) has been added describing the report controller, which provides, in conjunction with POWER, an integrated spooling facility. Again, this chapter carries no revision bars and it should be read in its entirety.

Another new chapter (Chapter 3.6) has been added describing the use of Structured Query Language/Data System (SQL/DS) commands in an application program. Again, this chapter carries no revision bars and it should be read in its entirety.

You will find information about all the new facilities of CICS/DOS/VS Version 1 Release 7 in the *CICS/DOS/VS Release Guide*.

The following functions are obsolete and have been omitted from the manual:

- Static user exits
- Asynchronous transaction processing
- ISAM support.

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	11
Chapter 1.4. Programming Techniques and Restrictions	21
Chapter 1.5. Exceptional Conditions	31
Chapter 1.6. Access to System Information	41
Chapter 1.7. Execution (Command Level) Diagnostic Facility	49
Chapter 1.8. Command Level Interpreter	61
Chapter 1.9. Temporary Storage Browse	69

Chapter 1.1. Introduction to Command Level Programming

The IBM Customer Information Control System/Virtual Storage (CICS/VS) command level application programming interface allows you to request CICS services by means of **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, PL/I, or RPGII. The commands are essentially the same in each language, differing only in the delimiter used, and, in the case of RPGII only, in the syntax.

Because of its fixed format, RPGII is not included in this manual. Instead, a separate manual is available entitled *CICS/DOS/VS Application Programmer's Reference Manual (RPGII)*.

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.

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

As a general rule, 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 41.

For a basic description of CICS, see the *CICS/VS Application Programming Primer*, which has been designed to teach you, 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/DOS/VS 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.

In general, 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 implemented at command level.

If you have an existing macro level program that you enhance using CICS commands, you must be aware that in some cases you will get unpredictable results, because of addressability and storage problems.

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.

Syntax Notation Used in This Manual

Throughout this manual, wherever a CICS command is shown, the symbols { }, [], |, and ... are used in defining the command format. These symbols are not part of the command and you should never include them in your code. Their purpose is to tell you how the command may be written, and you should interpret them as follows:

- Code punctuation symbols and uppercase characters exactly as shown.
- Lowercase characters indicate that user text should be coded as required. The character "b" is used in some places to indicate a blank.
- Square brackets [] are used to indicate that the enclosed identifiers are optional. The less than and greater than symbols < > are used to replace square brackets in the syntax displays produced by the command level interpreter. (See "Chapter 1.8. Command Level Interpreter" on page 61.)
- The "or" symbol | is used to separate alternatives.
- Underlining is used to denote that the identifier is the default; that is, the one that will be assumed if no explicit choice is made.
- Braces { } are used to enclose a set of alternatives, one of which must be coded.
- The ellipsis ... denotes that the immediately preceding identifier(s) can be coded repetitively.

To denote, for example, that either GTEQ, or EQUAL, or neither, can be coded (and that GTEQ is the default), the syntax notation would be:

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 following chapters.

Command Format

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

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
  DATASET('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 indicates that the rest of the line is a comment.
- The usual continuation conventions for macro instructions 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" as shown in the following 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 as shown in the following 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 halfword binary value. This puts a theoretical upper limit of 32,767 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 *journalized 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 399).

Finally, for *temporary storage, transient data, and file control*, the dataset definitions may themselves impose further restrictions. You'll find any such

restrictions documented in the books describing installation and resource definition.

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

The argument values are defined in the following sections.

Argument Values in Assembler Language

In general, 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 which use the macro replacement facilities.

An absolute expression must be a single term which 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 = 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 value of L'data-area.

Although the DESTIDLENG, FROMLENGTH, KEYLENGTH, LENGTH, PFXLENG,

TOLENGTH, or VOLUMELENG options are shown as required options in the syntax for a command, these options are always optional in an assembler language program which specifies a data area (except in the case of 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 or not 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 the name of any BLL (base locator for linkage) cell, or by any COBOL data name which contains a copy of such a pointer in a BLL cell.
- “pointer-ref” can be replaced by the name of any BLL cell.

- “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 which 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, or VOLUMELENG options may be shown as required options in the syntax for a command, these options are always optional in a PL/I program which specifies a data area (except in the case of 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 or not 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; refer to the *CICS/DOS/VS Installation and Operations Guide* for details. The translator requires a partition 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 SYSIPT.

- | For COBOL and PL/I, the translator writes its output (the translated source program) on SYSPCH, and writes the source listing, error messages, and so on, on SYSLST.
- | For assembler language, the translator writes its output to SYSPCH, which also contains error messages (if any) as assembler comments. SYSLST 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 accept also the commands that can be used to access DL/I data bases. 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 105.

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 must contain 80-byte fixed-length unblocked records.

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 unblocked records.

- | Subsequent 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 must contain 121-byte fixed-length unblocked records.

Translated Code

Assembler Language

The invocation of a CICS assembler-language application program obeys system standards, which 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 no COMMAREA, entry is X'FF000000'.
- 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, whose contents are:

- 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.

As well as the invocation of the DFHECALL macro, the translator also inserts the following macros into your source program:

DFHEIENT This macro is inserted after the first CSECT or START instruction. It performs 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).

```

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

```

The above source program is translated to:

INSTRUCT	DFHEIGBL ,	INSERTED BY TRANSLATOR
	CSECT	
*	DFHEIENT	INSERTED BY TRANSLATOR
	EXEC CICS SEND MAP('DFH\$AGA') MAPONLY ERASE	
	DFHECALL =X'1804C0000800000000046204000020', (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

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 which describes the EIB is also included automatically.

The example in Figure 1 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.

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. (Note that 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 14 shows a simple assembler-language application program that uses such variables in dynamic storage.

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 only sets up 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 translator option NOPROLOG. This will enable you to specify your own DFHEIENT macro with the CODEREG operand so that you can specify more

```

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$AGA')
DFHECALL =X'1802C000080000000040900000020',(CHA7,=CL7 'DFH$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$AGB*
  ',( RF,DFH$AGB0)
DFHEIRET           INSERTED BY TRANSLATOR
DFHEISTG           INSERTED BY TRANSLATOR
DFHEIEND           INSERTED BY TRANSLATOR
END

```

Figure 2. Translated Code for User Variables

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 as follows:

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

For example, the source code shown in Figure 1 on page 13 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 either explicitly specified or obtained by default. It is recommended that, because register 13 points to the save area defined in dynamic storage by DFHEISTG, you use register 13 as the first dynamic storage register.

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

Assembler language programs, translated with the DLI option, will have a DLI initialization call inserted after each CSECT statement. Assembler language programs larger than 4095 bytes that 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.

| Note that, in general, 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. In this case, DFHEIPLR will not be assumed by the expansion 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.

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

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

may be translated to:

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

Declarations for the generated variables DFHEIVO and DFHC0070 are included automatically in working storage; their names are reserved. The string within the quotes moved to DFHEIVO consists of characters some of which may be unprintable. The use of EXEC, CICS, DLI, END-EXEC, or names starting with DFH, as names for user variables should be avoided.

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.

It is possible to translate program segments for later inclusion into the procedure division. In order to use the library management functions MAINT (DOS/VS(E)) or LIBRARIAN (VSE/SP) to catalog pretranslated code for later inclusion in input to the compiler, a job of the following sequence should be run:

- Step 1 punches a JOB card and JCL to invoke the required library management function
- Step 2 is the translator which punches translated code to follow the JCL in step 1
- Step 3 punches the end of job JCL.

The combined punch output must then be routed back to the job input queue (via the spooler, for example POWER, or by physically handling cards) so that the library management function saves the translated output in the library.

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 commences with a valid PL/I **PROCEDURE** statement, the translator will insert the declaration of the EIB structure.

It is possible to translate program segments for later inclusion into a main program. In order to use the library management functions **MAINT** (DOS/VS(E)) or **LIBRARIAN** (VSE/SP) to catalog pretranslated code for later inclusion in input to the compiler, a job of the following sequence should be run:

- Step 1 punches a **JOB** card and **JCL** to invoke the required library management function
- Step 2 is the translator which punches translated code to follow the **JCL** in step 1
- Step 3 punches the end of job **JCL**.

The combined punch output must then be routed back to the job input queue (via the spooler, for example **POWER**, or by physically handling cards) so that the library management function saves the translated output in the library. It is possible to translate program segments for later inclusion into a main program.

Translator Options

The translator provides a number of 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 (indicated by underlines) 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. These statements must precede

the source program; there is no batching facility. 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. If coded in the **PARM** operand of the **EXEC** job control statement, the **XOPTS** 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.

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.

DEBUG|NODEBUG (COBOL and PL/I only)

specifies whether or not 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 Execution Diagnostic Facility 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)

EPILOG specifies that the translator is to insert the macro DFHEIRET. NOEPILOG prevents the translator inserting the macro DFHEIRET. DFHEIRET is described on page 13.

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

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/DOS/VS Intercommunication Facilities Guide*.

LANGLVL(1)|LANGLVL(2) (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.

PROLOG|NOPROLOG (ASM only)

PROLOG specifies that the translator is to insert the macros DFHEISTG, DFHEIEND, and DFHEIENT. NOPROLOG prevents the translator inserting the macros DFHEISTG, DFHEIEND, and DFHEIENT. These macros are described on page 12.

NUM|NONUM (COBOL only)

specifies whether or not 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 or not 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)

QUOTE indicates to the translator that the double quotation marks ("") should be accepted as the character to delineate literals; APOST indicates 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)

indicates whether or not 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.

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 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 in the source listing its own sequence numbers; this is necessary 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 eight characters and must not overlap the source program (as specified in the MARGINS option).

The default is SEQUENCE(73,80).

SOURCE|NOSOURCE (COBOL and PL/I only)

specifies whether or not the translator is to produce a listing of the source program. For compatibility with previous releases, LIST|NOLIST is acceptable as equivalent.

SPACE(n) (COBOL only)

indicates the required 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

SPIE 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 or not 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 to improve 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. Refer to the *CICS/DOS/VS Resource Definition (Macro)* manual.

General Programming Techniques

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

Multithreading: is a technique, used by CICS, that 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 that all CICS application programs 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 (provided that, 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 insured 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; refer to "Chapter 4.3. Task Control" on page 353 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 to the total number of pages in a program) for relatively long periods.
- **Working set** - the number and combination of pages of a program needed 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.
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 awaiting a reply from a terminal user.
 - d. Use command-level file control locate-mode input/output rather than move-mode.

- 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, in rows rather than in columns.
- g. Issue as few as possible GETMAIN commands.
- h. In COBOL programs, avoid using EXAMINE or VARIABLE MOVE operations, because these expand into subroutine executions.

- 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.

No attempt should be made to 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 unreference 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 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, note that 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

Care should be exercised 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. 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.

Any program that contains a mixture of commands and macros should return to the program that called it using an EXEC CICS RETURN command.

For example, 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 only be released in the command level program. Do not use a DFHISC 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.

Program Size

The load module resulting from any application program must not occupy more than 524,152 bytes of main storage.

Entry Point Address

For all programs, the entry point address must be less than 32,768 bytes from the load point.

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.

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).

Commands Contained Within Macros and COPY Code

Macro instructions 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, PL/I, or RPGII application programs using LINK or XCTL commands (see "Chapter 4.4. Program Control" on page 355).

However, because assembler-language application programs containing commands are invoked by a system standard call, they can be invoked also by a COBOL, PL/I, or RPGII 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, PL/I, or RPGII, but with only one. 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 such that this stub is the first CSECT in the load module.

Because assembler-language application programs containing commands are always passed the parameters EIB and COMMAREA when 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.

COBOL Considerations

Restrictions

The following restrictions apply to a COBOL program that is to be used as a CICS application program. (Refer to 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 the special options:

REPORT WRITER
SEGMENTATION
SORT
TRACE

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

COUNT
DYNAM
FLOW
STATE
STXIT
SYMDMP

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

ACCEPT
CURRENT-DATE
DATE
DAY
DISPLAY
EXHIBIT
INSPECT
SIGN IS SEPARATE
STOP RUN¹
STRING
TIME
UNSTRING

¹ Because STOP RUN can be generated by the COBOL compiler, you must always code either a COBOL GOBACK statement or a CICS RETURN command at the end of your program.

6. You should not use the 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. Optimization option of the DOS Full COBOL V3 compiler cannot be used.

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

9. The length of working storage plus the length of the TGT (task global table) must not exceed 64K bytes.

10. 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.

Compilers Supported

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

- DOS Full COBOL Version 3 Compiler (5736-CB2)
- DOS/VS COBOL Compiler (5746-CB1).

Base Locator for Linkage

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 were 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 358.)

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 data 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.

BLL and Chained Storage Areas

If access is needed 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 needed 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 in the linkage section, additional statements are required to establish addressability to the extra area. An additional BLL cell is required for each extra 4096 bytes (or part) added to the 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 they address is not maintained.) An ADD statement is required also for each extra 4096 bytes (or part); it is placed after the statement that establishes addressability to the data area.

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.

BLL and the Optimization Feature

If an application program is to be compiled using the DOS/VS COBOL compiler with the optimization (OPT) feature, a special compiler control statement must be inserted at appropriate places within the program to ensure addressability to a particular area defined in the linkage section. This control statement has the form:

```
SERVICE RELOAD fieldname
```

where "fieldname" is the symbolic name of a specific storage area which is also defined in an 01-level statement in the linkage section. The SERVICE RELOAD statement must be used following each statement which 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 for 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 is used (described earlier in the chapter), 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 used to establish addressability to the data, as follows:

```
EXEC CICS HANDLE CONDITION
  ERROR(GIVEUP)
  LENGERR(BADLEN)
  END-EXEC
EXEC CICS READ DATASET('FILEA')
  RIDFLD(PART-REQD)
  SET(A-POINTER)
  LENGTH(A-LRECL)
  END-EXEC
SERVICE RELOAD A-DATA.
BADLEN.
SERVICE RELOAD A-DATA.
```

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.
```

If areas larger than 4096 bytes are being addressed, the secondary BLL cells must be reset after the SERVICE RELOAD statement has been executed. (Resetting a BLL cell is described in the previous section.)

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.

In order to use the library management functions MAINT (DOS/VS(E)) or LIBRARIAN (VSE/SP) to catalog pretranslated code for later inclusion in input to the compiler, a job of the following sequence should be run:

- Step 1 punches a JOB card and JCL to invoke the required library management function
- Step 2 is the translator which punches translated code to follow the JCL in step 1
- Step 3 punches the end of job JCL.

The combined punch output must then be routed back to the job input queue (via the spooler, for example POWER, or by physically handling cards) so that the library management function saves the translated output in the library.

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

PL/I Considerations

Restrictions

The following restrictions apply to a PL/I program that is to be used as a CICS application program.

Refer to 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. You should 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.)

Refer to 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). A consequence of this restriction for PL/I users is that CONTROLLED variables cannot be used.
6. If you declare a variable with the attributes STATIC and EXTERNAL you should 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.

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/DOS/VS Recovery and Restart Guide*.

Compilers Supported

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

- DOS PL/I Optimizing Compiler, Version 1, Release 5.0

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 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.

MAINT (DOS/VS(E)) or LIBRARIAN (VSE/SP) to catalog pretranslated code for later inclusion in input to the compiler, a job of the following sequence should be run:

- Step 1 punches a JOB card and JCL to invoke the required library management function
- Step 2 is the translator which punches translated code to follow the JCL in step 1
- Step 3 punches the end of job JCL.

The combined punch output must then be routed back to the job input queue (via the spooler, for example POWER, or by physically handling cards) so that the library management function saves the translated output in the library.

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.

In order to use the library management functions

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 alphabetical order of name, at the end of this chapter, and under EIBRCODE in Appendix A, “EXEC Interface Block” on page 413. Their numbers are listed in numerical order under EIBRESP also in Appendix A.

There are three possible states that an application program can be in with respect to a condition detected during the attempted execution of a command, as follows:

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 conditions ENQBUSY, NOJBUFSP, NOSPACE, NOSTG, QBUSY, SESSBUSY, and SYSBUSY, 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 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.

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 ...
```

As 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 POWER.

These commands are primarily for the use of the system programmer, and are described in the *CICS/DOS/VS 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.

You should not include commands in an error routine that may give rise to the same condition that caused the branch to the routine; you should 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.

You should 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, 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 abend code AEY7, as described for the DFHEIP module in the *CICS/DOS/VS 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. Otherwise the condition table is scanned to see what to do.
 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, then:
 - 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 found, this entry determines the action.
 - If ERROR is searched for and not found, the task is abended.
- The commands ALLOCATE, ENQ, GETMAIN, JOURNAL, READQ TD, and WRITEQ TS 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 option NOSUSPEND 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 fields EIBFN, EIBRCODE, and EIBRESP (in the EIB) to find out which condition has occurred and in which command. The EIB is described in Appendix A, "EXEC Interface Block" on page 413.

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.

You will find a list of the conditions that can be used in this command at the end of the chapter.

You cannot include more than sixteen conditions in the same command; the conditions 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. (Refer to "Chapter 4.4. Program Control" on page 355 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's 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.

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.

You will find a list of the conditions that can be used in this command at the end of the chapter.

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; the conditions should be separated by at least one space. You must specify additional conditions in further IGNORE CONDITION commands.

Suspend Condition Handling (PUSH and POP)

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

Normally, when a CICS program calls a subroutine, the program or routine that receives control inherits the current HANDLE commands. These commands may not be appropriate within the called program. The called program can use the command, PUSH, 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, refer to the list of conditions at the end of the indicated chapter.

Condition	Command	Chapter	Condition	Command	Chapter
CBIDERR	ALLOCATE CONVERSE EXTRACT ATTACH SEND	(3.3)	EOF	CONVERSE RECEIVE	(3.3) (3.3)
CCERROR	SPOOLOPEN SPOOLWRITE	(3.4) (3.4)	ERROR	General exceptional condition (1.5). Not included in the list of conditions in the syntax of individual commands.	
DISABLED	ENDBR DELETE READ READNEXT READPREV RESETBR REWRITE STARTBR UNLOCK WRITE	(2.4) (2.4) (2.4) (2.4) (2.4) (2.4) (2.4) (2.4) (2.4) (2.4)	EXPIRED	DELAY POST	(4.2) (4.2)
DSIDERR	ENDBR DELETE READ READNEXT READPREV RESETBR REWRITE STARTBR UNLOCK WRITE	(2.4) (2.4) (2.4) (2.4) (2.4) (2.4) (2.4) (2.4) (2.4) (2.4)	IGREQCD	ISSUE ABORT ISSUE ADD ISSUE END ISSUE ERASE ISSUE NOTE ISSUE QUERY ISSUE REPLACE ISSUE SEND ISSUE WAIT	(3.5) (3.5) (3.5) (3.5) (3.5) (3.5) (3.5) (3.5) (3.5)
DSSTAT	ISSUE RECEIVE	(3.5)	IGREQID	SEND CONTROL SEND MAP SEND TEXT	(3.2) (3.2) (3.2)
DUPKEY	READ READNEXT READPREV	(2.4) (2.4) (2.4)	ILLOGIC	DELETE ENDBR READ READNEXT READPREV RESETBR REWRITE SPOOLCLOSE SPOOLOPEN STARTBR UNLOCK WRITE	(2.4) (2.4) (2.4) (2.4) (2.4) (2.4) (2.4) (3.4) (3.4) (2.4) (2.4) (2.4)
DUPREC	WRITE REWRITE	(2.4) (2.4)			
ENDDATA	RETRIEVE	(4.2)			
ENDFILE	READNEXT READPREV	(2.4) (2.4)			
ENDINPT	RECEIVE	(3.3)			
ENQBUSY	ENQ	(4.3)	INBFMH	CONVERSE RECEIVE	(3.3) (3.3)
ENVDEFERR	RETRIEVE	(4.2)			
EOC	ALLOCATE CONVERSE RECEIVE RECEIVE MAP RECEIVE PARTN SEND	(3.3) (3.3) (3.3) (3.2) (3.2) (3.3)	INVERRTERM INVLDCL	ROUTE ROUTE SEND CONTROL SEND MAP SEND TEXT	(3.2) (3.2) (3.2) (3.2) (3.2)
EODS	CONVERSE ISSUE RECEIVE RECEIVE RECEIVE MAP RECEIVE PARTN	(3.3) (3.5) (3.3) (3.2) (3.2)	INVMPSZ INVPARTN	RECEIVE MAP SEND MAP RECEIVE MAP RECEIVE PARTN	(3.2) (3.2) (3.2) (3.2)

Condition	Command	Chapter	Condition	Command	Chapter
LENGERR (Continued)	WRITE WRITEQ TD	(2.4) (4.6)	NOTAUTH (Continued)	READNEXT READPREV RELEASE RESETBR RETRIEVE RETURN REWRITE SPOOLCLOSE SPOOLOPEN SPOOLWRITE	(2.4) (2.4) (4.4) (2.4) (4.2) (4.4) (2.4) (3.4) (3.4) (3.4)
MAPERROR	SPOOLWRITE	(3.4)		START STARTBR UNLOCK WAIT JOURNAL WRITE WRITEQ TD WRITEQ TS XCTL	(4.2) (2.4) (2.4) (5.5) (2.4) (4.6) (4.7) (4.4)
MAPFAIL	RECEIVE MAP SPOOLWRITE	(3.2) (3.4)		CANCEL DELETE READ READNEXT READPREV RESETBR RETRIEVE STARTBR	(4.2) (2.4) (2.4) (2.4) (2.4) (4.2) (2.4)
NAMEERROR	SPOOLOPEN	(3.4)	NOTFND	DELETE ENDBR JOURNAL READ READNEXT READPREV RESETBR RETRIEVE STARTBR	(2.4) (2.4) (5.5) (2.4) (2.4) (2.4) (4.2) (2.4)
NOBUFSPP	JOURNAL	(5.5)	NOTOPEN	DELETE ENDBR JOURNAL READ READNEXT READPREV RESETBR RETRIEVE STARTBR	(2.4) (2.4) (5.5) (2.4) (2.4) (2.4) (4.2) (2.4)
NONVAL	ISSUE LOAD	(3.3)		DELETE ENDBR JOURNAL READ READNEXT READPREV RESETBR RETRIEVE STARTBR	(2.4) (2.4) (5.5) (2.4) (2.4) (2.4) (4.2) (2.4)
NOPASSBKRD	RECEIVE	(3.3)		DELETE ENDBR JOURNAL READ READNEXT READPREV RESETBR RETRIEVE STARTBR	(2.4) (2.4) (5.5) (2.4) (2.4) (2.4) (4.2) (2.4)
NOPASSBKWR	SEND	(3.3)		DELETE ENDBR JOURNAL READ READNEXT READPREV RESETBR RETRIEVE STARTBR	(2.4) (2.4) (5.5) (2.4) (2.4) (2.4) (4.2) (2.4)
NOSPACE	REWRITE SPOOLCLOSE SPOOLOPEN SPOOLWRITE WRITE WRITEQ TD WRITEQ TS	(2.4) (3.4) (3.4) (3.4) (2.4) (4.6) (4.7)		DELETE ENDBR JOURNAL READ READNEXT READPREV RESETBR RETRIEVE STARTBR	(2.4) (2.4) (5.5) (2.4) (2.4) (2.4) (4.2) (2.4)
NOSPOOL	SPOOLCLOSE SPOOLOPEN SPOOLWRITE	(3.4) (3.4) (3.4)		DELETE ENDBR JOURNAL READ READNEXT READPREV RESETBR RETRIEVE STARTBR	(2.4) (2.4) (5.5) (2.4) (2.4) (2.4) (4.2) (2.4)
NOSTART	ISSUE LOAD	(3.3)		DELETE ENDBR JOURNAL READ READNEXT READPREV RESETBR RETRIEVE STARTBR	(2.4) (2.4) (5.5) (2.4) (2.4) (2.4) (4.2) (2.4)
NOSTG	GETMAIN	(4.5)		DELETE ENDBR JOURNAL READ READNEXT READPREV RESETBR RETRIEVE STARTBR	(2.4) (2.4) (5.5) (2.4) (2.4) (2.4) (4.2) (2.4)
NOTALLOC	CONNECT PROCESS CONVERSE EXTRACT ATTACH EXTRACT PROCESS FREE ISSUE ABEND ISSUE CONFIRMATION ISSUE DISCONNECT ISSUE ERROR ISSUE SIGNAL POINT RECEIVE SEND WAIT CONVID WAIT TERMINAL	(3.3) (3.3) (3.3) (3.3) (3.3) (3.3) (3.3) (3.3) (3.3) (3.3) (3.3) (3.3) (3.3) (3.3) (3.3) (3.3) (3.3)		DELETE ENDBR JOURNAL READ READNEXT READPREV RESETBR RETRIEVE STARTBR UNLOCK WAIT JOURNAL WRITE WRITEQ TD OVERFLOW PARTNFAIL PGMIDERR HANDLE ABEND LINK LOAD RELEASE XCTL	(2.4) (2.4) (5.5) (2.4) (2.4) (2.4) (4.6) (2.4) (2.4) (2.4) (2.4) (2.4) (2.4) (2.4) (2.4) (2.4) (2.4) (2.4) (2.4) (2.4)
NOTAUTH	CANCEL DELETE DELETEQ TD DELETEQ TS ENDBR LINK LOAD JOURNAL READ READQ TD READQ TS	(4.2) (2.4) (4.6) (4.7) (2.4) (4.4) (4.4) (5.5) (2.4) (4.6) (4.7)	QBUSY QIDERR	READQ TD DELETEQ TD DELETEQ TS READQ TD READQ TS WRITEQ TD WRITEQ TS	(4.6) (4.6) (4.7) (4.6) (4.7) (4.6) (4.7)

Condition	Command	Chapter	Condition	Command	Chapter
QZERO	READQ TD	(4.6)	SYSIDERR (Continued)	FREE	(3.3)
RDATT	CONVERSE	(3.3)		READ	(2.4)
	RECEIVE MAP	(3.2)		READNEXT	(2.4)
	RECEIVE	(3.3)		READPREV	(2.4)
RETPAGE	SEND CONTROL	(3.2)		READQ TD	(4.6)
	SEND MAP	(3.2)		READQ TS	(4.7)
	SEND PAGE	(3.2)		RESETBR	(2.4)
	SEND TEXT	(3.2)		RETRIEVE	(4.2)
ROLLEDBACK	SYNCPOINT	(5.6)		REWRITE	(2.4)
RTEFAIL	ROUTE	(3.2)		START	(4.2)
RTESOME	ROUTE	(3.2)		STARTBR	(2.4)
SELNERR	ISSUE ABORT	(3.5)	TERMERR	UNLOCK	(2.4)
	ISSUE ADD	(3.5)		WRITE	(2.4)
	ISSUE END	(3.5)		WRITEQ TD	(4.6)
	ISSUE ERASE	(3.5)		WRITEQ TS	(4.7)
	ISSUE NOTE	(3.5)		CONVERSE	(3.3)
	ISSUE QUERY	(3.5)		ISSUE ABEND	(3.3)
	ISSUE REPLACE	(3.5)		ISSUE CONFIRMATION	(3.3)
	ISSUE SEND	(3.5)		ISSUE COPY	(3.3)
	ISSUE WAIT	(3.5)		ISSUE DISCONNECT	(3.3)
SESSBUSY	ALLOCATE	(3.3)		ISSUE EODS	(3.3)
SESSIONERR	ALLOCATE	(3.3)		ISSUE ERASEAUP	(3.3)
	CONVERSE	(3.3)		ISSUE ERROR	(3.3)
	EXTRACT ATTACH	(3.3)		ISSUE LOAD	(3.3)
	FREE	(3.3)		ISSUE PRINT	(3.3)
	ISSUE DISCONNECT	(3.3)		ISSUE SIGNAL	(3.3)
	ISSUE SIGNAL	(3.3)		RECEIVE	(3.3)
	POINT	(3.2)		SEND	(3.3)
	RECEIVE	(3.2)		WAIT SIGNAL	(3.3)
	SEND	(3.3)	TERMIDERR	ISSUE COPY	(3.3)
	WAIT TERMINAL	(3.3)		START	(4.2)
SIGNAL	CONVERSE	(3.3)	TRANSIDERR	START	(4.2)
	RECEIVE	(3.3)	TSIOERR	PURGE MESSAGE	(3.2)
	SEND	(3.3)		SEND CONTROL	(3.2)
	WAIT CONVID	(3.3)		SEND MAP	(3.2)
	WAIT SIGNAL	(3.3)		SEND PAGE	(3.2)
	WAIT TERMINAL	(3.3)		SEND TEXT	(3.2)
SYSBUSY	ALLOCATE	(3.3)	UNEXPIN	ISSUE ABORT	(3.5)
SYSIDERR	ALLOCATE	(3.3)		ISSUE ADD	(3.5)
	CANCEL	(4.2)		ISSUE END	(3.5)
	DELETE	(2.4)		ISSUE ERASE	(3.5)
	DELETEQ TD	(4.6)		ISSUE NOTE	(3.5)
	DELETEQ TS	(4.7)		ISSUE QUERY	(3.5)
	ENDBR	(2.4)		ISSUE RECEIVE	(3.5)
				ISSUE REPLACE	(3.5)
				ISSUE SEND	(3.5)
				ISSUE WAIT	(3.5)
				RECEIVE MAP	(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; refer to the *CICS/VS Application Programmer's Reference Manual (Macro Level)* 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/DOS/VS 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. This means that you should not use any of the CICS fields as arguments in CICS commands, because these fields may be altered by the EXEC interface modules.

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/DOS/VS 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 defined in Appendix A, "EXEC Interface Block" on page 413.

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 common system area (CSA), the common work area (CWA), the terminal control table user area (TCTUA), the transaction work area (TWA), and to the EXEC interface block (EIB). No more than four options can be specified in one ADDRESS command.

ADDRESS Command Options

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.

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 355.)

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 under "BLL and the Optimization Feature" on page 27.)

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.

Example 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 BLLCLELS.
  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.

ABCODE

specifies a variable that is set to the current value of the abend code (abend codes are documented in the *CICS/DOS/VS 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.

BTRANS

specifies that the value required is an indicator showing that 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 that the terminal is defined as having the extended color capability (X'FF'); or no extended color capability (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, 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 201.

DESTID

specifies that the value required is the identifier of the outboard destination, padded with blanks on the right to eight 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 that the terminal is defined as having the extended data stream capability (X'FF'); or no extended data stream capability (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.

FACILITY

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

Note: You should always use the QNAME option (described on page 47) 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 TCAFCAI, 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.

HIGHLIGHT

specifies that the value required is an indicator showing that the terminal is defined as having the extended highlight capability (X'FF'); or no extended highlight capability (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 that the terminal supports magnetic slot reader (MSR) control (X'FF') or does 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. If the task is not initiated from a terminal, INVREQ occurs. The format of the value is an 8-byte character string.

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 eight byte operator security key, copied from fields in the TCTTE. 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 operator security key, copied from TCTTESK. If this option is specified and there is no TCTTE for the task, INVREQ occurs. This option returns only the 24-bit mask corresponding to operator security key values 1 through 24. For the 64-bit mask use

the OPERKEYS option. The format of the value is a 3-byte character string.

OUTLINE

specifies that the value required is an indicator showing that 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 BMS fast-path 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 that the terminal supports partitions (X'FF') or does 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

specifies that the value required is the name of the TCTSE (terminal control table system entry) associated with the principal facility. If

there is no TCTTE for the task or if the principal facility is not an LU6 or MRO session, INVREQ occurs. The format of the value is a 4-byte character string.

PS

specifies that the value required is an indicator showing that the terminal is defined as having the programmed symbols capability (X'FF'); or no programmed symbols capability (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 current 3270 screen. If this option is specified and there is no TCTTE for the task, INVREQ occurs. The format of the value is halfword binary.

SCRNWID

specifies that the value required is the width of the current 3270 screen. 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 that 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 which 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 null 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.

VALIDATION

specifies that the value required is an indicator showing that 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 translator option EDF, 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 (the PCT). You must also ensure that the programs and maps that are used by EDF are specified in the processing program table (the PPT). EDF uses temporary storage and BMS. You can only use EDF 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 macro instructions are coded, or the terminal control table (the TCT) is in application programs monitored by EDF.

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 which uses partitions, or which does its own request unit (RU) chaining, (RU chaining is described in "Chaining of Input Data" on page 244) 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 51.

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 (the EIB) and the DL/I interface block (the DIB).
 - The program's working storage in hexadecimal and character form.
 - The last ten commands executed, including all argument values, responses, and so on.
 - The contents (in hexadecimal) of any address location within the CICS partition.
- 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 that command displays are 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 EDF has security required, 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 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/DOS/VS Resource Definition (Online)* manual or the *CICS/DOS/VS 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, you should 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 four-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 that 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 routing transaction CRTE. 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 3 on page 53. 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. Note that PF10 and PF11 become undefined if there are no further displays backward or forward respectively.

```

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 3. Typical EDF Display

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 57.

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), then 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) keys. 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 69.

CONTINUE

causes the user transaction to continue unless the screen has been modified. In the latter case, EDF redisplay 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 108 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 413 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 4 on page 56, 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.

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 4. "Stop-Conditions" Display

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.

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 later under "Overtyping EDF Displays."

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.

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 partition by overtaking the address field.

Overtyping EDF Displays

As mentioned above, certain areas of an EDF display can be overtaken. These areas can be identified by use of the tab keys; the cursor stops only at fields that can be overtaken (excluding fields within the menu).

- Any command can be overtaken with "NOOP" or "NOP" before execution; this suppresses execution of the command. Use of the ERASE EOF key, or overtaking 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 overtaken, but not the keyword of the argument. An optional argument cannot be removed, nor can an option be added or deleted. Overtaking must not extend beyond the argument value displayed. Any modification that is not overtaking 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 overttyped 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 **EIBR CODE** and **EIBRESP** fields, when displayed as part of the **EXEC** interface block, can be overttyped with any desired hexadecimal codes. This does not apply when these fields are part of a command display.

When a field representing a data area of a program is overttyped, 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 overttyped, 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 overttyped 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 overttyping 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, overttyping 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 5, and in Figure 6 on page 60.

<pre> TRANSACTION: XDLI PROGRAM: UPDATE 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 </pre>	TASK NUMBER: 0000111 DISPLAY: 00
--	--

Figure 5. 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')			
SEGLLENGTH (+00010)			
LOCKED			
INTO ('SMITH')			
WHERE (ACCOUNT = '12345')			
FIELDLENGTH (+00005)			
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 6. 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 screen. 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 data base 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 master terminal transaction CEMT.

Invoking the Command Level Interpreter

The command level interpreter is a CICS application program and runs as a CICS transaction. It is started by the transaction identification of "CECI", or "CECS", 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 throughout 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 DATASET('FILEA')

will give the screen display shown in Figure 7. Note that 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/DOS/VS Installation and Operations 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 68.

Modifying the command input to:

READ DATASET('FILEA') RIDFLD('009000')

will give the screen display shown in Figure 8. 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 9 will appear showing the result of execution.

It is possible to prevent unauthorized access by the interpreter to resources such as data sets. Refer to the security rules later in the 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 7 on page 63. 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 illustrated 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 65.
- 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 DATASET('FILEA')  
RIDFLD('009000') INTO(data-area)
```

can be entered on the command input line, as:

```
READ DAT(FILEA) RID(009000)
```

or at a minimum, as:

```
READ D(FILEA) R(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.

```

READ DATASET('FILEA')
STATUS: COMMAND SYNTAX CHECK
EXEC CICS READ
  Dataset('FILEA')
  < SYsid() >
  SEt() | Into()
  < Length() >
  RIDfld()
  < Keylength() < GEneric > >
  < RBa | RRn | DEBRec | DEBKey >
  < GTeq | Equal >
  < Update >

NAME=
```

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 7. "Command Syntax Check" 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 7), it indicates that the command which 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.

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.

```

READ DATASET('FILEA') RIDFLD('009000')
STATUS: ABOUT TO EXECUTE COMMAND
EXEC CICS READ
  DATASET('FILEA')
  < SYsid() >
  SEt() | Into()
  < Length() >
  RIDfld('009000')
  < Keylength() < GEneric > >
  < RBA | RRn | DEBRec | DEBKey >
  < GTeq | Equal >
  < Update >

```

NAME=

PF 1 HELP 2 HEX 3 END 4 EIB 5 VAR 6 USER 7 SBH 8 SFH 9 MSG 10 SB 11 SF

Figure 8. "About to Execute Command" Display

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 the description of a variable later in the chapter.)

About to Execute Command

This display (as shown in Figure 8) 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 DATASET('FILEA') RIDFLD('009000')
STATUS:  COMMAND EXECUTION COMPLETE
EXEC CICS READ
  Dataset( 'FILEA'  )
  < SYsid() >
  SEt() |
    Into( 'U009000I. COLLINGTON'          NAME=
    < Length( +00080 ) >
    RIDfld( '009000' )                  SURREY, ENGLAND    0987654321 ' ... )
    < Keylength() < GEneric > >
    < RBa | RRn | DEBRec | DEBKey >
    < GTeq | Equal >
    < Update >

| RESPONSE: NORMAL
| PF 1 HELP 2 HEX 3 END 4 EIB 5 VAR 6 USER 7 SBH 8 SFH 9 MSG 10 SB 11 SF
| EIBRESP=+0000000000

```

Figure 9. "Command Execution Complete" Display

Command Execution Complete

This display (as shown in Figure 9) 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 overttyping. 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 overttyping 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 413 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 which contained the commands which 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 which 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 that will be the end of the interpreter session.

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.

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.

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/DOS/VS Resource Definition (Online)* or *CICS/DOS/VS Resource Definition (Macro)* manual for details on constructing a PPT and a PCT.)

Chapter 1.9. Temporary Storage Browse

The Browse Transaction

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 10 on page 70. 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 CEBRname (where "name" 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. Before you enable the transaction, therefore, you should consider the possible consequences of using it. In particular, you should ensure that data cannot be browsed by unauthorized personnel. In other words, you should employ **resource level security**.

Using the Transaction

If you invoked the browse transaction from a terminal with the identifier L77A, you would receive the display shown in Figure 10 on page 70.

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, you should 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.

```

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 10. Initial Display Produced By the Browse Transaction

PURGE

Erase the contents of the queue being browsed.

If the queue is recoverable, you should 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

Show the first page of this queue.

BOTTOM

Show the last page of this queue.

FIND /string

Find 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

Make the specified line the second line on the displayed page.

COLUMN nnnn

Move the displayed area to this column of the queue.

GET xxxx

Transfer 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.

PUT xxxx

Transfer 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 you must 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, you should 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. You should code a DFHTST TYPE = SECURITY system macro for each queue that can be browsed.

Part 2. Files and Data Bases

	Chapter 2.1. General Description of File Control Facilities	75
	Chapter 2.2. File Control - VSAM Considerations	87
	Chapter 2.3. File Control - DAM Considerations	89
	Chapter 2.4. File Control - Commands, Options, and Conditions	93
	Chapter 2.5. DL/I Services (EXEC DLI Command)	105
	Chapter 2.6. DL/I Services (DL/I CALL Statement)	115

Introduction to Files and Data Bases

CICS transactions can access files and data bases, 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 DAM data sets. When you access files through the file control program, you do not have to consider such things as buffer management, blocking and deblocking, and access method dependencies. File control is described in Chapters 2.1 through 2.4.

DL/I data bases give you a greater degree of data independence than file control does. You are presented with a logical view of the data base 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 data bases are processed by the IBM licensed program DL/I DOS/VS, Program Number 5746-XX1.

CICS has two programming interfaces to DL/I. You are recommended to use what is called the "EXEC DLI interface", as it is simpler to use and can be used with EDF.

The other DL/I programming interface is known as 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 105.

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 115.

| 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.

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 in the form of individual records. 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 must identify 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)
- Direct Access Method (DAM).

The data sets handled by these access methods are called VSAM data sets and DAM 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** 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 browsing through the data set.

To enable VSAM to determine the physical location of a record in a KSDS, VSAM creates and maintains an index which 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** is one in which each record is identified by its relative byte address. 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** 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 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

An application program may find it convenient 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 key-sequenced data set 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 key-sequenced data set in which records are accessed using the alternate key.

When a record is updated by way of a path, the corresponding alternate index 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 it has been defined to VSAM when it is created to belong 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).

You should, however, be aware that 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 which can take on values 1, 2, 3, or 4, and which defines how the data set can 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. For any such data set, VSAM imposes a restriction that only a single FCT entry can be open for update at any one time. If an attempt is made to open a further FCT entry for update, the open will fail with an appropriate VSAM return code.

A data set defined with share options 3 or 4 can, on the other hand be updated concurrently through more than one FCT entry. For such data sets however, CICS is unable to ensure that integrity of data is preserved.

DAM Data Sets

CICS supports access to keyed and nonkeyed DAM data sets. DAM support makes use of the physical nature of a record on a DASD device. DAM data sets are made up of unblocked records with the following format:

Count	Physical (recorded) key	Data

Physical record

Keyed DAM files have a physical key identifying the DAM record. The count area contains the physical key length, the physical data length, and the record's data location. Nonkeyed DAM files have a zero key length in the count area of the physical record.

CICS may define a further structure on top of DAM data sets. It introduces the concept of blocked data sets:

Count	Physical key	Data
		logrec 1 logrec 2

Physical record
(block)

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 DAM concept of one logical record for every physical record.

To retrieve a physical record from a DAM file under CICS, a record identification field (RIDFLD) has to be defined 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, then 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 DAM files. The addressing mode for CICS DAM data sets is set in the FCT using the RELTYPE keyword.

A more detailed discussion of record identification and DAM record access is given later in "Chapter 2.3. File Control - DAM Considerations" on page 89.

Data Set Identification

You use the DATASET option in a file control command to specify the symbolic name of an entry in the FCT that identifies the data set to be accessed. The FCT entry must be defined in the CICS system (unless the data set resides in a remote system) and must have been associated with a physical data set before the command can be executed.

This association is done by including a job control statement for the data set in the CICS job. The FCT entry name is used as the filename on the DLBL statement and the 44-character physical data set name is specified as the file id on the DLBL statement. The data set will be allocated to CICS at the time of CICS job initiation and will remain allocated through to CICS termination.

Accessing Data Sets From CICS Application Programs

The following sections describe the facilities available to application programs for accessing data sets. The discussion is presented in terms of VSAM data sets, though most of the facilities apply equally to DAM data sets. Where there are differences, or particular considerations that are appropriate only to DAM, these are noted in the

text and in "Chapter 2.3. File Control - DAM Considerations."

Retrieving Records

Direct Reading

A record in the data set is read by means of the READ command. 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 an 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. 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 backwards 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 forwards through a data set. The INVREQ condition is raised if a READPREV is attempted during a browse initiated using a generic key.

The options "key equal to" and "key greater than or equal to" 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). In 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 sync point, 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 be performing 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 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. It must be noted however that 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 were 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, since the keylength has been changed. To continue browsing from this new point, remove the KEYLENGTH option from subsequent READNEXT commands.

Note also that if a "key equal to" search is specified on a STARTBR command, or a 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. Note however, that if more than one FCT entry refers to the same base data set, and this has been defined to VSAM with share options 1 or 2, only a single FCT entry may be

open at any time. Multiple concurrent updates to a data set through different FCT entries can be made only if the data set is defined with share options 3 or 4; for such data sets, CICS cannot ensure that data integrity is preserved.

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 data set 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 are 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, and they must always be written from an area provided by the application program.

Adding to a KSDS

When adding a record 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 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. Note that 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 sync point is issued, or at task termination.

Review of File Control Command Options

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 STARTBTR 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 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, it is possible to 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.

When you retrieve a record using the READ, READNEXT, or READPREV commands, the record is retrieved and placed in main storage according to which of the options INTO or SET 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.

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 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 DAM 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 since 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. 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 the process of updating a VSAM data set that uses shared resources (LSRPOOL not equal to NONE in the FCT). A READ UPDATE has been executed successfully for a record in the data set, but a REWRITE has not yet been performed. Before the REWRITE, the transaction issues a request to browse through the file starting at the same record, or at a record in the same control interval. Since 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.

In order 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 data set if a previous READ UPDATE has not yet been completed with a REWRITE, DELETE or UNLOCK command.

CICS does not, however, detect every situation which may cause a deadlock. The application programmer can avoid deadlocks by following these rules:

- 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.
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 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 DAM 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 in one of three ways:

- 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 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, it must be noted that 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 accessing a data set 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. The reason for this is that 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 is 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 sync point is automatically performed). For VSAM recoverable file processing, you must be aware of certain further considerations.

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 sync point.

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 since 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. Moreover, 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 - DAM Considerations

Record Identification

You identify records in DAM 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 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 MBBCCCHHR (RELTYPE operand omitted).

The type of block reference being used must be specified in the RELTYPE operand of the DFHFCT TYPE = DATASET system macro which defines the 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 = DATASET system macro that defines the 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 = DATASET 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 one-byte binary number (first record = 0).

The examples in Figure 11 on page 90 assume a physical key of four bytes and a deblocking argument of three bytes.

Browsing Records From DAM data sets

The record identification field must contain a block reference (for example, TTR or MBBCCCHHR) 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 MBBCCCHHR 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

Byte	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	T	T	R	PH-KEY		KEY										Search by relative track and record and key; deblock by key
	M	B	B	C	C	H	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 11. Examples of Record Identification

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.

It should be noted at this point that the options DEBREC and DEBKEY need to be specified on

the STARTBR command when browsing blocked data sets, in order 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. 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 DAM 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 DAM Data Sets

When adding records to a DAM 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.

An attempt to add undefined or variable-length records is limited to the single track specified. If insufficient space is available on that track, the NOSPACE condition occurs. However, an attempt may be made to add the record on another track simply by altering the track identifier and using another WRITE command. 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.
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 four-byte record description field (RDF) must be included in each record. The first two bytes specify the length of the record (including the 4-byte RDF); the other two bytes consist of zeros.

DAM 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.

Read a Record (READ)

```
READ
DATASET(name)
{INTO(data-area)|SET(ptr-ref)}
[LENGTH(data-area)]1
RIDFLD(data-area)
[KEYLENGTH(data-value)]2[GENERIC]3
[SYSID(name)]
[RBA3|RRN3|DEBKEY4|DEBREC4]
[GTEQ|EQUAL]3.
[UPDATE]
```

Conditions: DISABLED, DSIDERR, DUPKEY³, ILLLOGIC³, INVREQ, IOERR, ISCINVREQ, LENGERR, NOTAUTH, NOTFND, NOTOPEN, SYSIDERR

- ¹ Mandatory with SYSID, and with INTO when reading variable-length records
- ² Mandatory with SYSID unless RBA or RRN is coded also, in which case it is invalid
- ³ VSAM only
- ⁴ Blocked DAM only

You use this command to read a record 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 only specify 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 into a specified data area:

```
EXEC CICS READ
  INTO(RECORD)
  DATASET('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)
  DATASET('MASTVSAM')
  RIDFLD(ACCTNO)
  KEYLENGTH(4)
  GENERIC
  GTEQ
  UPDATE
```

Write a Record (WRITE)

```
WRITE
DATASET(name)
FROM(data-area)
[LENGTH(data-value)]1
RIDFLD(data-area)
[KEYLENGTH(data-value)]2
[SYSID(name)]
[RBA|RRN]3
[MASSINSERT]3
```

Conditions: DISABLED, DSIDERR, DUPREC, ILLOGIC³, INVREQ, IOERR, ISINVREQ, LENGERR, NOSPACE, NOTAUTH, NOTOPEN, SYSIDERR

- ¹ Mandatory with SYSID, and with FROM when writing variable-length records
- ² Mandatory with SYSID unless RBA or RRN is coded also, in which case it is invalid
- ³ VSAM only

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)
  DATASET('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
DATASET(name)
FROM(data-area)
[LENGTH(data-value)]1
[SYSID(name)]
```

Conditions: DISABLED, DSIDERR, DUPREC, ILLOGIC², INVREQ, IOERR, ISINVREQ, LENGERR, NOSPACE, NOTAUTH, NOTOPEN, SYSIDERR

- ¹ Mandatory with SYSID, and with FROM when rewriting variable length records
- ² VSAM only

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)
  DATASET('MASTER')
```

For VSAM data sets you must not change the key field in the record.

Delete a Record (DELETE) - VSAM Only

```
DELETE
DATASET(name)
[RIDFLD(data-area)]1
[KEYLENGTH(data-value)]2
[GENERIC [NUMREC(data-area)]]]
[SYSID(name)]
[RBA|RRN]
```

Conditions: DISABLED, DSIDERR, DUPKEY, ILLLOGIC, INVREQ, IOERR, ISINVREQ, NOTAUTH, NOTFND, NOTOPEN, SYSIDERR

- ¹ Mandatory with GENERIC
- ² Mandatory with SYSID unless RBA or RRN is coded also, in which case it is invalid

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
DATASET('MASTVSAM')
RIDFLD(ACCTNO)
KEYLENGTH(4)
GENERIC
NUMREC(NUMDEL)
```

Release Exclusive Control (UNLOCK)

```
UNLOCK
DATASET(name)
[SYSID(name)]
```

Conditions: DISABLED, DSIDERR, ILLLOGIC¹, IOERR, ISINVREQ, NOTAUTH, NOTOPEN, SYSIDERR

- ¹ VSAM only

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
DATASET(name)
RIDFLD(data-area)
[KEYLENGTH(data-value)]1 [GENERIC]2
[REQID(data-value)]
[SYSID(name)]
[RBA2 | RRN2 | DEBKEY3 | DEBREC3]
[GTEQ | EQUAL]2
```

Conditions: DISABLED, DSIDERR, ILLLOGIC², INVREQ, IOERR, ISINVREQ, NOTAUTH, NOTFND, NOTOPEN, SYSIDERR

- ¹ Mandatory with GENERIC or SYSID unless RBA or RRN is coded also, in which case it is invalid
- ² VSAM only
- ³ Blocked DAM only

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
DATASET(name)
{INTO(data-area)|SET(ptr-ref)}
[LENGTH(data-area)]1
RIDFLD(data-area)
[KEYLENGTH(data-value)]2
[REQID(data-value)]
[SYSID(name)]
[RBA|RRN]3
```

Conditions: DISABLED, DSIDERR, DUPKEY³, ENDFILE, ILLOGIC³, INVREQ, IOERR, ISCVREQ, LENGERR, NOTAUTH, NOTFND, NOTOPEN, SYSIDERR⁴

- ¹ Mandatory with SYSID, and with INTO when reading variable length records
- ² Mandatory with SYSID unless RBA or RRN is coded also, in which case it is invalid
- ³ VSAM only
- ⁴ Not raised following successful STARTBR to remote system if that system becomes unavailable. Abend AZI4 issued instead.

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
DATASET(name)
{INTO(data-area)|SET(ptr-ref)}
[LENGTH(data-area)]1
RIDFLD(data-area)
[KEYLENGTH(data-value)]2
[REQID(data-value)]
[SYSID(name)]
[RBA|RRN]
```

Conditions: DISABLED, DSIDERR, DUPKEY, ENDFILE, ILLOGIC, INVREQ, IOERR, ISCVREQ, LENGERR, NOTAUTH, NOTFND, NOTOPEN, SYSIDERR³

- ¹ Mandatory with SYSID, and with INTO when reading variable length records
- ² Mandatory with SYSID unless RBA or RRN is coded also, in which case it is invalid
- ³ Not raised following successful STARTBR to remote system if that system becomes unavailable. Abend AZI4 issued instead.

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
DATASET(name)
RIDFLD(data-area)
[KEYLENGTH(data-value)]1[GENERIC]
[REQID(data-value)]
[SYSID(name)]
[GTEQ|EQUAL]2
[RBA|RRN]2
```

Conditions: DISABLED, DSIDERR, ILLOGIC², INVREQ, IOERR, ISCINVREQ, NOTAUTH, NOTFND, NOTOPEN, SYSIDERR

¹ Mandatory with SYSID unless RBA or RRN coded also, in which case it is invalid

² VSAM only

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 be restarted.

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 DAM user the sort of skip sequential capability that is available to VSAM users through use of the READNEXT command.

End Browse (ENDBR)

```
ENDBR
DATASET(name)
[REQID(data-value)]
[SYSID(name)]
```

Conditions: DISABLED, DSIDERR, ILLOGIC¹, INVREQ, ISCINVREQ, NOTAUTH, NOTOPEN, SYSIDERR

¹ VSAM only

You use this command to end a browse on a data set on a local or remote system.

You should 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

DATASET(name)

specifies the symbolic name of the data set to be accessed. The name must be alphanumeric, up to 7 characters in length, and must have been defined in the file control table (FCT).

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 FCT. Otherwise, the FCT entry will be used to determine if the data set is on a local or remote system.

DEBKEY (blocked DAM 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 DAM 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.

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. Note that 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 86.

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 and PL/I 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 DAM 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.

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 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 data set 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 data set is disabled. A data set 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

occurs if a data set name referred to in the DATASET option cannot be found in the FCT.

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, 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.

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 (refer to Appendix A, "EXEC Interface Block" on page 413 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 data set entry specification in the FCT.
- A REWRITE command, or a DELETE command without the RIDFLD option, is issued for a data set for which no previous READ UPDATE command has been issued.
- A READNEXT, READPREV, ENDBR, or RESETBR command is issued for a data set for which no previous STARTBR command has been issued.
- A READPREV command is issued for a data set 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.

- 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 DAM data set, or for a VSAM ESDS data set.
- A DELETE command with the RIDFLD option specified is issued for a VSAM data set when a READ UPDATE command is outstanding.
- Following a READ UPDATE command for a data set, a WRITE or READ UPDATE command is issued for 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 (refer to Appendix A, "EXEC Interface Block" on page 413 for details).

Default action: terminate the task abnormally.

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.

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 data set is CLOSED and UNENABLED. The CLOSED, UNENABLED state is reached after a close request has been received against an OPEN ENABLED data set and the data set is no longer in use. This state can also be specified as the initial state by means of the FILSTAT parameter of the DFHFACT TYPE = DATASET system macro.
- The requested data set is still open and in use by other requests, but a close request against the data set 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 data set or a CLOSED, DISABLED data set. In the first case, the data set is opened as part of

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/DOS/VS Messages and Codes* manual.

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 data set 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)

This chapter outlines the EXEC DLI commands that can be used in command level application programs that are used to access DL/I data bases. These application programs can be written only in COBOL or PL/I.

Execution of the application programs requires the installation of the DL/I DOS/VS licensed program (program number 5746-XX1).

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 *DL/I DOS/VS Application Programming: High Level Programming Interfaces*.

For online application programs, the commands are translated by the appropriate command language translator (see "Chapter 1.3. Command Language Translator" on page 11) 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 MPS (multiple partition support) application programs, the DL/I link edit stub is invoked.

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 115. 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 49.

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 data base is as follows:

1. Schedule your access to the data base. That is, tell the system that you want access, and define the kind of access.
2. Perform operations involving the data base. 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	MPS	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
LOAD	NO	NO	YES
CHECKPOINT	YES	YES	YES
SCHEDULE	YES	NO	NO
TERMINATE	YES	NO	NO

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, you should refer to the publication *DL/I DOS/VS Application Programming: High Level Programming Interfaces*.

Schedule the PSB

```
SCHD
PSB(name)
```

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]
 [SEGMENT(name)]
 [SEGLENGTH(integer-expr)]
 [OFFSET(integer-expr)]
 [LOCKED]
 [INTO(data-area)]
 [WHERE(where clause)]
 [FIELDLENGTH(integer-expr)])

For the object segment:
 [VARIABLE]
 [FIRST|LAST]
 [SEGMENT(name)]
 [SEGLENGTH(integer-expr)]
 [OFFSET(integer-expr)]
 [LOCKED]
 INTO(data-area)
 [WHERE(where clause)]
 [FIELDLENGTH(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]
 [SEGMENT(name)]
 [SEGLENGTH(integer-expr)]
 [FROM(data-area)]
 [WHERE(where clause)]
 [FIELDLENGTH(integer-expr)])

For the object segment:
 [VARIABLE]
 [FIRST|LAST]
 [SEGMENT(name)]
 [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)]
[FROM(data-area)]
[SEGLENGTH(integer-expr)]
```

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).

Replace One or More Segments

```
REPL
[USING PCB(integer-expr)]
For each parent segment:
[VARIABLE]
[SEGMENT(name)]
[SEGLENGTH(integer-expr)]
[OFFSET(integer-expr)]
[FROM(data-area)]
For the object segment:
[VARIABLE]
[SEGMENT(name)]
[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)]
[FROM(data-area)]
[SEGLENGTH(integer-expr)]
```

The full form of the command is DELETE.

Terminate Access to the PSB

```
TERM
```

The full form of the command is TERMINATE

Request a Basic Checkpoint

```
CHKP
ID(data-area) | ('char-expr')
```

The full form of the command is CHECKPOINT.

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 may be coded in any order.

Similarly, you must code keywords which 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.

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 data base description (the DBD).

"value" is a reference to a data area containing the value to be compared.

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);
```

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	COBOL	PL/I
DIBSTAT	PIC XX	CHAR(2)
DIBSEGM	PIC X(8)	CHAR(8)
DIBFLAG	PIC XX	CHAR(2)
DIBSEGLV	PIC XX	CHAR(2)
DIBKFBL	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

GE segment not found

GK different segment type at same level returned

II segment to insert already exists

LB segment to load already exists (batch only)

NE index segment not found

TG TERM attempted when PSB not scheduled

A full list of status codes is given in the application programming reference manual for DL/I VSE.

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.

DIBFLAG

is a flag indicating that an online task had to wait for a resource owned by an MPS batch task. The value is either HIGH-VALUE in COBOL, HIGH(1) in PL/I or LOW-VALUE in COBOL, LOW(1) in PL/I.

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 COBOL and PL/I, the use of the EXEC DLI command in a CICS application program to request DL/I services.

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).
          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 SCHED PSB(SAMPLE1) END-EXEC.
*  PERFORM TEST-DIB THRU OK.
* *****
*  RETRIEVE ROOT SEGMENT AND ALL ITS DEPENDANTS
* *****
*  MOVE 'A300' TO SEGKEYA.
```

COBOL Example of EXEC DLI Commands (Continued)

```
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.

* *****
* RETRIEVE SEGMENT DIRECTLY USING CONCATENATED KEY
* *****
* AND THEN DELETE IT AND ITS DEPENDANTS
* *****
  MOVE 'A200B230' TO CONKEYB.
  EXEC DLI GU USING PCB(1)
```

COBOL Example of EXEC DLI Commands (Continued)

```
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 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
  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  AREA1             CHAR (80);
/* DEFINE SEGMENT I/O AREA */  
DCL 1  AREAB             CHAR (80);
DCL 1  AREAC             CHAR (80);
DCL 1  AREAG             CHAR (250);
/*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(AREA1)
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 */  
/* ***** */  
AREAA = 'DATA FOR NEW SEGMENT INCLUDING KEY';
EXEC DLI ISRT USING PCB(1) SEGMENT(SEGA) FROM(AREA1);
CALL TEST_DIB;
/* ***** */  
/* RETRIEVE 3 SEGMENTS IN PATH AND REPLACE THEM */  
/* ***** */  
SEGKEYA = 'A200';
```

PL/I Example of EXEC DLI Commands (Continued)

```
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';
SEGNAM = 'SEGA';
SEGKEYC = 'C520';
PCBNUM = 1;
EXEC DLI GU USING PCB(PCBNUM)
  SEGMENT((SEGNAM))
    KEYS(CONKEYB) SETPARENT
  SEGMENT(SEGC) INTO(AREAC)
    WHERE(KEYC=SEGKEYC);
CALL TEST_DIB;
/* ***** */
/* RETRIEVE ROOT SEGMENT USING BOOLEAN OPERATORS */
/* ***** */
SEGKEY1 = 'A050';
```

PL/I Example of EXEC DLI Commands (Continued)

```
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 */
/* **** 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)

DL/I is a general-purpose data base control system that executes in a virtual-storage environment under VSE. DL/I simplifies the creation of data bases by CICS application programs. It also simplifies the subsequent maintenance of those data bases by CICS application programs. The DL/I licensed program (program number 5746-XX1) is used, running as part of the CICS partition.

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 data base CALL statements running in batch mode. (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 data bases.

| This chapter describes the DL/I CALL statement method of accessing DL/I data bases 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 105.

The two methods of accessing DL/I data bases 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.

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-reentrant, 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 data base 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 data base access requirements of each DL/I data base to be accessed by the application program.

A CICS application program designed to access DL/I data bases 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 data bases 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 data base to be accessed.

If the scheduling call is unsuccessful, an INVREQ 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 405) 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 data base is required, the PSB must be rescheduled. The previous position of the data base has been lost.

To access DL/I data bases, you should:

1. Issue a DL/I call to schedule the PSB and obtain PCB addresses.
2. Issue a DL/I call to access the data base.
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 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) POINTER COMP	
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 119).

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 7 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.

For CICS/DOS/VS this default is the first PSB name associated with the application program in the DL/I DOS/VS application control table (the ACT) as defined during DL/I DOS/VS system generation.

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 data base. 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.

For information on how to build an SSA, refer to the publication *DL/I DOS/VS Application Programming: CALL and RQDLI Interfaces*.

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 data base. 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 Data Base Call

The format of the CALL statement to request DL/I services is as follows:

ASM:

```
CALLDLI ASMTDLI[,([parmcount,]function  
,pcb,workarea[,ss1,ss2,...])]
```

COBOL:

```
CALL 'CBLTDLI' USING [parmcount,]  
function,pcb,workarea[,ss1,ss2,...]
```

PL/I:

```
CALL PLITDLI ([parmcount,]function  
,pcb,workarea[,ss1,ss2,...])
```

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:

“CHKP”

request that a checkpoint be issued.

“GU”

get a unique segment identified by SSAs.

“GN”

get the next segment in the data base, optionally qualified by SSAs.

“GNP”

get the next segment within the scope of the current hierarchy in the data base, 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.

“GHPN”

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 data base.

“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 data base 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.

“ss1,ss2,...”

are the names of the segment search arguments.

For details of these calls, refer to the publication

DL/I DOS/VS Application Programming: CALL and RQDLI Interfaces

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 data base 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 405.) 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 data base is required, because the position of the data base 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

You should 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

In the above table, NORESP means "normal response".

For the two responses INVREQ (invalid request) and NOTOPEN (not open), 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
TASKNA	X'02'	12-2-9	00000010
PSBSCH	X'03'	12-3-9	00000011
LANGCON	X'04'	12-4-9	00000100
PSBFAIL	X'05'	12-5-9	00000101
PSBNA	X'06'	12-6-9	00000110
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 condition names in the above table have the following meanings:

PSBNF - PSB not found

TASKNA - task not authorized

PSBSCH - PSB scheduled

LANGCON - language conflict

PSBFAIL - PSB initialization failed

PSBNA - PSB in scheduling call not defined in program application control table entry, is too long, or not delimited by a blank

TERMNS - terminal unscheduled

FUNCNS - function unscheduled

MPS batch - MPS batch program issues PCB call for read-only PSB or for 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
Data base stopped	X'01'	12-1-9	00000001
Intent scheduling conflict	X'02'	12-2-9	00000010

If fields UIBFCTR and UIBDLTR are normal, you should examine the DL/I status codes in the PCB. These status codes are listed in the publication *DL/I DOS/VS Application Programming: CALL and RQDLI Interfaces*.

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 "Dnnn", where "nnn" is the DL/I pseudoabend code.

Examples of DL/I Requests

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 OCL40
AREA1 DS CL3
AREA2 DS CL37
DLIUIIB
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 OCL256
DBPC1NM DS OCL12
DBPC1NMA DS OCL14
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 OH
* 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,.....
* ISSUE DL/I CALL: REPLACE SEGMENT AT CURRENT POSITION
CALLDLI ASMTDLI,(REPLFUN,PCB1,IOAREA,SSA1)
CLC UIBFCTR,GOODRC
BNE ERROR4
```

ASM Example of DL/I Call (Continued)

```
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  
     MOVE ..... TO AREA1.  
     MOVE ..... TO AREA2.
```

COBOL Example of DL/I Call (Continued)

```
* 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 116.*
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

```
PLIUIIB: 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) STATIC 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 DLIUIIB;
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_DBDB_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_SEGNAMES_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 FSB 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;
  ELSE IF PCB1_STATUS_CODE~=GOOD_STATUS_CODE THEN DO;
    /* EXAMINE PCB1_STATUS_CODE */
    .
  END;
END;
```

PL/I Example of DL/I Call (Continued)

```
/* 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 116.*
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.*

Part 3. Data Communication Operations

Chapter 3.1. Introduction to Data Communication Operations	129
Chapter 3.2-1. Introduction to Basic Mapping Support	131
Chapter 3.2-2. Minimum Function BMS	135
Chapter 3.2-3. Standard Function BMS	161
Chapter 3.2-4. Full Function BMS	181
Chapter 3.2-5. BMS Macro and Command Reference Summary	205
Chapter 3.3. Terminal Control	239
Chapter 3.4. Report Controller	293
Chapter 3.5. Batch Data Interchange	323
Chapter 3.6. Structured Query Language/Data System (SQL/DS)	331



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.

You are recommended to refer to the *CICS/DOS/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 239.

The report controller provides commands and options that may be used, in conjunction with POWER, to provide an integrated spooling facility. The report controller is described in "Chapter 3.4. Report Controller" on page 293.

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 3.5. Batch Data Interchange" on page 323.

Structured Query Language (SQL/DS) is supported by CICS. SQL/DS is described in "Chapter 3.6. Structured Query Language/Data System (SQL/DS)" on page 331.

Chapter 3.2-1. Introduction to Basic Mapping Support

Basic mapping support (BMS) is an interface between CICS and its application programs. There are three versions of BMS, as follows:

- 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 135.

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/DOS/VS Resource Definition (Online)* manual. Alternatively, you can use the DFHPPT system macro, as described in *CICS/DOS/VS 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 BMS**, **standard BMS**, and **full BMS**.

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 BMS version or the standard BMS 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

Appendices D, E, and F contain sample programs that illustrate, among other things, various aspects of programming for BMS. You should refer to these programs as you study new topics. Note, however, that 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. Refer to the *CICS/DOS/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 haven't 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 *Operator's Guide for the IBM 3270 Information Display System*, 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 MIDT) 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. You should note, however, that 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 156. 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/DOS/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 136) 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

As mentioned above, the 3270 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, DFHIMDI, 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 205. However, using minimum BMS you will not use all of the operands.

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]}]
  [,HIGHLIGHT={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 207.

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/DOS/VS Installation and Operations 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, HIGHLIGHT, 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 145.

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 205.

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, HIGHLIGHT, 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]]
  [,HIGHLIGHT={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. You should 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, you should 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 154.

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, HIGHLIGHT, 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 205.

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}]
  [,HIGHLIGHT={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 207. 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

Appendices 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/DOS/VS Installation and Operations 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 core image library.

When you catalog the physical map, you should consider whether to add a suffix to its name (specified with the PHASE statement). The reason for suffixing a map is that you might wish 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 12 on page 146 summarizes the map set selection logic used by BMS. Minimum BMS supports only 3270 terminals. There are only two TERM values that have meaning under minimum BMS. They are 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, you should ask your system programmer to ensure that your physical maps are cataloged with the correct suffixes. In particular, you should note the following points about suffixing:

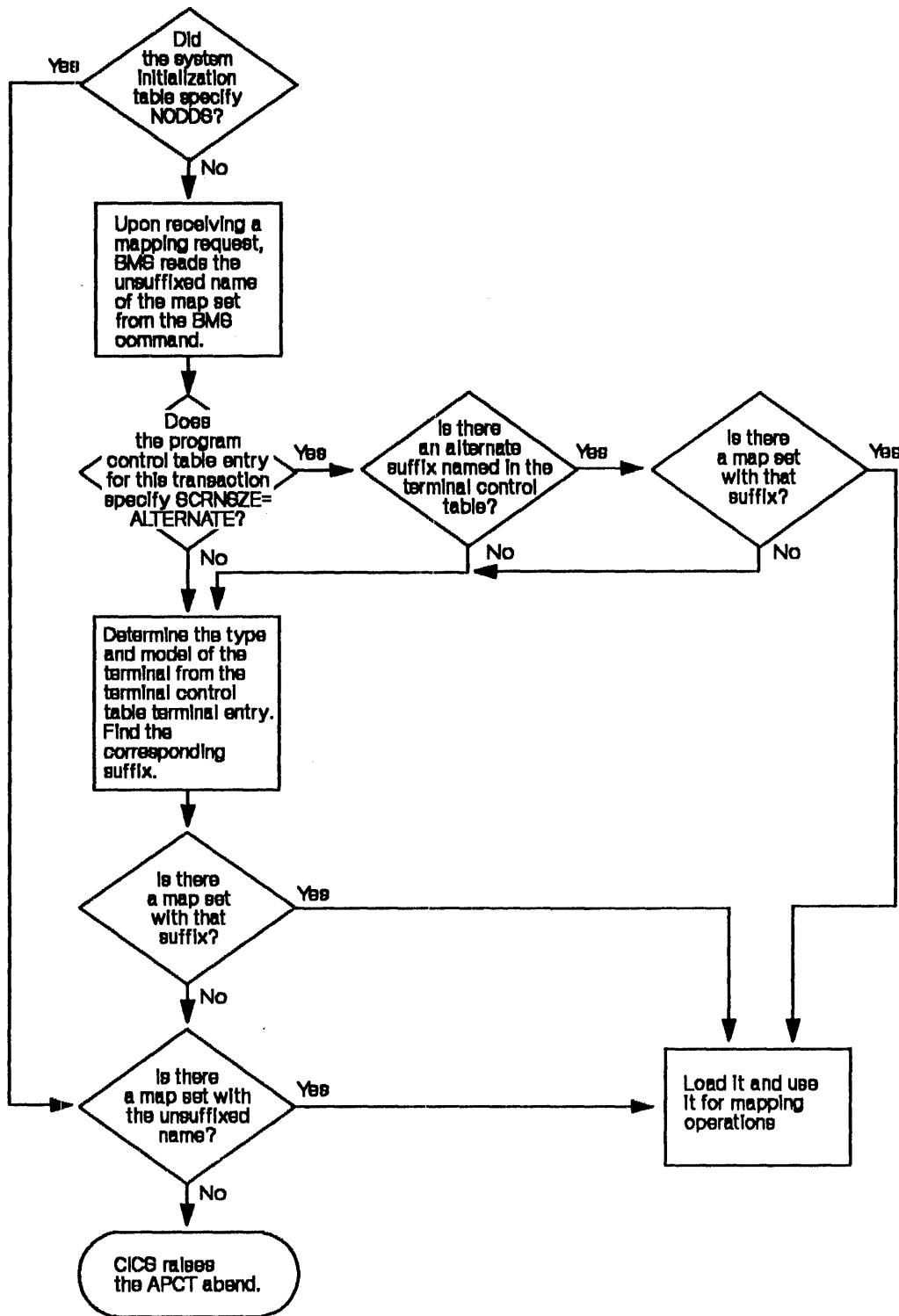


Figure 12. BMS Map Set Suffixing Logic

1. If you specify TERM or SUFFIX on your DFHMSD macro, you should 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 12 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 140, 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.

Appendices 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 145. The following examples assume that the source library has been assigned to SYSLIB by suitable job control language as described in the *CICS/DOS/VS Installation and Operations 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 145.

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 beginning and end of the data structure are labeled automatically with the map name extended by the suffixes "S" and "E" respectively. This E suffix is useful if 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 157.
- 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.

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 DFHMD 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/DOS/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 MAP0(MAPE-MAPO),MAPO1
COBOL    MOVE LOW-VALUES TO MAPO
PL/I      DCL STR BASED CHAR(32767);
.
.
.
SUBSTR(ADDR(MAPO)->STR,1,
STG(MAPO))=LOW(STG(MAPO))

```

¹Data 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 222.

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 ACCOUNT0,CUSTNO..... (1)
          MVC ACCOUNTC,DFHBLUE..... (2)
          MVC ACCOUNTH,DFHBLINK..... (3)

COBOL    MOVE CUSTNO TO ACCOUNT0.... (1)
          MOVE DFHBLUE TO ACCOUNTC... (2)
          MOVE DFHBLINK TO ACCOUNTH.. (3)

PL/I      ACCOUNT0=CUSTNO;..... (1)
          ACCOUNTC=DFHBLUE;..... (2)
          ACCOUNTH=DFHBLINK;..... (3)

```

Additional installation defined named attribute constants can be created and catalogued 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/DOS/VS 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
[devctr1...]

```

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 153) 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 142), 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.

Remember that however you acquire storage you should 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), you should 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 449

If you don't 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. You should 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 don't 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 207.

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 don't 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" on page 25 and the coding example in the description of the **BASE** option under "Map Definition Macro Operand Summary" on page 207.

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 data base.

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.

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."

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. Note that 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 DFHIMSD macro.
2. Set the length 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 229. 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 143.) 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 **EIBCPSON**. 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. Note that 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 213.

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, you should 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**. Note, however, that **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 239. The RECEIVE PARTN command is described on page 173.

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 31.

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 224.

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 31. 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 236.

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 32 for an explanation of the **ERROR** condition.

You should remember that 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.

You can also get reports printed by using spooler commands, as described in "Chapter 3.4. Report Controller" on page 293.

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 appropriate *Resource Definition* manual.

For more information, see the *CICS/DOS/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/DOS/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. Refer to "Printer Support" on page 163. 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 342) 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).

Note however, that output from several instances of your transaction may be interleaved on the transient data queue. This can be avoided if all the data to be printed by an instance of your transaction is stored in a single transient data queue item. 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 doesn't 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 135. 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 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 179) This is an alternative to field data format, but you are recommended not to use it.

Text Processing

You use the SEND TEXT command to send text to a terminal.

The syntax of the SEND TEXT command under standard BMS is as follows:

```
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

The data area containing the text to be sent is specified in the FROM option. The LENGTH option specifies the length of this area. The text may contain embedded new line characters (X'15'), embedded blanks (X'40'), and embedded character attribute controls. Character attribute controls are discussed below.

When formatting the text, BMS splits it into lines of length less than or equal to the terminal page width. The terminal page width is specified by the PGESIZE or ALTPGE operands of the DFHTCT TYPE=TERMINAL system macro, or by the PAGESIZE or ALTPAGE attributes of the TYPETERM definition. BMS pads the ends of lines with blanks rather than splitting words. BMS starts each line with a single blank corresponding to the 3270 attribute byte. On a 3270 the attribute byte is set to unprotected and normal intensity.

If a line of text ends with a character, however, and the next character is a blank, BMS will process the data as if it were a sentence, that is, the blank will be removed and the next character positioned in the first column of the next line.

Where a line of text ends with a blank and the next character is also a blank, BMS will honor all blanks to process the data as if it were in table format.

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 blanks (X'40') embedded in the text are honored.

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 226.

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.

You should note that 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, you should 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 159.

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, you should 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/DOS/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. You should 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 (note that 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

You should 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.

Remember that 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 192, you should note that BMS precedes each block of text (that is, the text specified in a single SEND TEXT command) by a single blank attribute byte. Single words should not be split 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 should embed new line characters in your text data, and insure that your TCT page width setting is at least 121 characters (subject to the discussion on printer line widths in "3270 Printers Without the NLEOM Option" on page 163).

Printers and Device Independence

You should 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/VS library CICS170.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.

It is important to note that 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/DOS/VS 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 12 on page 162.

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 173. 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 174.

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.

Remember that 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 12 on page 162.

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 174. 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 12 on page 162. 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 forty; 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 remember that 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 you should 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 DFHIPPT 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 outbound 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 175.

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, LENGTHERR

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; you should 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 should 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 179) 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 205.

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/DOS/VS 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/DOS/VS 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 181.

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 205.

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 41.

Supplied Constants

A selection of MSR control bit patterns has been created for CICS and stored in the copy book DFHMSRCA. Figure 13 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.
DFHMSIPIY	IN PROCESS mode set. Yellow on.
DFHMSIPN	IN PROCESS mode 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.

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

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 1 2 3 4	USER LOCK AUTO Ai1S Ai2S	User mode. Turn the yellow light on if the same bit is on in STATE VALUE. Locked/Unlocked. If locked, MSR input is inhibited. 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. Suppress audible alarm 1 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 1 2 3 4		Light 1 (Green) Light 2 (Yellow) Light 3 (Red) Audible Alarm 1 (Long buzz) Audible Alarm 2 (Short Buzz)
4 (INDICATOR VALUE) Performs similar function to STATE VALUE.			

Figure 13. MSR Control Byte Values

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 205.

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/DOS/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 181, 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 DFHIMDI 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.

- Report controller. Full BMS is required when using the CICS report controller (a separately-priced feature.)

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 should 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 terminal 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 192.

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 405. 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

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 192.

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 184.

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 161.

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 201. 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 182.

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 183.

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 185. 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 207.

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/DOS/VS 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 DFHISIT 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 190. 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.

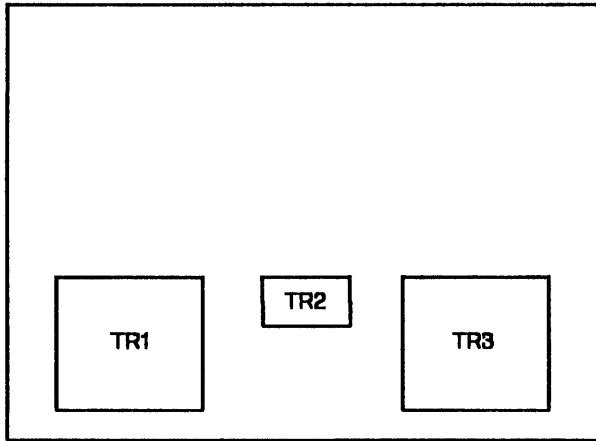
Floating Maps, Header and Trailer Maps

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 190.

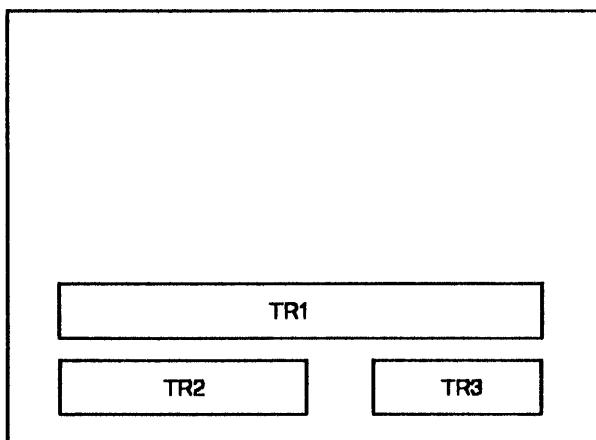
Avoid using floating maps, header maps, or trailer maps in a RECEIVE MAP command. This is because the floating map will be positioned on an empty page and the meanings of the LINE, COLUMN, and JUSTIFY operands of the DFHMDI macro are modified as explained in "Map Definition Macro Operand Summary" on page 207.

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.

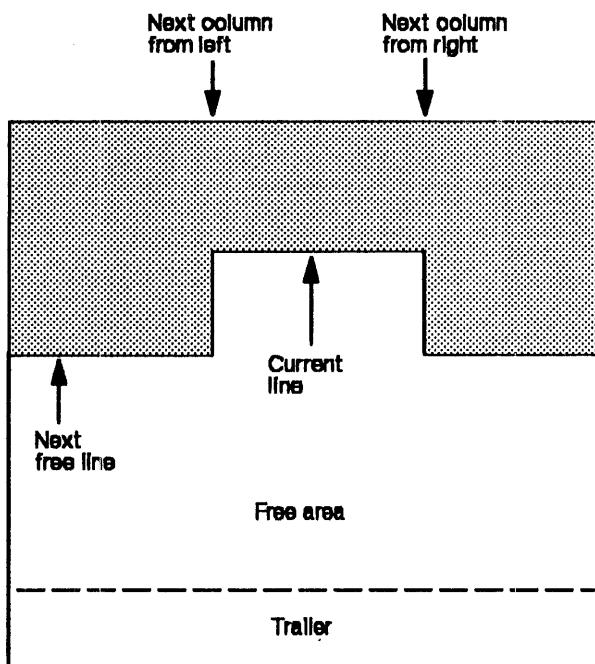
If a header map is not used, JUSTIFY = FIRST must be specified for the first map used to start a new page. Failure to specify this will result in a further page being sent for each SEND MAP command (as each map will be placed at the bottom of a page, causing page overflow).

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.

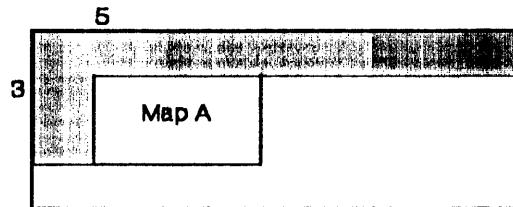
In processing a BMS command, BMS determines whether the area of the page required by the map is wholly available or whether any part of it has been used by an earlier command. "Used" means actually filled by a map or rendered unavailable.

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,...
```

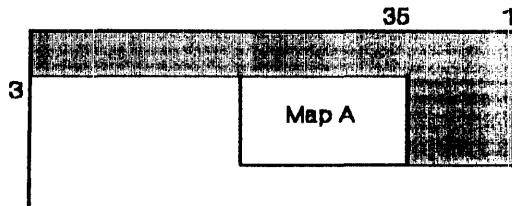
all columns to the left of the leftmost map column, for the full depth of the map, are unavailable, as shown (by the cross hatching) in the following diagram:



When JUSTIFY= RIGHT is coded, as in the following definition:

```
MAPA DFHMDI ...,LINE=3,COLUMN=35,  
JUSTIFY=RIGHT,...
```

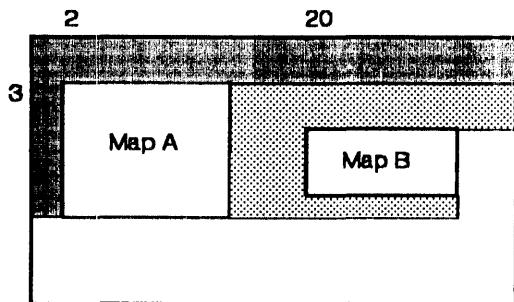
all columns to the right of the rightmost map column, for the full depth of the map, are unavailable, as shown in the following diagram:



When two or more maps are placed so that they share certain lines, as in the following definitions:

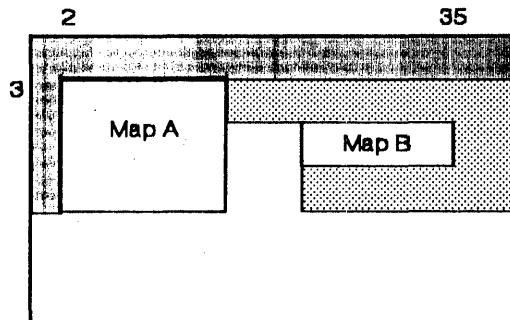
```
MAPA DFHMDI  . . . ,LINE=3,COLUMN=2,
                JUSTIFY=LEFT, . . .
MAPB DFHMDI  . . . ,LINE=4,COLUMN=20,
                JUSTIFY=LEFT, . . .
```

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:

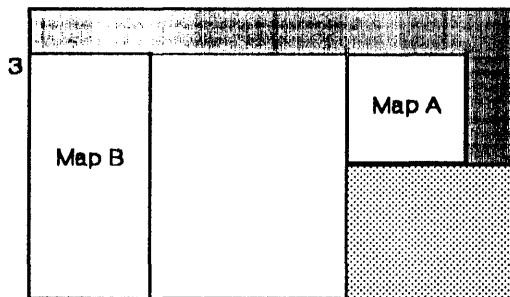


Similarly unavailable are all columns to the left (if the higher map is left justified) or to the right (if the higher map is right justified) of the "used" area beneath the higher map. The following two diagrams illustrate similar situations:

```
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 14 on page 191.

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.

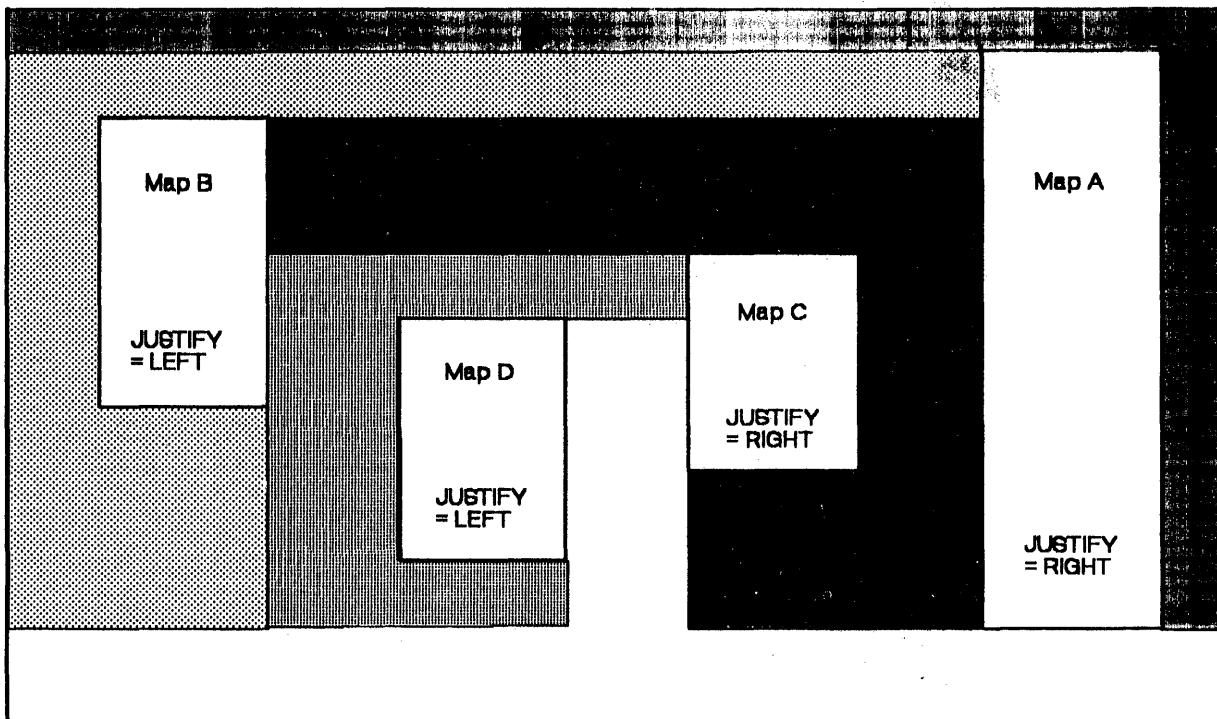


Figure 14. Many BMS Maps on One Page

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 201.

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 15.

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.

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 PI/I version of the program for this transaction is shown in Figure 16 on page 194.

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.

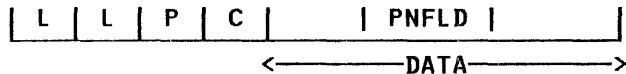
You can use the HEADER and TRAILER options to specify data to be placed at the top and bottom of each page. As a page boundary can occur as a result of any SEND TEXT ACCUM command, the HEADER and TRAILER options should be repeated on each SEND TEXT ACCUM command in the BMS logical message. Automatic 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.

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 201).

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:



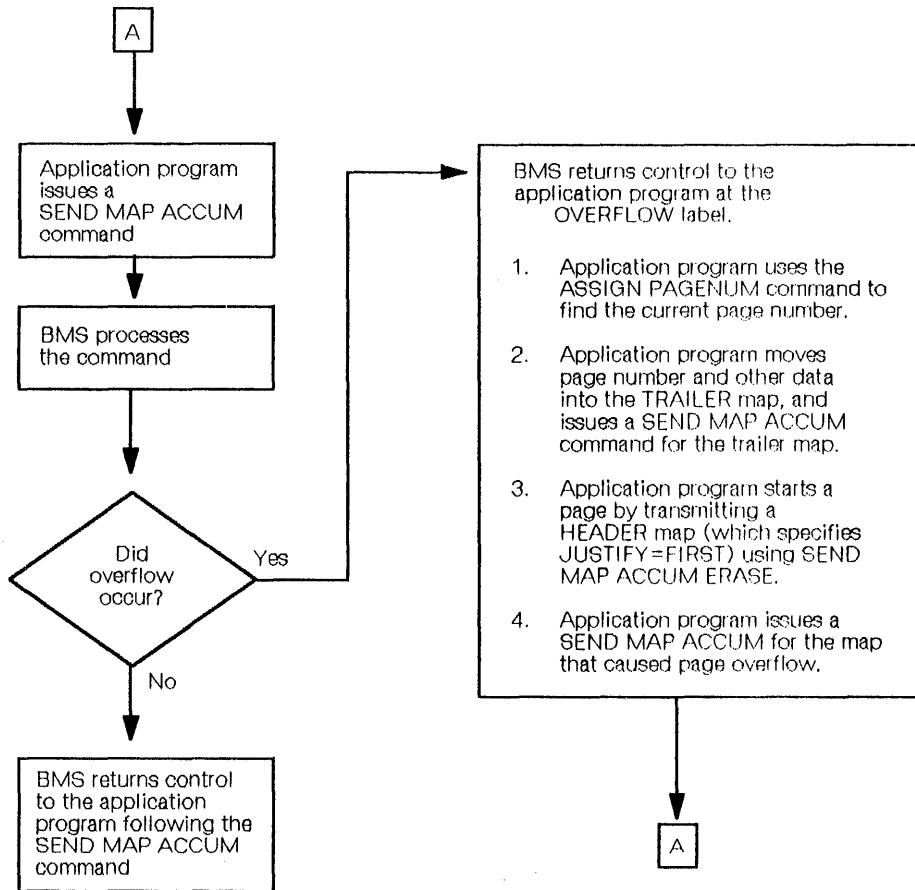


Figure 15. Overflow Processing

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.

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 cause 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('0'),
 2 HEADPAD CHAR(1),
 2 HEAD CHAR(14) INIT('PAGE NO. 00');
```

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.

```
/* 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 16. Example of Paging

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 201.

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/DOS/VS 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, RTE SOME

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 or not 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 198.

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 DFHISIT 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=ENTR Y) for the transaction issuing the ROUTE command. (See the *CICS/DOS/VS Resource Definition (Online)* manual, or the *CICS/DOS/VS 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 by CICS whenever a message becomes undeliverable. The error message will be sent to the terminal associated with the task that is sending the message; alternatively, the application program can specify a different terminal to receive such error messages by using the ERRTERM option. In addition to sending an error message, CICS lets the master terminal operator know how many undeliverable messages have been deleted for a 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 201.

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/DOS/VS 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 length field	24-byte title 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. Since 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 DESCOUNT 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. Note, however, that it is not necessary to use overflow control areas.

2. The ASSIGN DESCOUNT 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 (CMMSG)

CICS provides a message switching transaction (CMMSG), 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/DOS/VS 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, as 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. (Note that pages may be built for multiple terminal types by a single BMS command if routing is in effect.)

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 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, refer to Appendix A, "EXEC Interface Block" on page 413. 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 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, TSI0ERR
```

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 (PGA)" 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 PGA 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,
INREQ, RETPAGE, TSI0ERR
```

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.

You should 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}]
  [,HIGHLIGHT={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])]
[,OFBMT={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 numerical 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. Refer to the publication *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 135 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=DATAAREA1, MODE=IN
```

```
MAPSET2 DFHMSD TYPE=DSECT,  
TERM=3270, LANG=COBOL,  
BASE=DATAAREA1, MODE=OUT
```

The symbolic description maps of this example might be referred to in a COBOL application program as follows:

```
LINKAGE SECTION.  
01 DFHBLDS COPY DFHBLDS.  
. . .  
02 TIOABAR PIC S9(8) COMP.  
02 MAPBASE1 PIC S9(8) COMP.  
. . .  
01 DFHTIOA COPY DFHTIOA.  
01 DATAAREA1 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 DATAAREA1; 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 = DATAAREA1** is deleted from this example, an additional 02 statement is needed to establish addressability for MAPSET2; the 01 DATAAREA1 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.

Refer to "Map Positioning" on page 187 for a detailed discussion 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 158.

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 222).

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 135 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 161. 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, HIGHLIGHT, 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,HIGHLIGHT,PS,VALIDN)
DSATTS=(COLOR,HIGHLIGHT,PS,VALIDN)

MAPONLY is equivalent to:

MAPATTS=(COLOR,HIGHLIGHT,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 144.

HEADER

allows the map to be used during page building without terminating the overflow condition (see "Floating Maps, Header and Trailer Maps" on page 186 for further details). This operand may be specified for more than one map in a map set.

HIGHLIGHT

specifies the default highlighting attribute for all fields in all maps in a map set. This is overridden by the HIGHLIGHT operand of the DFHMDI, which is in turn overridden by the HIGHLIGHT 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 HIGHLIGHT is specified when EXTATT = NO, a warning is issued and the option ignored. If HIGHLIGHT is specified, but EXTATT is not, EXTATT = MAPONLY will be assumed.

The HIGHLIGHT 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, it specifies the position of the map on the page. On DFHMDF, it specifies the position of a field within a BMS map.

Here is what 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

$LINE = (\text{screeendepth} - \text{mapdepth} + 1)$

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. Refer to "Map Positioning" on page 187 for a more detailed discussion.

JUSTIFY on DFHMDF specifies the field justifications for input operations. This operand is ignored for VTAM-supported 3600, 3650, and 3790 terminals, because input mapping is not available.

Here is what 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 175.

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.

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.

Refer to "Map Positioning" on page 187 for a detailed discussion 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, HIGHLIGHT, OUTLINE PS, SOSI, TRANSP, and VALIDN. This list must include all the attribute types to be specified for individual fields in the map (DFHMDF macro).

Where possible 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 161.

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 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 F10    PIC X(30).
  02 FILLER PIC X.
  02 FILLER PIC X(3).
  02 F20    PIC $$,$$0.00.
  02 FILLER PIC X.
  02 FILLER PIC X(3).
  02 F30    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 (

Refer to 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 (

Refer to 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 145.

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 following table.

In addition, you should 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

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
3270 ³ blank	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.

more information, see the *CICS/DOS/VS Resource Definition (Macro)* manual.

TIOAPFX

specifies whether BMS should include a filler in the symbolic description maps to allow for the unused TIOA prefix.

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 186). 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 should 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 catalogued 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/DOS/VS Installation and Operations 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 161.

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 partset 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 174.

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 145.

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 you should 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 below. Combinations other than shown must be generated

separately; a bit map to help you do this is given on page 224.

The value of an attribute constant can be determined by referring to the publication *An Introduction to the IBM 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 DFHDFDT 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, DFHIBASE, or DFHDFHI should be used.

Standard List DFHBMSCA

Constant	Meaning
DFHBMPEM	Printer end-of-message
DFHBMPNL	Printer new-line
DFHBMASK	Autoskip
DFHBMPNP	Unprotected
DFHBMMUNN	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
DFHDFDT	Default
DFHDFCOL ¹	Default color
DFHBLUE	Blue
DFHRED	Red
DFHPINK	Pink

Standard List DFHBMSCA (Continued)

Constant	Meaning
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
DFHMFT	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
DFHDFR	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

¹ For text processing only. Use for constructing embedded set attribute orders in user text

² Cannot be used in set attribute orders

| The table on page 224 contains a bit map for attributes other than those listed in copy book DFHBMSCA.

| 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 table mean the following:

| H = high
| N = numeric
| P = protected
| S = automatic skip
| U = unprotected
| Y = yes

Bit Map for Attributes Other than Listed in DFHBMSCA

prot	a/n	hi	spd	ndp	mdt	ebcd	asci	char
U					Y	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					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						61	2F	/
P			Y			E4	55	U
P			Y			E5	56	V
P		H	Y			E8	59	Y
P		H	Y			E9	5A	Z
P				Y		6C	25	%
P	S				Y	6D	5F	_ (underscore)
P	S					F0	30	0
P	S					F1	31	1
P	S		Y			F4	34	4
P	S		Y			F5	35	5
P	S	H	Y			F8	38	8
P	S	H	Y			F9	39	9
P	S			Y		7C	40	ä
P	S			Y	Y	7D	27	,

Attention Identifier Constants (DFHAID)

The constants supplied in copy book DFHAID are as follows:

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

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 DFIIMSRCA.

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:

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.
DFHMSFIN	IN PROCESS (user) mode set. 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. 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.

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 154. The INPARTN option is fully described in "Setting the Expected Input Partition" on page 173.

Data from certain logical units is not mapped, but is left unaltered. Refer to 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 only available 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, LENGTHERR

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 173

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 only available 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 172.

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]
[LDC(name)]

Full BMS

[ACCUM]
[SET(ptr-ref)|PAGING]
[TERMINAL|WAIT|[LAST]]
[REQID(name)]
[L40|L64|L80|HNEOM]

Conditions: IGREQCD, IGREQID, INVLDC, INVMPSZ, INVPARTN, INVREQ, OVERFLOW, RETPAGE, TSIOERR

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.

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]
[LDLDC(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)]
[JJUSTIFY(data-value)|JUSFIRST|
JUSLAST]
[ACCUM]
[L40|L64|L80|HONEYOM]
```

Conditions: IGREQCD, IGREQID, INVLDC, INVPARTN, INVREQ, RETPAGE, TSIOERR

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 162.

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

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 203.

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|HONEYOM]
```

Conditions: IGREQCD, IGREQID, INVPARTN, TSIOERR

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 203.

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|NONEOM]
```

Conditions: IGREQCD, IGREQID, INVLDC, INVPARTN, INVREQ, RETPAGE, TSIOERR

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]
[LAST]
```

Conditions: IGREQCD, INVREQ, RETPAGE, TSIOERR

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 183.

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 183.

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, INVLCDC, INVREQ, RTEFAIL, RTE SOME

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 195.

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 41.

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 191. 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 181.

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, 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 PESTAT 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, 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 outbound map to be used. (This option applies only to 3650 logical units with outbound 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 158. It is ignored if the target terminal does not support formfeed (that is, the DFHTCT TYPE=TERMINAL system macro does not specify FF=YES)

FREEKB

specifies that the 3270 keyboard should be unlocked after the data is written. If FREEKB is omitted, the keyboard remains locked.

Note that 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 192.

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 LENGTHERR 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 198.

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 176.

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 163.

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. (Refer to the *CICS/DOS/VS CICS-Supplied Transactions manual*.)

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, refer to "Logical Messages for Terminal Operator Paging" on page 184.

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, refer to "Logical Messages for Terminal Operator Paging" on page 184.

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 181 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, refer to "TITLE Option of the ROUTE Command" on page 198.

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, refer to "Cumulative Text Formatting" on page 192.

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
INVLD C
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 not supported by 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.

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.
- **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)** 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 243.

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.

Distributed transaction processing is described fully in the *CICS/DOS/VS Intercommunication Facilities 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 381.

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/DOS/VS 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 252.

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 31.

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
TOLLENGTH	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 command is issued by the task. 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/DOS/VS Intercommunication Facilities 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 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 17).

The system macros referred to in the following tests are described in the *CICS/DOS/VS 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/DOS/VS 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 architecture 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 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 VTAM logical unit in much the same way as it does with a 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 "Bibliography" on page v.)

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/DOS/VS 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

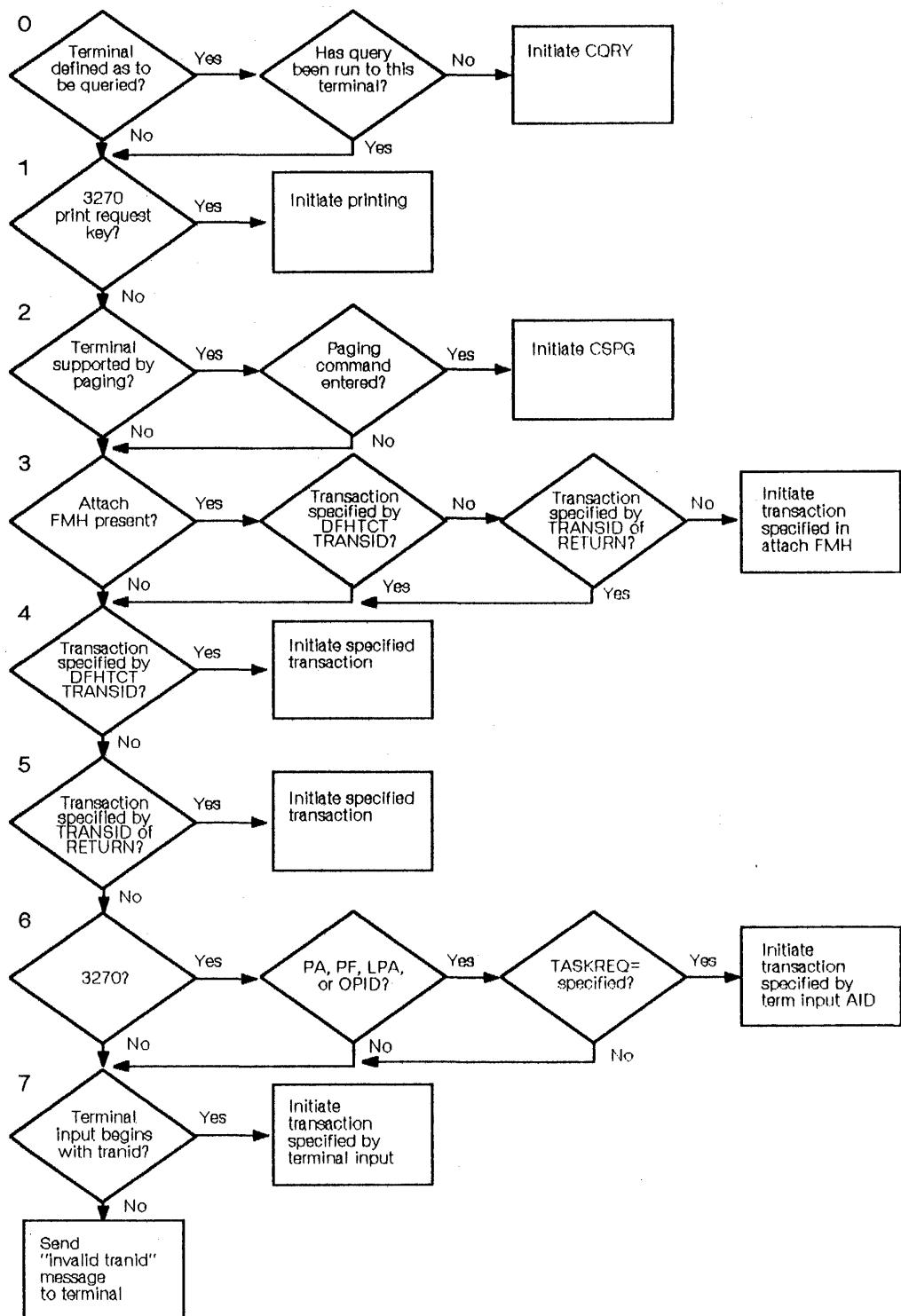


Figure 17. Terminal Oriented Task Identification

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; 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 Bibliography.)

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

If the user data contains one or more FMHs, the output request must specify the FMH option. When sending output data to a logical unit that expects an FMH, the FMH must be at the start of the user data to be transmitted.

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. Since 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 TERMTYPE 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 issues it, 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.

Sync Point Processing (ISSUE PREPARE)

```
ISSUE PREPARE
[CONVID(name)|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, refer to the *CICS/DOS/VS Intercommunication Facilities 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(name)|SESSION(name)]
```

Conditions: INVREQ, NOTALLOC,
SYSIDERR

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.

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.

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 18 on page 251.

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:

User Application Program Command	CICS ¹ Actions	Programmable Terminal Action
RECEIVE	Read initial (TI) Read continue (TT)	Write initial (TI) Write continue (TT)
SEND	Read interrupt (RVI) ² Read continue (TT) ³	Write reset (TR) or Write continue Write reset
SEND	Write initial (TI) Write continue (TT)	Read initial (TI) Read continue (TT)
RECEIVE	Write reset (TR) ⁴ Read initial (TI)	Read continue (TT) 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 18. BTAM Programmable Terminal Programming

- < 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 41.

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.

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.

| Do not use the ISSUE PRINT command in transactions that are invoked by a remote terminal.

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 31.

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.

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 31. 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 355.

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
DFHBMPLN	Printer new line
DFHBMASK	Autoskip
DFHBMUNP	Unprotected
DFHBMUNN	Unprotected and num
DFHBMPRO	Protected
DFHBMTRY	Bright
DFHBMNDAR	Dark
DFHBMFSE	MDT set
DFHBMPRF	Protected and MDT set
DFHBMASF	ASKP and MDT set
DFHBMASB	Auto and bright
DFHBMPS0	Shift out value X'0E'
DFHBMPS1	Shift in value X'0F'
DFHBMEOF	Field erased

Constant	Meaning	Constant	Meaning
DFHBMDET	Field detected	DFHUNNON	Unprotected numeric intensify light pen non-display non-print non-detectable MDT
DFHSA ¹	Set attribute (SA) order	DFHPROTI	Protected intensify light pen detectable
DFHERROR	Error code	DFHPROTN	Protected non-display non-print non-detectable
DFHCOLOR ¹	Color	DFHMFET	Mandatory fill and mandatory enter and trigger
DFHPS ¹	PS	DFHDFR	Default outline
DFHHLT ¹	Highlight	DFHUNDER	Under
DFH3270 ¹	Base 3270 field attribute	DFHRIGHT	Right
DFHVAL	Validation	DFHOVER	Overline
DFHOUTLN	Field outlining attr code	DFHLEFT	Left
DFHBKTRN	Background transp attr code	DFHBOX	Left+over+right+under lines
DFHALL ¹	Reset all to defaults	DFHSOSI	SOSI=yes
DFHDFT	Default	DFHTRANS	Background transparency
DFHDFCOL ¹	Default color	DFHOPAQ	No background transparency
DFHBLUE	Blue		
DFHRED	Red		
DFHPINK	Pink		
DFHGREEN	Green		
DFHTURQ	Turquoise		
DFHYELLO	Yellow		
DFHNEUTR	Neutral		
DFHBASE ¹	Base PS		
DFHDFHI ¹	Normal		
DFHBLINK	Blink		
DFHREVR	Reverse video		
DFHUNDLN	Underscore		
DFHMFIL ²	Mandatory fill		
DFHMENT ²	Mandatory enter		
DFHMF	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		

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)

```
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)
[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

```
BUILD ATTACH
[ATTACHID(name)]
[PROCESS(name)]
[RESOURCE(name)]
[RPROCESS(name)]
[RRESOURCE(name)]
[QUEUE(name)]
[IUTYPE(name)]
[DATASTR(name)]
[RECFM(name)]
```

LUTYPE6.1 Logical Unit (Continued)

```
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)|TERMINID(data-area)}
```

Condition: INVREQ

```
FREE
[SESSION(name)]
```

Conditions: INVREQ, NOTALLOC,

```
POINT
[SESSION(name)]
```

Condition: NOTALLOC

WAIT SIGNAL

```
WAIT TERMINAL
[SESSION(name)]
```

Conditions: NOTALLOC, SIGNAL

```
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/DOS/VS Intercommunication Facilities 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(name)]
[INTO(data-area)|SET(ptr-ref)]
LENGTH(data-area)
[MAXLENGTH[(data-value)]]
[NOTRUNCATE]

Conditions: EOC, LENGERR,
NOTALLOC, SIGNAL, TERMERR

SEND
[CONVID(name)]
[FROM(data-area)]
[LENGTH(data-value)]
[INVITE|LAST]
[CONFIRM|WAIT]

Conditions: EOC, INVREQ, LENGERR,
NOTALLOC, SIGNAL, TERMERR

CONVERSE
[CONVID(name)]
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(name)]

Conditions: INVREQ, NOTALLOC,

CONNECT PROCESS
CONVID(name)
PROCNAME(data-area)
PROCLENGTH(data-value)
SYNCLEVEL(data-value)
[PIPLIST(data-area)]
[PIPLENGTH(data-value)]

Conditions: INVREQ, NOTALLOC,
LENGERR

LUTYPE6.2 Logical Unit (Continued)

EXTRACT PROCESS
[PROCNAME(data-area)]
PROCLENGTH(data-area)]
[CONVID(name)]
[SYNCLEVEL(data-area)]
[PIPLIST(ptr-ref)]
PIPLENGTH(data-area)]

Conditions: INVREQ, NOTALLOC

ISSUE ABEND
[CONVID(name)]

Conditions: INVREQ, NOTALLOC,
TERMERR

ISSUE CONFIRMATION
[CONVID(name)]

Conditions: INVREQ, NOTALLOC,
TERMERR

ISSUE ERROR
[CONVID(name)]

Conditions: INVREQ, NOTALLOC,
TERMERR

ISSUE SIGNAL
[CONVID(name)]

Conditions: NOTALLOC, TERMERR

WAIT
CONVID[(name)]

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/DOS/VS Intercommunication Facilities 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/DOS/VS Intercommunication Facilities 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/DOS/VS Intercommunication Facilities 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

```
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]
[DEST(name)]
```

Conditions: LENGERR,
RDATT (not TCAM)

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.

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: LENGTHERR, 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 421.

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)

```
RECEIVE
  [INTO(data-area)|SET(ptr-ref)]
  LENGTH(data-area)
  [MAXLENGTH[(data-value)]]
  [NOTRUNCATE]
  [ASIS]
  [BUFFER]
```

Condition: LENGTHERR

```
SEND
  FROM(data-area)
  LENGTH(data-value)
  [DEST(name)]
  [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: LENGTHERR

ISSUE PRINT

```
ISSUE COPY
  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

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.

```
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

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, refer to the *CICS/DOS/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, 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, 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, ENDINPT,
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, LENGTHERR, 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, LENGTHERR, 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]
```

Conditions: 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:** LENGTHERR, 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:** LENGTHERR, 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: LENGTHERR

```
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: LENGTHERR

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.

CNOTCOMPL

indicates that the request/response unit (RU) sent as a result of this SEND command will not complete the chain. If this option is omitted and chain assembly has been specified, the RU will terminate the chain.

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, refer to documentation supplied by the subsystem 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 refer to the documentation supplied by 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, it is important to note 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, refer to documentation supplied by the subsystem 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.

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 31.

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. (Note that the receiving CICS system will use just the first four 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, refer to documentation supplied by the subsystem 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 than 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, refer to documentation supplied by the subsystem 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, refer to documentation supplied by the subsystem 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, refer to documentation supplied by the subsystem 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, refer to documentation supplied by the subsystem 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, refer to documentation supplied by the subsystem 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/DOS/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/DOS/VS Intercommunication Facilities 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

occurs if the 3651 is unable to initiate 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

occurs when an inbound SIGNAL data-flow 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.

service, or if all sessions are out of service.

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

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.

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.

Chapter 3.4. Report Controller

The report controller is a separately priced feature of CICS. It provides an end user interface to the POWER LST and POWER RDR queues.

This makes it possible for end users (who may not be DP professionals) to control the printing of their own reports, without needing to know anything about either CICS or POWER.

The report controller provides panel-driven transactions (CEMS and CEOS) for the end user. With these transactions, the user can control when the report is printed, which printer is used, how many copies to print, and so on. They do this by **changing the characteristics of the report** (this is the phrase we use to include all these operations: you will see it used in the *CICS/DOS/VS Report Controller User's Guide*.)

The characteristics of the report are set up initially by the program you write to create the report. The basic commands for creating reports (SPOOLOPEN, SPOOLWRITE, SPOOLCLOSE) are provided by CICS itself, but you need the report controller feature to be able to use the options that set many of these characteristics.

In addition to options related to controlling printing, the report controller provides several other options that make report creation easier than existing methods.

If you are designing application programs to create reports that you want end users to control and print using the report controller, you should first read the *CICS/DOS/VS Facilities and Planning Guide*. This gives an overview of report creation using the commands and options that are described in detail here. It also gives an overview of how the

end user will control and print their reports: how they will use the various characteristics that you set up. This will help you to understand how your choice of options will influence their work.

For more detailed information about using the report controller transactions, look at the *CICS/DOS/VS Report Controller User's Guide*, which has been designed specially for non-DP end users.

We provide a sample program, listed in Appendix G, "Report Controller Sample Programs" on page 549. This is based on the Low Balance Report sample program in Appendixes D, E, and F, and it shows how to create a report using the report controller. The source code is supplied in three versions: assembler language, COBOL, and PL/I.

In addition, we supply a transaction (SREP) that creates a selection of reports so that end users can try out the report controller. This also shows you how to create reports, and the source program is also supplied in each of the three languages.

For more details about running both the sample program and SREP, see the *CICS/DOS/VS Installation and Operations Guide*. The reports that SREP creates form the basis of the tutorial in the *CICS/DOS/VS Report Controller User's Guide*.

You can use the command interpreter (CECI) to try out the report controller commands. (You can see, although you cannot execute, the commands, even when you do not have the report controller installed). However, we recommend that you do not use CECI to create reports.

Reports

In this chapter, a **report** is a file of data to be printed. In one special case, it can be a file of job control (JCL) statements and, optionally, data to be processed by a batch program. Reports are held as POWER spool files during and after creation, and before, during, and sometimes after, printing.

Because a report is equivalent to a spool file, the commands you use to create them begin with **SPOOL**, to distinguish them from commands you use for creating ordinary files. We sometimes refer to them collectively as the "SPOOL" commands. As you might expect, they are:

- EXEC CICS SPOOLOPEN REPORT ...
- EXEC CICS SPOOLWRITE REPORT ...
- EXEC CICS SPOOLCLOSE REPORT ...

The **REPORT** option identifies the command as a report controller command, as well as specifying a name for the report. There are many other options that:

- Identify the report (to users and to CICS): see "Identifying a Report" on page 295.
- Specify how the report is to be created: see "Creating a Report" on page 296.
- Affect its appearance: see "Formatting a Report" on page 300.
- Influence how, when, and where it will be printed: see "Printing a Report" on page 307.
- Provide for security and recovery: see "Security and Recovery" on page 310.

There are three basic **report types** that you can create, as follows:

- Standard report
- Resumable report
- Log report.

Your choice will depend on the way in which your transactions are going to add data to the report. You can have one transaction create the whole

report in one go (**standard**), you can have several transactions or a pseudoconversational transaction create a report in separate stages (**resumable**), or you can have many independent transactions interleaving their data to create a continuous report (**log**).

We also provide for several different methods of formatting the report. The **report format** that you choose will depend on how you want to control such things as page breaks, line lengths, new lines, headings, footing, and page numbering. Your choice will depend largely on whether you are converting an existing program to use the SPOOL commands, or writing a new one. If the former, you should find that we have provided a suitable format.

If your program uses ASA carriage control characters, machine code carriage control characters (MCC), the SNA character string (SCS), a 3270 data stream, or BMS mapping, there is an output format for you. Not only that, but all these formats (with the exception of BMS) are converted to MCC, so that you can print any report on any of your printers, with the same results. However, if you want to use special features of your SCS or 3270 printers, you can avoid the conversion.

It is partly because of this convertibility and partly because we provide options for control over page breaks, line lengths, headings, footings, and page numbering, that you need not worry about the format for new programs. For simplicity, choose NOCC (no carriage control) or ASA; you will have maximum flexibility with minimum fuss.

If you have special requirements for graphics or an output format not mentioned above, you can write your own **escape** program which is invoked when the report is ready to be printed.

Finally, there is a special format (JCL) for creating input to batch jobs. We shall return to report formats later, but first let us see how you identify your reports and specify how they are to be created.

Identifying a Report

Report Name

The report name is the primary means of identifying the report. It is probably the name that the end user will know the report by. Your naming conventions for reports can influence the end user's efficiency, because they can use partial (or generic) names to request information about reports. If you allocate two or three characters to a user or group of users, and then start all their report names with those characters, it will be easy for them to list all their reports by typing only those characters.

You identify the report by name in the REPORT(name) option. This option makes the report you are creating into a report that the end user can print using the report controller transactions CEMS or CEOS. You must use the REPORT(name) option on each command: if you do not, the command is not recognized as a **report controller** command.

Report Title

You can also provide a 32-character **title** for the report, if you want to. If you do not, the report name is used. The title appears on the CEMS and CEOS report characteristics panel.

Userdata

In addition, you can provide a 16-character description known as **userdata**. If you do not, the first 16 characters of the title are used. If you do not provide either title or userdata, again, the report name is used.

The userdata appears on the report controller report list panel. This is quite significant from the point of view of security, because any report controller user can see the contents of userdata for all reports. So, do not put any sensitive information in userdata. If you do not put

anything in userdata, be careful about sensitive information in the title.

You can put whatever information you want in userdata. Some things that you may want to use it for are:

- To provide a reference number or other code, so that whoever created a report can recognize it on the report controller list panel. (You may have many users creating reports with the same name, or one user may create many versions of the same report. The reference number could be supplied by them to your application program, or generated by the application and displayed back to them.)
- To provide a reference number or other code, in the text of a report, so that whoever is controlling printing can relate the output back to the report controller list panel. (They may have several versions of the same report: this will help them to see which versions have been printed.)
- To provide a more meaningful name for the report. For example, if you code:

```
REPORT(TIM74MAY)
TITLE('DEPT74 TIMESHEET J.SMITH')
```

and let **USERDATA** default, DEPT74 TIMESHEET is visible to all report controller users. It simply elaborates a little on the report name. J.SMITH is only visible to report controller users authorized to see the characteristics of that report.

Tokens

A **token** is an 8-byte identifier that is used by each spool command so that the CICS task can recognize which report is being created. When a report is first opened for input, CICS allocates a token to it. This token must be stored by the application program, to be used when the report is written to and when it is closed.

When a resumable report is closed, the token is passed back to CICS and must be used when the report is opened again.

Report Number

You have no control over the report number in your program. The report number is allocated by POWER. It is shown on the CEMS/CEOS report list and report characteristics screens, so it can be used to distinguish between reports of the same name that are waiting to print. It is also printed on the separator pages preceding and following each report, if you choose to have them.

Creating a Report

Report Types

As stated earlier, you can create three different types of report. They are described in more detail here, together with examples of coding to create each type.

Standard Report

A report created by a single execution of a CICS transaction (a single task). A standard report is opened, written to, and then closed. If your program has all the data it needs to create the whole report before opening it, the report can be standard.

The following coding shows how to create a standard report:

```
EXEC CICS SPOOLOPEN
  REPORT(N21001)
  TOKEN(TOK001)
  NOCC
```

```
EXEC CICS SPOOLWRITE
  REPORT(N21001)
  TOKEN(TOK001)
  FROM(I0001)
```

```
EXEC CICS SPOOLWRITE
  REPORT(N21001)
  TOKEN(TOK001)
  FROM(I0001)
```

```
EXEC CICS SPOOLWRITE
  REPORT(N21001)
  TOKEN(TOK001)
  FROM(I0001)
```

```
EXEC CICS SPOOLCLOSE
  REPORT(N21001)
  TOKEN(TOK001)
  HOLD
```

Resumable Report

A report created by a series of separate CICS tasks, each of which may append data to it. A resumable report is opened in exactly the same way as a standard report. At the end of the task it is closed with the RESUME option. At the beginning of the next task it is opened with the RESUME option, and so on, until it is finally closed without the RESUME option.

The following coding shows how to create a resumable report:

TRN1:

```
EXEC CICS SPOOLOPEN
  REPORT(N21002)
  TOKEN(TOK002C)
  ASA
```

```
EXEC CICS SPOOLWRITE
  REPORT(N21002)
  TOKEN(TOK002)
  FROM(I0002)
```

```
EXEC CICS SPOOLCLOSE
  REPORT(N21002)
  TOKEN(TOK002)
  RESUME
```

```
EXEC CICS RETURN
  TRANSID(TRN2)
  COMMAREA(TOK002)
```

TRN2:

```
EXEC CICS SPOOLOPEN RESUME
  REPORT(N21002)
  TOKEN(TOK002)
```

```
EXEC CICS SPOOLWRITE
  REPORT(N21002)
  TOKEN(TOK002)
  FROM(I0002)
```

```
EXEC CICS SPOOLCLOSE
  REPORT(N21002)
  TOKEN(TOK002)
  RELEASE
```

If your program is pseudoconversational, that is, control is returned to CICS while the user is entering data on the screen, you should create a resumable report. You can gather some of the data, write it to the report and then send another screen for the user to enter more data. You then close the report with the RESUME option, and store the TOKEN for use by the transaction that will open the report again. CICS uses the TOKEN to identify reports. You can use this technique for a series of different transactions, or for the same transaction invoked iteratively.

You must also create a resumable report if you want to issue a sync point during report creation. This is because a sync point closes any report that is open, and sets it to READY status. To prevent this, you can code a SPOOLCLOSE RESUME command before the sync point. You must specify LOGICAL recovery for the report, if you are using syncpointing. (See "Report Recovery" on page 310.)

The following coding shows how to create a resumable report, in a program that includes a sync point:

```
EXEC CICS SPOOLOPEN
  REPORT(N21003)
  TOKEN(TOK003)
  LOGICAL
  NOCC

EXEC CICS SPOOLWRITE
  REPORT(N21003)
  TOKEN(TOK003)
  FROM(I0003)

EXEC CICS SPOOLCLOSE
  REPORT(N21003)
  TOKEN(TOK003)
  RESUME

EXEC CICS SYNCPOINT

EXEC CICS SPOOLOPEN RESUME
  REPORT(N21003)
  TOKEN(TOK003)

EXEC CICS SPOOLWRITE
  REPORT(N21003)
  TOKEN(TOK003)
  FROM(I0003)
.
```

Notes:

1. You cannot create a resumable report in JCL format.
2. You cannot direct a resumable report to a destination at a different node.
3. At SPOOLOPEN RESUME, a resource security level (RSL) check is performed. The operator must have the RSL value that is specified on the first SPOOLOPEN, among the RSLKEYs in the sign-on table entry.
4. Any number of resumable reports with the same name may be opened at the same time.

Log Report

A report that, once open, can be written to by a number of tasks running concurrently. Records written to each CICS task are interleaved, but this type of report appears as a standard report to POWER.

You need to open a log report only once, and then close it once when all records have been written. The log report can be closed either by the task that opened it or by another task.

A log report may be created in any format. For any particular report name, only one log report may be open at once with that name. An attempt to open a log report that is already open will result in the WRONGSTAT condition being raised.

The following coding shows how to create a log report:

Task 1:

```
EXEC CICS SPOOLOPEN
  REPORT(N21004)
  TOKEN(TOK004)
  LOG
  ASA

EXEC CICS SPOOLWRITE
  REPORT(N21004)
  TOKEN(TOK004)
  FROM(I0004)
.
```

Task 2:

```
EXEC CICS SPOOLWRITE
  REPORT(N21004)
  TOKEN(TOK004)
  FROM(I0004)
```

.

Task 3:

```
EXEC CICS SPOOLWRITE
  REPORT(N21004)
  TOKEN(TOK004)
  FROM(I0004)
```

.

Task n:

```
EXEC CICS SPOOLWRITE
  REPORT(N21004)
  TOKEN(TOK004)
  FROM(I0004)
```

```
EXEC CICS SPOOLCLOSE
  REPORT(N21004)
  TOKEN(TOK004)
```

Opening a Report

The **SPOOLOPEN** command is used in two ways:

- To open a report for the first time
- To reopen a resumable report that has been closed.

Opening a Report for the First Time

When you open a report for the first time, you select, either explicitly or by default, many options that specify the characteristics of the report.

In addition to specifying **REPORT**, **USERDATA**, **TITLE**, and **TOKEN**, you also specify:

- The report format
- Other formatting options
- Options that help control printing
- Options for security and report recovery.

These options are discussed in "Report Controller Options" on page 313.

Here is an example of the command that opens a report for the first time:

```
EXEC CICS SPOOLOPEN
  REPORT('MKT01A')
  TOKEN(TOKENC)
  NOCC
  TITLE('MARKET ANALYSIS A')
  HEAD(HEADC)
  USERDATA(USERC)
  DATETIME
  SEP
  HEADNUM
```

Reopening a Resumable Report

The **RESUME** option specifies that a resumable report is being reopened. The only other options you specify with this are **REPORT** and **TOKEN**, so the command will look like this:

```
EXEC CICS SPOOLOPEN RESUME
  REPORT(WAGEREPT)
  TOKEN(TOKEN1)
```

Here, **WAGEREPT** is a literal character string and **TOKEN1** is the name of a variable whose data value is the token saved from the **SPOOLCLOSE RESUME** command last executed for this **REPORT**.

When you use **SPOOLOPEN RESUME**, the next write will append a record to the report with the same token. The report must have previously been closed with the **RESUME** option.

Writing Data to a Report

You use the **SPOOLWRITE** command to write data to a report. You usually specify the data to be written using the **FROM** option, except for BMS mapping, when you can use the **MAPONLY** option instead. You can let CICS calculate the length of the data, or you can specify the length using the **FLENGTH** option. Note that if you are writing a COBOL program to create a non-BMS report, you **must** specify **FLENGTH**.

You can write any number of lines in one **SPOOLWRITE** command. CICS calculates how many lines to write in different ways, depending upon the report format. We go into more detail later, in "Formatting a Report" on page 300.

Usually, you specify the length of each line using the LINELENGTH option. The number of lines written by each SPOOLWRITE is calculated by dividing the FLENGTH value by the LINELENGTH value, any remainder being padded with blanks and considered one line. This results in a report consisting of lines of fixed length. This applies to ASA, MCC, NOCC, JCL, and sometimes to SCS and T3270.

For ASA and MCC, the number of lines written also depends on any valid control characters found in column 1 after the FROM area has been split up into lines by dividing FLENGTH by LINELENGTH.

For JCL, the LINELENGTH is always 80. You must ensure that each JCL statement begins in column 1 and ends in column 80, after the FROM area has been split up into lines.

With other report formats, each SPOOLWRITE command is assumed to be for a page. With BMS mapping, the map determines how many lines are written. With SCS and T3270, all of the data can be considered as a single unit and written as one record. Control orders within the data stream control the number of lines to be written. With ESCAPE format, you must control the lines and pages in your escape program.

Here is an example of the command that writes a line of data to a report:

```
EXEC CICS SPOOLWRITE
  REPORT('MKT01A')
  TOKEN(TOKENC)
  FROM(PRINTLINEC)
  FLENGTH(132)
```

Closing a Report

You use the SPOOLCLOSE command to close a report. In addition to the mandatory options REPORT and TOKEN, there are some other options that you can use with the SPOOLCLOSE command.

The LOG option identifies that a report is a log report.

When you close a report for the last time, you specify DELETE, HOLD, or RELEASE. Whether you close a report as HOLD or RELEASE influences the job of the end user who has to print the report, so you should look at the guidance information in "Controlling When Reports are Printed" on page 308.

The DELETE option is used only when you do not, after all, want already-written report records to remain as a spool file on the POWER LST queue.

When you close a resumable report, you specify the RESUME option. This option indicates that the report has not been closed permanently: writing to it may be resumed by a later CICS task. For this purpose, the SPOOLCLOSE command passes the token back to CICS. You must store the token and use it when you code the SPOOLOPEN RESUME to reopen the report.

A sync point forces a report to be closed in release status, so if you wish to create a report across more than one logical unit of work, you must code a SPOOLCLOSE RESUME command before the SYNCPOINT command. SPOOLCLOSE RESUME puts the report in resume status (POWER DISP=A). You must then code a SPOOLOPEN RESUME command before you can write to the report again.

The following coding shows examples of the SPOOLCLOSE command:

```
EXEC CICS SPOOLCLOSE
  REPORT( )
  TOKEN( )
  HOLD

EXEC CICS SPOOLCLOSE
  REPORT( )
  TOKEN( )
  RESUME
```

Formatting a Report

Report Formats

You can create a report in a number of different formats, which you specify at SPOOLOPEN time using the following options:

NOCC No carriage control

ASA ASA control characters.

MCC Machine code carriage control

SCS SNA character string.

T3270 3270 data stream

MAP BMS mapping.

ESC Escape mapping, for special things like graphics.

JCL Job control language format. The report is placed on the POWER RDR queue to be input to a batch job.

The format of a report dictates which of the options are valid, for example, headings or footings are not supported for a BMS map.

Format Conversion

Whether you specify ASA, MCC, NOCC, SCS, or T3270, the report controller arranges that the report can be printed on a system printer, on an SCS printer, or on a 3270-type printer, with the same results.

It does this by converting all these formats to MCC. If you do not want the conversion, because you want to use the special features of a 3270-type or SCS printer, for instance, color, you can specify **NOCONV**. If you do this, you will get the special features if you use the appropriate printer. However, the report cannot be printed on a different kind of terminal printer, or directly through POWER to a channel attached system printer.

Format conversion enables distributed report printing: each recipient will receive an identical report, whether it was printed on a CICS printer or on a system printer.

NOCC and ASA reports created using SPOOLOPEN, are always converted to MCC. ASA reports created by batch programs can also be printed on SCS or 3270 printers.

ESC, MAP, and JCL reports are not converted, but remain fixed for the lifetime of the report.

As we said earlier, it should be fairly easy for you to decide which format to use, whether you are writing a new program or converting an old one. Let us consider the simplest formats first.

NOCC, ASA, and MCC Formats

NOCC (no carriage control) is the format to choose if you do not need to be able to insert unconditional page breaks and line skips. The LINES option specifies the maximum number of lines on each page, and a new page is started whenever the LINES value is exceeded by the line count.

The number of lines created by each SPOOLWRITE command is calculated by dividing the LINELENGTH value into the FLENGTH value. The default LINELENGTH is 132. FLENGTH defaults to the length of the data area specified by the FROM option. Each line thus created counts as one, and each heading and footing counts as 2, in the line count.

As well as automated headings and footings, you can also specify page numbering and the DATETIME option with NOCC. NOCC data is always converted to MCC format. The machine code characters are inserted as the lines are inserted. Do not allow for them in your LINELENGTH.

If you want to be able to specify unconditional line skips or page breaks in your program, you should choose ASA format. You can also make use of the LINES option to have an automatic page break wherever a page is filled.

Using ASA characters is simple: all you do is code one of the following characters in the first position of each line that you want to print:

- blank** space 1 line before writing the data
- 0** space 2 lines before writing the data
- space 3 lines before writing the data
- +** suppress space, that is, print on same line as previous line
- 1** skip to channel 1 (new page) before writing the data.

ASA data is always converted to MCC format. You can improve performance by writing several ASA lines in one SPOOLWRITE command. The resulting increase in speed may, however, be offset by the increased storage needed for buffers.

If you write more than one ASA line in one SPOOLWRITE command, the LINELENGTH value is divided into the FLENGTH value to calculate the number of lines. You must ensure that after this calculation is made, the ASA control characters will be in column 1. The total number of lines printed will be FLENGTH divided by LINELENGTH adjusted to take account of the ASA characters. The default LINELENGTH is 133 (allowing for the ASA character and 132 characters of data.)

If you specify LINES(0), you can skip to channels 2 through 12. LINES(0) means that no automatic page breaks are inserted. If you want automatic page breaks, you cannot skip to channels 2 through 12.

In addition to ASA control characters, LINES, and LINELENGTH, you can also have automatic headings and footings, page numbering, and the DATETIME option with ASA format.

During report printing, invalid characters (less than X'40') within ASA and MCC data records are converted to blanks to prevent terminal errors. However, if the printer is defined to support DBCS (double byte character set), records will be printed as they are received from POWER.

MCC (machine code control) format is much the same as ASA, but you specify hexadecimal control characters to control unconditional line skips and page breaks.

Note: There may be a slight performance improvement if you use MCC format, because no conversion takes place.

The options you can use for formatting NOCC, ASA, and MCC reports are:

LINES

Specifies the page length in terms of number of lines on the page

LINELENGTH

Specifies the page width in terms of number of characters in the line

HEAD and FOOT

Specify heading and footing text

HEADNUM, FOOTNUM, and BOTHNUM

Specify page numbering

DATETIME

Specifies date and time.

Page Length

LINES (in the range 0 through 99) can be specified at SPOOLOPEN for NOCC, ASA, or MCC format reports and for reports with SCS or T3270 format (without NOCONV). During creation, the report controller keeps a line count and if LINES is exceeded, a page break is created using appropriate carriage control characters.

The number of lines per page are counted according to the report format, as follows.

ASA and MCC

The line count is based on the carriage control character, that is, the actual number of printed lines is calculated.

A heading counts 2 lines (that is, one header line and one blank line)

A footing counts 2 lines (that is, one blank line and one footing line)

If the carriage control character specifies skip to line 1 on new page, the line count is reset.

NOCC

Each line counts one line.

A heading counts 2 lines.

A footing counts 2 lines.

If LINES is not specified, a system default is used. If LINES(0) is explicitly specified, no page formatting is attempted; DATETIME, HEAD, FOOT, HEADNUM, FOOTNUM, and BOTHNUM are invalid options with LINES(0).

Line Length

LINELENGTH can be specified at SPOOLOPEN for ASA, MCC, or NOCC format reports.

LINELENGTH can also be specified at SPOOLOPEN for reports with SCS or T3270 format providing NOCONV is not specified. During creation, the report controller will insert new line (NL) carriage control characters when the SPOOLWRITE reaches the line length specified.

Headings

A heading (defined as the top line of each new page) is supported for reports of ASA, MCC, and NOCC format. The heading is supplied at SPOOLOPEN time and will appear as the first line of each printed report page.

A heading is printed whenever:

- Printing of a report begins
- A carriage control character "skip to new page" is encountered
- Lines/page (if specified) is exceeded.

A blank line will always appear immediately after a heading line.

Footings

A footing (defined as the bottom line of each page) is supported for reports of ASA, MCC, and NOCC format. The footing is supplied at SPOOLOPEN time and will appear at the bottom of each printed page.

The bottom line is calculated from the lines/page option or by implying the system default.

A footing is printed whenever:

- A carriage control character "skip to new page" is encountered (except if it appears on the very first line of the report)
- Lines/page or the system default is exceeded
- The end of the report is reached.

A blank line will always appear immediately before the footing.

Page Numbering

Automated page numbering can be requested by specifying the HEADNUM, FOOTNUM, or BOTHNUM options of the SPOOLOPEN command.

The page number(s) will appear as follows:

HEADNUM

The page number will appear as the last 5 characters (leading zeros suppressed) of the heading line. (If no heading line is supplied at SPOOLOPEN, a blank heading line is produced.)

FOOTNUM

The page number will appear as the last 5 characters (leading zeros suppressed) of the footing line. (If no footing line is supplied at SPOOLOPEN, a blank footing line is produced).

BOTHNUM

Page numbers appears as specified for HEADNUM and FOOTNUM.

Date and Time

On a report where a heading is specified at SPOOLOPEN or a heading is implied by specifying HEADNUM or BOTHNUM, the specifying of DATETIME triggers the report controller to insert on each heading line the date and time of creating the report.

If no heading is specified, a blank heading line is produced.

The date and time will appear 19 characters before the end of the line in one of following system defined formats:

DD/MM/YY HH:MN
MM/DD/YY HH:MN
YY/MM/DD HH:MN

where:

DD is the day of the month

MM is the month

YY is the year

HH is the hour of the day

MN is the minute within the hour

These formats are as specified in the CSA (defined in the SIT).

SCS and T3270 Formats

You can use these formats to take advantage of SCS or T3270 printer options (with NOCONV), or to keep all reports using same format, or you can use T3270 format to convert a report based on BMS mapping to print on a system printer: see "BMS Mapped Reports" on page 305.

Usually, when you send SCS (SNA character string) data or a 3270 data stream to a printer, you do not send it as individual lines, but as one long string containing SCS or 3270 control orders as well as data. It is the printer that acts upon these control orders and formats the data accordingly. You can continue to do this to create reports using the SPOOL commands, but the report controller provides another option: by default, the report controller converts SCS and T3270 reports into MCC format just as it does for NOCC reports,

dividing LINELENGTH into FLENGTH. For example, if your report is based on a screen width of 80 characters, you specify LINELENGTH(80) to ensure correct format conversion.

In addition, the following SCS control orders are supported:

BS - Backspace
CR - Carriage return
FF - Forms feed
LF - Line feed
NL - New line
TRN - Transparent

Any unsupported SCS orders are treated exactly as they are by printers that do not support them: if recognized as architected SCS orders, their default actions are taken as specified in *Systems Network Architecture: Sessions Between Logical Units*.

3270 to MCC Conversion

The following buffer control orders will be honored (12/14 bit addressing only) when carrying out format conversion:

SBA - Set buffer address
EUA - Erase unprotected to address
RTA - Repeat to address

Other 3270 orders will be deleted from the data stream. Blanks will be inserted where necessary to maintain correct format. The orders deleted from the data stream are:

SF - Start field
SFE - Start field extended
MF - Modify field
SA - Set attribute
IC - Insert cursor
PT - Program tab

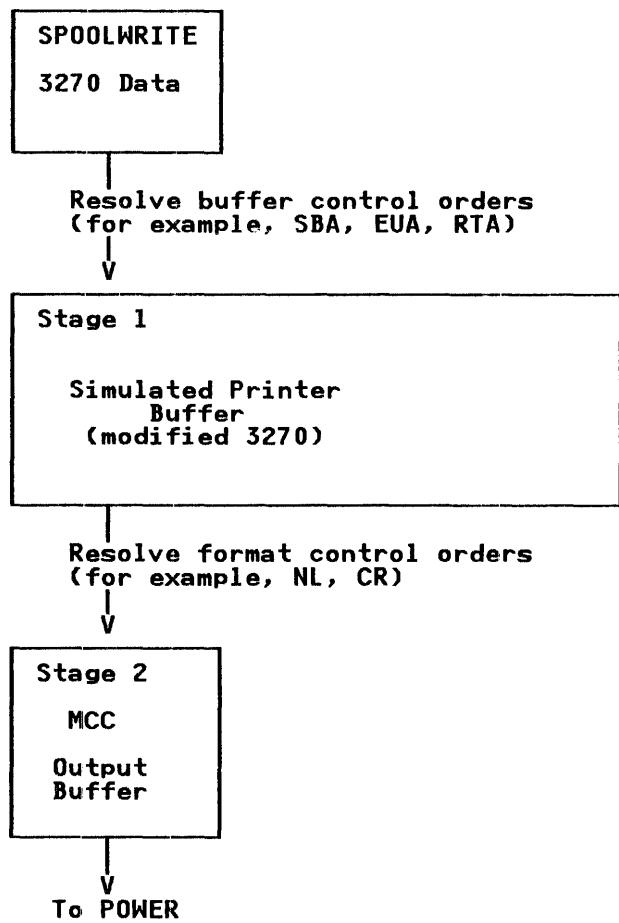
The following format control orders will be honored when carrying out format conversion:

CR - Carriage return
NL - New line
FF - Forms feed
EM - End message
GRE - Graphics escape (Defaulted to ' ')

Everything else will be treated as data and included in the data part of the MCC report.

Conversion is carried out as if the SPOOLWRITE data stream were prefixed by a command byte and WCC (write control character) byte (you must **not** include these in the SPOOLWRITE). The command is ERASE WRITE, the WCC defines both that the printout format is determined by NL/CR and that the buffer should be printed at the end of the write.

The 3270 data stream is converted to MCC in two stages, as shown in the following diagram.



The first stage emulates the data arriving at a printer buffer. Buffer control orders are resolved as the data is placed in the buffer.

The second stage creates an MCC report from this data. The format of this report is such that if

printed on a system printer it would be as near as possible to the format of the original 3270 data stream if it had been printed directly on a 3270 printer (that is, the format control orders are resolved).

Conversion starts from the start of the simulated print buffer, and continues until an EM order or the end of the buffer is reached. Note that to prevent unwanted data being included in this conversion, you must include an EM order with each SPOOLWRITE.

Simulated Buffer Size: The simulated buffer size is 2K bytes. To accommodate large data streams this will automatically expand to 4K or 8K bytes. If an address translation (for example, SBA) is beyond the 8K buffer, the task abends APSG. If there are more than 8K bytes of modified 3270 data (that is, after stage 1 conversion), the additional data will wrap to the start of the buffer.

Any unsupported 3270 orders are replaced by blanks where necessary, or are merely ignored. For example, SF (start field) occupies one character position in the print line, so it is replaced with one blank character. You can print reports in these formats on any printer: a system printer, or a CICS printer, whether it is a 3270-type or an SCS printer. You can also use the LINES option to control automatic page breaks, just as you can for NOCC.

You cannot have automated headings and footings, page numbering, or the DATETIME option with SCS or T3270 formats.

The NOCONV Option

For SCS and T3270 formats, you can specify the NOCONV option to prevent format conversion. If you do, these formats remain fixed for the lifetime of the report. Specifying NOCONV restricts a report created in T3270 format to printing only on a CICS 3270 printer, and a report created in SCS format to printing only on a CICS SCS printer. Use NOCONV if you want to use SCS or 3270 orders which are not supported by the conversion process (see above).

If you specify a destination, it must be for one that will ultimately be a CICS printer, because NOCONV reports cannot be printed on a system printer. If you do not specify a destination, you should ensure that reports are created using a display device whose SPOOLTO destination represents a CICS printer.

If you specify invalid control characters in a NOCONV report, the printer task will be abended.

You cannot use the LINES and LINELENGTH options when you use the NOCONV option.

BMS Mapped Reports

If you have reports that include a lot of constant text, you can use BMS to format it. You could use BMS reports instead of preprinted forms. The data that is usually preprinted could be held as constants in the map.

You could use the MAPONLY option of BMS to include a page of text that never varies, for example, a page of purchase conditions, or instructions.

There are two ways to use BMS.

The first way is by using BMS format. With this method, the report is formatted at print time. Only the variable data is held on the spool file, possibly saving space, but you can only print the report on a CICS printer.

The second way is to open the report in T3270 format. This specifies that T3270 format will be used to format the report. You then add the SET option to each SEND MAP command. This sets a pointer to an area containing the mapped data. See "SEND MAP Command" on page 226. You then insert a SPOOLWRITE command, writing FROM the area pointed to by the SET.

This method allows you to print BMS-mapped reports on a CICS printer or on a system printer. It is the only way that BMS-mapped reports can be printed on a non-CICS printer. The T3270 format is described in "SCS and T3270 Formats" on page 303.

BMS Mapping Format

You specify that the report will be formatted using BMS maps, by opening the report with the MAPNAME option. You must also use the MAPNAME option on each SPOOLWRITE. You must name the map being used on the SPOOLOPEN, the SPOOLWRITE, or on both. You can use more than one map in a report, but you cannot create a page from more than one map: one map is considered to be one page long. In other words, there is no ACCUM option, and you cannot specify headers, trailers, floating maps, and so on.

If you specify a destination, it must be for a CICS printer, because BMS-mapped reports cannot be printed on a system printer. If you do not specify a destination, you should ensure that reports are created using a display device whose SPOOLTO destination specifies a CICS printer.

You can use all the options for controlling printing, and for security and recovery. However, the automated formatting features available for NOCC, ASA, and MCC formats are not available for MAP format. This includes automatic page breaks, headings and footings, page numbering, and the DATETIME option. All your formatting must be specified within the map itself.

The length of the data is calculated from the length of the physical map. Remember that the map set and maps that you specify must be available to format the report at the time it is printed. Once the report is being printed, however, disabling a map will have no effect until after the printer has been stopped.

ESCAPE Format

If you want to create reports in any format other than those described above, you must provide an **escape program** that will be invoked at printing time to format the report. You can use this for graphics, or graphics and text on the same page, for instance. These reports cannot be printed on a system printer, but they can be printed on CICS printers.

You specify the name of an escape program using the ESCAPE option on the SPOOLOPEN command. You must ensure that this program is locatable at printing time. You can use several different escape programs for different reports.

The data written by each SPOOLWRITE is considered one unit and is written as one record, the maximum record size being 32767. You must specify the length of the data as FLENGTH if you are using COBOL; if you are using assembler language or PL/I, the length is calculated from the length of the FROM area.

You can use all the options for controlling printing, and for security and recovery. However, the automated formatting features available for NOCC, ASA, and MCC formats are not available for ESCAPE format. This includes automatic page breaks, headings and footings, page numbering, and the DATETIME option. All your formatting must be specified within your escape program itself.

When printing ESCAPE reports, CEMS or CEOS printer commands cannot be actioned until control is passed back from the user escape program. Control is passed back at end-of-copy and at end-of-report. This means that you cannot stop or pause a printer until the end of copy or report is reached.

When the report is ready to be printed, it is written to temporary storage. At end-of-report a CICS LINK is performed to the name supplied with the ESCAPE option. An 80-byte COMMAREA is passed to the escape routine and consists of:

On Output Link

link name. 8-byte temporary storage name. The name consists of a concatenation of the hex characters FB00, the characters PS, and xxxx, where "xxxx" is the termid of the printer serving the destination.

On Return

A single byte return code followed by 79 bytes of user message text. If the return code is X'00', the escape report will be set to its normal disposition (that is, it will be held or deleted). If the return code is not X'00', message DFH5467 will be sent to CSPW,

containing the return code and any message text that you may insert in the trailing 79 bytes of the COMMAREA. The report on the POWER queue will then be set to ERRPRT status.

When control is returned from the escape routine, the temporary storage is deleted.

JCL Format

Using the SPOOL commands you can create a file of input to a batch program. JCL format forces a LINELENGTH of 80, to create a file of 80-byte card images. These files are known as JCL reports, or jobs, and are written to the POWER RDR queue (not to the POWER LST queue). As you might expect, the options available for creating JCL reports are different from those available for creating printable reports. In fact, there are very few options.

A JCL report may be created as a standard report, or by many transactions as a log report. It may not be created as a resumable report.

A JCL report cannot be defined as physically recoverable; however, it can be defined as logically recoverable. And, because log reports cannot be logically recoverable, a log report in JCL format is not recoverable at all.

The command that opens a JCL report looks like this:

```
EXEC CICS SPOOLOPEN
  REPORT(REP1)
  TOKEN(TOKC)
  JCL
  [LOG]
  [LOGICAL]
```

Some of the characteristics of JCL reports that CEMS users can use to control batch job submission can be specified on the JOB card, but there are no more options that you can specify on the SPOOLOPEN command. There are no options for formatting: a JCL report is always 80 characters wide and written continuously. The resource security level value (RSL value) is assumed to be 1.

Even the job status is derived from the JOB card you write to the file, and not from the SPOOCLOSE command. The SPOOCLOSE command looks like this:

```
EXEC CICS SPOOLCLOSE
  REPORT(REP1)
  TOKEN(TOKC)
```

JOB Card Parameters

The following parameters on the JOB card written to a JCL report are of relevance to users of the report controller:

- CLASS
- PRIORITY
- DISP
- JOBNAM
- USER
- DEST or LDEST

The class, priority, and status of JCL reports can be changed by any authorized user of the CEMS transaction, if the user has the RSL value of 1 included in the RSLKEYs in their sign-on table entry.

If the JOB statement is missing, the report name will be AUTONAME, the class will be A, and the POWER defaults will be used.

If a report submitted to POWER does not contain POWER JECL, the required JECL will be inserted by POWER, and the characteristics of the JCL report will default to those allocated by POWER. You must ensure that the syntax of the JCL is correct, because CICS does not check it.

When creating JCL reports, if a POWER EOJ statement is output, the POWER job is made available for running before the SPOOLCLOSE is issued. This means that SPOOLCLOSE DELETE will only delete the last POWER job and not the complete JCL report (job).

Printing a Report

You associate a report with a printer by coding the DESTINATION option and, if the printer is remote, the NODE option. Even if there is a printer assigned to that destination, the report will be printed only when the report is in READY status. Even then, it will be printed only if the printer is assigned to the same output class.

Reports are printed, according to priority, within a class. To print a report, the printer must also be assigned to the same FORMS type. All these characteristics can be altered by the CEMS or CEOS user. In addition, you can set the number of copies and whether the report has separator pages (SEP or NOSEP), and the end user can also alter these. What the end user cannot alter is the RSL value. In addition to destination, status, class, priority, and forms, the RSL value determines whether a report can be printed by a given printer.

Controlling Where Reports are Printed

Destinations

The report controller uses logical printer destinations. The **destination** associated with a printer is the means by which you can direct reports to a printer, without naming the printer explicitly in your program. The destination can have a number of printers associated with it. The destination for a printer can be specified as the SPOOLDEST in the TERMINAL definition (or as SPLDEST in the DFHTCT macro). This destination can be overridden when the printer is started (either by using CEMS or CEOS or by using the POWER PSTART command). This destination can also be changed temporarily by using CEMS or CEOS.

You can specify the destination at which a report is to be printed, in the SPOOLOPEN command. However, you may not want to code a specific destination for the report in your program, in which case, you can let the destination vary according to the display device used to create the report. Each display device can have a default destination specified in the SPOOLTO attribute of its TERMINAL definition (or in the DFHTCT macro). If SPOOLTO is not specified, the

destination defaults to ****SYSPRT**, a special destination that indicates that a report is destined for the system printer.

You can specify ****SYSPRT** as a destination on the **SPOOLOPEN** command to explicitly direct the report to the system printer.

Whether the report is created using a destination coded in the **SPOOLOPEN** command, or derived from **SPOOLTO**, it can be altered using the transactions CEMS or CEOS.

Note: A report controller destination is the same thing as a **POWER USERID**, but it is not the same as a **CICS USERID**. Note also that a report controller destination is not the same thing as a transient data destination.

Examples of destinations are as follows.

Display devices:

DD01 SPOOLTO(DD01DEST)
DD02 SPOOLTO(unspecified)

Printers defined to CICS:

PR01 SPOOLDEST(DD01DEST)
PR02 SPOOLDEST(DD01DEST)
PR03 SPOOLDEST(XXXXDEST)

Reports created at DD01, with no **DESTINATION** specified on the **SPOOLOPEN** command are directed to PR01 or PR02.

Reports created at DD02, with no **DESTINATION** specified are directed to a system printer.

Reports created at DD01 or DD02, with a destination of **XXXXDEST** specified are directed to PR03.

Reports created at DD01 or DD02, with a destination of ****SYSPRT** specified are directed to a system printer.

Routing Reports to Remote Destinations

You can route a report to a destination associated with a printer at a different node in the network. You cannot do this with resumable reports, nor with JCL format. Specifying **NODE** without **DESTINATION** means that the report will be routed to a system printer. The **SPOOLDEST** of the display device is ignored.

Controlling When Reports are Printed

Report Status

When you close a report for the last time, you specify **HOLD** or **RELEASE**.

Both result in a change in the status of the report on the **POWER LST** queue. A report closed as **HOLD** (**POWER DISP = L**) will appear to report controller end users as **HELD**. The end user will have to reset this status to **READY** before the report can be printed. When the report has been printed, the status is reset to **HELD** again. This means that the report can be printed again, possibly by mistake. You may therefore want to limit the use of **HOLD** as an option when closing reports.

On the other hand, **HOLD** does have the advantage that a report lost because of a printing failure can be reprinted without being created again.

HOLD gives end users the most complete control over their reports, if that is what they want. They will always have to intervene before a report can be printed, and they will always have to delete their reports from the queue when they no longer need them.

A report closed as **RELEASE** (**POWER DISP = D**) will appear to the end user as **READY**. It will be printed as soon as a printer is available for the appropriate destination, output class, and security level (RSL). After being printed, the report is deleted from the **POWER LST** queue and is not available for reprinting. If you use **RELEASE**, you should ensure that you can create again, if necessary, a report lost because of a failure when it was being printed.

On the other hand, RELEASE has the advantage that reports are printed and deleted from the queue with no operator intervention, beyond ensuring that a printer is started for the correct destination and class.

Output Class

Each report is assigned to one of 26 output classes. Each printer is also assigned to one of the 26 output classes. A report can only be printed by a printer assigned to the same output class. The default output class for reports is specified in the DFHSIT SPOOL operand.

There is no restriction on the number of different destinations, forms types, and RSL values you can have for the same output class. However, you may find it helpful to use output classes to keep reports with different forms types separate. This is particularly helpful if most reports are closed with the RELEASE option (READY to print), or if the user releases many reports all at once.

A printer prints reports according to priority for the destination, output class, and RSL values it is assigned to. If the READY to print reports all have equal priority, all those of one forms type will be printed before going on to the next forms type. However, where different priorities are used, extra work for the end user who has to change the paper would result, unless the forms types are kept in separate output classes. The output class can be changed by the CEMS or CEOS user.

Priority

Each report is assigned a priority value from 1 through 9. For a particular destination and class, reports with priority 9 will be printed first, followed by priority 8, and so on, down to 1 (as long as the RSL value is valid for that printer). The default priority value is 3. The priority can be changed by the CEMS or CEOS user.

Controlling How Reports are Printed

Forms Type

The forms type is a 4-character name for a type of stationery. You may, for example, have different types of preprinted stationery, or different sizes of paper. It can be useful to separate different forms types by assigning them to different output classes; see output class, above.

Before a report can be printed, the forms type specified for the report must match the forms type specified for the printer. These can both be altered by the CEMS or CEOS user. However, no check is made on the physical size of the paper that is actually loaded. The report will be printed when the names of the forms match up.

Number of Copies

You can specify how many copies of a report are to be printed. The default number is 1, and the CEMS or CEOS user can change the number. Note that there is another way the user can print more than one copy: if the report is closed in HOLD status, it will return to the HELD queue after being printed. It can then be printed again, as many times as the user wants.

Separator Pages

A separator page is a cover page to be printed with a report. It makes it easier to see where one report ends and the next begins, and also gives some information about the report. You can specify whether or not separator pages will be printed with a report. If you specify SEP, two separator pages are printed preceding the body of the report, and two following it. If you specify NOSEP, no separator pages will be printed.

The size and content of the separator pages is determined by the DESTINATION name of the printer on which the report is printed. If the DESTINATION is **SYSPRT, the separator pages will be the standard POWER separator pages. If the DESTINATION is anything else, the separator pages are written by the report controller itself. The width of the report controller separator pages depends on the width (LINELENGTH) of

the report that they accompany (which, if the correct forms is loaded, is less than or equal to the width of the paper.) Each report controller separator page shows the following information:

- The contents of the USERDATA field
- The CICS USERID of the user who created the report (AUTHOR)
- The date and time the report was printed
- The DESTINATION the report was printed at (DEST)
- The TERMINAL name (TRMIDNT) of the printer (PRT)
- The output CLASS (CL)
- The FORMS type (FNO)
- The FCB code (always blank for CICS-created reports)
- The network NODE at which the report was printed
- The network NODE at which the report was created (O-NODE)
- The name of the CICS system that created the report (O-USR: always SYSCICS followed by the identifier specified in the DFHSIT SPOOL operand)
- The report number (O-NBR).

The CEMS or CEOS user can decide whether to have separator pages or not, and can override what you code.

Security and Recovery

Resource Security Level

Unauthorized access to a report can be prevented by specifying a resource security level (RSL) in the range 1 through 24, or "PUBLIC". If an RSL value is not coded on the SPOOLOPEN command, a value of "PUBLIC" will be assumed. You cannot code RSL on a JCL report: an RSL of 1 is always assumed.

The RSL check is carried out when:

- A SPOOLOPEN RESUME command is issued
- A SPOOLWRITE or SPOOLCLOSE of a log report is issued

- An attempt is made to gain access to the detailed characteristics of a report or change its control information using the CEMS or CEOS transactions. The transaction definitions supplied for CEMS and CEOS have RSLC(YES) specified. You may alter this if you want, by copying the definition. You may also want to use transaction security to limit access to CEMS and CEOS, by using TRANSEC.
- An attempt is made to print the report.

For reopening, writing to, and closing reports and accessing their characteristics, the RSL value must be included in the RSLKEY operand in the sign-on table (SNT) entry for the user. If the user is not signed on (using CESN or CSSN), the RSL value must be included in the OPERRSL keys in the TCT entry for the display that is being used.

For printing a report, the RSL value must be included in the OPERRSL keys in the TCT entry for the printer.

The RSL value for a report or a printer cannot be changed by the CEMS or CEOS user. If you are using RDO, the printer RSL can be changed by a CEDA user.

PRINTFAIL

Normally, a report that is being printed when the writer task (CEPW) abends, will be reprinted automatically when CEPW is recovered. This could result in, for example, two copies of an invoice being printed, or in sensitive information being printed while unauthorized people are present. To prevent this situation, you can specify PRINTFAIL for any report. A report with PRINTFAIL specified will not be reprinted without operator intervention.

Report Recovery

At SPOOLOPEN, the level of recovery required for the report is specified. The recovery level remains fixed for the lifetime of the report. Two types can be specified, as follows:

PHYSICAL

Each line is written directly to disk. This type of recovery makes heavy use of input/output, and its use should be carefully considered before it is selected.

You must specify PHYSICAL for LOG reports.

It is not allowed for JCL reports. Physical recovery, if coded, is ignored for JCL reports.

LOGICAL

The report is "checkpointed" by a SYNCPOINT command and the report is closed. If an abnormal termination (of either the task or CICS) occurs, the report will be deleted.

Sync Point

Issuing a SYNCPOINT command in an application program closes all reports.

All standard and resumable reports open at the time the sync point is issued are closed and their status set to "ready to print".

Standard reports with logical recovery for which a SPOOLCLOSE command has been issued before the SYNCPOINT is issued are committed, that is, they will be released for print. Only reports opened after the sync point are affected by abnormal termination of the task.

Log reports can only be recovered at the physical level. As a result, each record will be committed to disk.

See the *CICS/DOS/VS Recovery and Restart Guide* for more information about recovery.

Report Controller Commands

Open ASA, MCC, or NOCC Report

```
SPOOLOPEN
REPORT(name)
TOKEN(data-value)
{ASA|MCC|NOCC}
[TITLE(data-area)]
[CLASS(data-value)]
[LOG]
[USERDATA(data-area)]
[PRIORITY(data-value)]
[DESTINATION(name)]
[NODE(name)]
[RSL(data-value)]
[PHYSICAL|LOGICAL]
[COPIES(data-value)]
[FORMS(name)]
[SEP|NOSEP]
[PRINTFAIL]
[LINES(data-value)]
[LINELENGTH(data-value)]
[NONUM|HEADNUM|FOOTNUM|BOTHNUM]
[HEAD(data-area)]
[FOOT(data-area)]
[DATETIME]
```

Conditions: ILLOGIC, INVREQ, IOERR, NOSPACE, NOSPOOL, WRONGSTAT

Open MAP Format Report

```
SPOOLOPEN MAPNAME[(name)]
REPORT(name)
TOKEN(data-value)
[TITLE(data-area)]
[CLASS(data-value)]
[LOG]
[MAPSET]
[USERDATA(data-area)]
[PRIORITY(data-value)]
[DESTINATION(name)]
[NODE(name)]
[RSL(data-value)]
[PHYSICAL|LOGICAL]
[COPIES(data-value)]
[FORMS(name)]
[SEP|NOSEP]
[PRINTFAIL]
```

Conditions: CCERROR, ILLOGIC, INVREQ, IOERR, NOSPACE, NOSPOOL, WRONGSTAT

Open SCS or T3270 Report

```
SPOOLOPEN
REPORT(name)
TOKEN(data-value)
[SCS|T3270]
[TITLE(data-area)]
[CLASS(data-value)]
[LOG]
[USERDATA(data-area)]
[PRIORITY(data-value)]
[DESTINATION(name)]
[NODE(name)]
[RSL(data-value)]
[PHYSICAL|LOGICAL]
[COPIES(data-value)]
[FORMS(name)]
[SEP|NOSEP]
[PRINTFAIL]
[LINES(data-value)]1
[LINELENGTH(data-value)]1
[NOCONV]
```

Conditions: CCERROR, ILLOGIC, INVREQ, IOERR, NOSPACE, NOSPOOL, WRONGSTAT

¹ LINES and LINELENGTH cannot be specified for SCS (with NOCONV) and T3270 (with NOCONV).

Open JCL Format Report

```
SPOOLOPEN
REPORT(name)
TOKEN(data-value)
JCL
[LOG]
```

Conditions: CCERROR, ILLOGIC, INVREQ, IOERR, NOSPACE, NOSPOOL, WRONGSTAT

Open ESC Format Report

```
SPOOLOPEN ESCAPE(name)
REPORT(name)
TOKEN(data-value)
[TITLE(data-area)]
[CLASS(data-value)]
[LOG]
[USERDATA(data-area)]
[PRIORITY(data-value)]
[DESTINATION(name)]
[NODE(name)]
[RSL(data-value)]
[PHYSICAL|LOGICAL]
[COPIES(data-value)]
[FORMS(name)]
[SEP|NOSEP]
[PRINTFAIL]
```

Conditions: CCERROR, ILLOGIC, INVREQ, IOERR, NOSPACE, NOSPOOL, WRONGSTAT

Close a Report

```
SPOOLCLOSE
REPORT(name)
TOKEN(data-value)
[RELEASE|HOLD|RESUME|DELETE]
```

Conditions: ILLOGIC, INVREQ, IOERR, NOSPACE, NOSPOOL, NOTAUTH

Reopen a Report

```
SPOOLOPEN RESUME
REPORT(name)
TOKEN
```

Conditions: ILLOGIC, IOERR, INVREQ, NAMEERROR, NOSPACE, NOSPOOL, NOTAUTH, WRONGSTAT

Write to a MAP Report

```
SPOOLWRITE MAPNAME[(name)]
REPORT(name)
TOKEN
{FROM(data-area)
[FLENGTH(data-value)]|MAPONLY}
[MAPSET(name)]
[FLENGTH(data-value)]
[LOG]

Conditions: CCERROR, IOERR,
LENGERR, MAPERROR, NOSPACE, NOSPOOL,
NOTAUTH
```

Note: Commands similar to the report controller commands are used for the CICS interface to POWER, but they do not have the REPORT option; these commands are described in the *CICS/DOS/VS Customization Guide*.

Report Controller Options

ASA

Specifies that the report will be created using an ASA control character at the beginning of each line of data, as follows:

- blank space 1 line before writing the data
- 0 space 2 lines before writing the data
- space 3 lines before writing the data
- + suppress space (that is, print on the same line as the previous line)
- 1 skip to line 1 of new page before writing the data.

You should ensure that valid carriage control characters are used. If not, the CCERROR condition is raised.

If you specify LINES(0), you can skip to channels 2 through 12. LINES(0) means that no automatic page breaks are inserted. If you want automatic page breaks, you cannot skip to channels 2 through 12.

ASA data is always converted to MCC format. You can improve performance by writing several ASA lines in one SPOOLWRITE command. The resulting increase in speed may, however, be offset by the increased storage needed for buffers.

BOTHNUM

Specifies that page numbers are to appear on both headings and footings in the printed report. The page number will appear as the last 5 characters of each line with leading zeros suppressed. If you do not specify HEAD or FOOT, the rest of the heading or footing will be blank.

CLASS(data-value)

Specifies the POWER output class for reports in all formats except JCL. The default class is the one specified in the DFHSIT SPOOL parameter.

Data definition: 1 character (A through Z)

COPIES(data-value)

Specifies the number of copies (1 through 255) requested for this report. If not specified, 1 is assumed.

Data definition: signed binary halfword.

DATETIME

Specifies that the date and time of creating the report is to appear in the heading of each page.

The format of the date is determined by the format specified by the system programmer at CICS generation time, depending on the format chosen for the system. It can be one of following:

DD/MM/YY
MM/DD/YY
YY/MM/DD

The format of the time is HH:MM as specified in the CSA.

DELETE

Specifies that the report can be deleted without printing; it is no longer required.

DESTINATION(name)

Specifies the destination name (2 through 8 characters) at which the report is to be printed. For more information, see "Destinations" on page 307.

The valid characters that can be used in the destination name are as follows:

A through Z
0 through 9
@ \$ #
**SYSPRT

The first character must not be numeric.

Note: To avoid conflict with some POWER commands, DESTINATION should not be specified as LOCAL or SYSTEM.

ESCAPE(name)

Specifies the name of a user written escape program that will be invoked at printing time to format the output. For more information, see "ESCAPE Format" on page 305.

Data definition: 8 characters.

The valid characters that can be used in the escape name are as follows:

A through Z
0 through 9
@ \$ #

The first character must not be numeric.

Note: If the escape routine uses CICS commands, EIBRCODE must be set to zero at exit from the escape routine.

If you use the master terminal transaction (CEMT) to disable or copy escape programs after a printer has been started, the effects of that CEMT command may not be felt until after the printer has been stopped.

FLENGTH(data-value)

The FLENGTH option is mandatory only for a COBOL program writing a non-MAP

report. It specifies the length of the data supplied with a SPOOLWRITE command.

- If the report is in ASA, MCC, NOCC, or JCL format, the number of lines to write to the report is calculated by dividing the LINELENGTH or its default into the length of the data specified by the FLENGTH option. Any remainder from the division is considered a line and is padded with blanks to full length before being written to the report.

For ASA and MCC reports, the application programmer must ensure correct carriage control characters for all lines supplied in the command.

- If the report is in ESC, MAP, SCS (with NOCONV), or T3270 (with NOCONV) format, the full length of the data is considered one unit and is written as one record. The maximum record size is 32767.

If the FLENGTH option is omitted, the following applies:

- If the report is in ASA, MCC, NOCC, JCL, T3270, SCS, or ESC format, the FLENGTH option is implicitly calculated as the length of the data field specified in the FROM option. This length is resolved at program compilation time and cannot be dynamically changed during program execution.
- If the report is in MAP format, the length of the map is calculated by scanning the physical map. If the map set cannot be loaded from the library or the map cannot be located within the map set, the MAPERROR condition is raised.

Data definition: signed binary fullword.

FOOT(data-area)

Specifies the field containing the footing to be used in the report. The length of the field must be at least that specified by the LINELENGTH option (or its default). If

the report format is ASA or MCC, the first character of the field is used for the carriage control character.

Data definition: 1 through 205 characters.

FOOTNUM

Specifies that a page number is to appear as the last 5 characters of the footing of the report. If no footing is supplied at SPOOLOPEN, a blank footing line is inserted in the report.

FORMS(name)

Specifies the name of the stationery to be used to print the report. At print time, a match is made between the forms specified for the report and the forms type for the printer. (The forms type for a printer can be specified/ altered using the CEMS and CEOS transactions). If they match, the report is printed, otherwise the report will remain "in use" until the correct forms type is selected.

Note: It must be stressed that no physical check is made regarding the size of the paper in the printer; if the forms type is right, the report is printed, regardless of what paper is actually loaded.

Data definition: 4 characters.

The valid characters that can be used in the forms name are as follows:

A through Z
0 through 9
- . / @ \$ #

The first character must not be numeric.

FROM(data-area)

Specifies the field containing the data to be written. This option is mandatory. For ASA, MCC, NOCC, and JCL format reports, the data area can contain any number of lines. The report controller will calculate the number of lines by dividing LINELENGTH or its default into FLENGTH. Lines that are not filled will be padded with blanks before being written.

If FLENGTH is omitted, CICS calculates the length of the data area. It is the responsibility of the application programmer to ensure correct carriage control characters in all lines supplied. If a nonvalid carriage control character is encountered, the CCERROR condition is raised.

Data definition: 1 through 32767 characters for ESC, MAP, SCS (with NOCONV), and T3270 (with NOCONV). For ASA, MCC, NOCC, and JCL there is no upper limit.

HEAD(data-area)

Specifies the field containing the heading to be used in the report. The length of the field must be at least that specified by the LINELENGTH option (or its default). If the report format is ASA or MCC, the first character of the field should be left blank as it is used for the carriage control character.

Data definition: 1 through 205 characters.

HEADNUM

Specifies that a page number is to appear as the last 5 characters of the heading of the report. If no heading is supplied at SPOOLOPEN, a blank heading line is inserted in the report.

HOLD

(POWER DISP = L)

Specifies that the report is to be put in hold status after being closed. No attempt will be made to print the report until the status of the report has been changed using the CEMS or CEOS transactions, or, if preferred, by using the appropriate POWER command.

JCL

Specifies that the report is to be in JCL format, and that it is to be written to the POWER RDR queue as input to a batch program. See "JCL Format" on page 306. It is the responsibility of the application to ensure correct JCL syntax of the report. A JCL report has a forced LINELENGTH of 80 characters.

If a report submitted to POWER does not contain POWER JECL, the required JECL will be inserted by POWER, and the characteristics of the JCL report will default to those allocated by POWER.

The only other options you can specify with JCL are LOGICAL or LOG, but you cannot specify both for the same JCL report.

LINELENGTH(data-value)

Specifies the length of each line of the report. This value is specified at SPOOLOPEN and remains fixed for the entire life of the report. The valid range is 1 through 205 characters.

If DATETIME is specified as well, LINELENGTH must be greater than or equal to 20.

If HEADNUM, FOOTNUM, or BOTHNUM is specified as well, LINELENGTH must be greater than or equal to 6.

When deciding the linelength for a report, you should consider the type of printer on which the report will be printed. You are recommended to set the linelength to no more than the width of the printer carriage minus 1. If the linelength exceeds the width of the printer carriage, the format of the printed report may be unpredictable.

For ASA and MCC reports, LINELENGTH should include the carriage control character.

If this option is omitted, the following applies:

- A JCL report is always forced to 80 characters regardless of the LINELENGTH option.
- The LINELENGTH option is not meaningful for reports in ESC, MAP, SCS (with NOCONV), or T3270 (with NOCONV) formats.
- For ASA or MCC format report, 133 is the default.
- For NOCC, SCS, or T3270 formats, 132 is the default.

Data definition: Signed binary halfword.

LINES(data-value)

Specifies for ASA, MCC, NOCC, SCS (without NOCONV), and T3270 (without NOCONV) format reports, the number of lines accepted in each page of the report. The valid range is 0 through 99.

When writing to the report, if this number is exceeded without a "skip to new page" being encountered, a "default page break" takes place.

When the LINES option is not specified the system default is used.

If LINES(0) is explicitly specified, no page formatting is attempted and headings or footings are not allowed.

Data definition: Signed binary halfword.

LOG

Specifies that a log report is to be opened, written to, or closed. A log report is defined as a report that can be written to by a number of applications concurrently. A log report stays open until it is closed explicitly, or it is closed implicitly at CICS termination.

A log report can be in any format.

LOGICAL

Specifies that the report will have logical recovery.

Reports are not committed until a POWER checkpoint is issued. With LOGICAL recovery, a POWER checkpoint is issued only when the report is closed, either by a SPOOLCLOSE command, or when a sync point is issued.

If an abnormal termination occurs while a task is active, the report will be backed out to the last POWER checkpoint.

Logical recovery is not supported for LOG reports.

MAPNAME(name)

Specifies the name of the map to be used in a SPOOLWRITE MAPNAME or SPOOLOPEN MAPNAME command.

If MAPNAME() is specified on a SPOOLWRITE command, the value specified at SPOOLOPEN is used.

If MAPNAME() is specified on a SPOOLWRITE command, and no value is specified at SPOOLOPEN, the MAPERROR condition is raised.

Data definition: 7 characters.

The valid characters that can be used in the destination name are as follows:

A through Z
0 through 9
@ \$ #

The first character must not be numeric.

Note: Suffixes are not supported by the report controller.

MAPONLY

Specifies that only the map text is to be output. If you specify this option, you cannot specify FROM.

MAPSET(name)

Specifies the name of the map set that contains the MAPNAME to be used for a MAP format report.

If the MAPSET option is not specified, the map set name is assumed to be the same as the MAPNAME.

If MAPSET() is specified on a SPOOLWRITE command, the value specified at SPOOLOPEN is used.

If MAPSET() is specified on a SPOOLWRITE command and no value is specified at SPOOLOPEN, the MAPERROR condition is raised.

Data definition: 7 characters.

The valid characters that can be used in the destination name are as follows:

A through Z
0 through 9
@ \$ #

The first character must not be numeric.

Note: Suffixes are not supported by the report controller.

MCC

Specifies that the report will be created using MCC machine carriage control characters, as follows:

Hex	Result
0B	space 1 line immediately
13	space 2 lines immediately
1B	space 3 lines immediately
09	space 1 line after write
11	space 2 lines after write
19	space 3 lines after write
89	skip to channel 1 after write
8B	skip to channel 1 immediately
01	write without spacing
03	no operation

It is the responsibility of the application to ensure that the first character of each line to print is a valid carriage control character. If an invalid carriage control character is encountered, the CCERROR condition is raised.

NOCC

Specifies that no carriage control will be used when creating this report. You control page breaks using the LINES option. A new page will occur whenever the LINES option value is exceeded.

NOCONV

Specifies that no conversion to MCC format is required. If specified, the report can only be printed on an appropriate CICS printer. This option is applicable only to SCS and T3270 formats, and allows the user complete control over the data sent to the printer.

However, if incorrect data is sent, the writer task may abend.

When printing reports with this option, all the data specified with each SPOOLWRITE is sent to the printer. The printer buffer is then printed before the data associated with the next SPOOLWRITE is processed.

NODE(name)

Specifies the 8-character name of the target node, to which the file is to be routed by the system spooler (POWER). If you specify NODE and USERID, the file may be routed to a CICS printer. If you specify NODE alone, the file may be routed only to a system printer.

The valid characters that can be used in the node name are as follows:

A through Z
0 through 9
@ \$ #

The first character must not be numeric.

NONUM

Specifies that no automated page numbering is to take place.

NOSEP

Specifies that no separator pages are to appear with the printed report. Applies only to reports printed using the CEPW task.

PHYSICAL

Specifies that this report is to have physical recovery. When creating a report with physical recovery each SPOOLWRITE command, when completed, results in a physical write to disk of all data supplied in the command (with a POWER checkpoint). Physical recovery makes heavy use of input/output and should be carefully considered before being selected.

PRINTFAIL

Specifies that should CEPW abend (due to a CICS abend, and so on) before a report that is being printed has been closed, then when

CEPW has recovered, it will not attempt to reprint this report. The report status will be set to ERRPRT. Operator intervention is required before printing of the report can continue.

PRIORITY(data-value)

Specifies the priority of the report. Reports destined for the same destination that are READY will be printed in order of priority. When specified, the value must be in the range from 1 through 9 (where 9 is highest priority). If not specified, a value of 3 is assumed.

Data definition: 1 character.

RELEASE

(POWER DISP = D)

Specifies that, after completion of the SPOOLCLOSE command this report should be printed. The spooler will set the report to READY status. The report may be printed when the specified destination becomes available.

REPORT(name)

Specifies the name of the report. This option is required for every command.

SPOOLOPEN

An attempt to open a log report with the same name as one already open will be rejected.

SPOOLWRITE and SPOOCLOSE

The name must match the name specified in the SPOOLOPEN that used the same token. Otherwise, the NAMEERROR condition is raised.

Data definition: 2 through 8 characters.

The valid characters that can be used in the report name are as follows:

A through Z
0 through 9
- . / @ \$ #

The first character must not be numeric.

Do not use any of the following names, because you will not be able to manipulate them using CEMS or CEOS, or using POWER operator commands:

ALL
DEV
FREE
HOLD
LOCAL
PNET
RJE
STATUS

RESUME (POWER DISP = A)

Specifies that the report being closed is to be set in appendable status. When used in a SPOOLOPEN command, the report controller requires the report being opened to have previously been closed with the RESUME option, that is, the status of the report must be RESUME. This option cannot be used for JCL reports or for reports that have NODE specified on the associated SPOOLOPEN command.

You should issue a SPOOLCLOSE RESUME command before you issue a SYNCPOINT, and a SPOOLOPEN RESUME after the SYNCPOINT. Otherwise, the SYNCPOINT will result in the report being closed in READY status. However, there could be a transaction failure after the report has been closed, and before the SYNCPOINT. This could result in, for instance, file updates being backed out, while the report of file updates is not backed out.

RSL(data-value)

Specifies the security key that will be associated with the report. The value must be in the range of 1 through 24 or "PUBLIC" (127). If the RSL option is not specified, public is assumed.

Once the report is created, only terminal operators with the same resource level specification (specified in the sign-on table)

are able to alter its characteristics. The report can only be printed on printers with the same RSL value specified in OPERRSL.

The security key is checked at SPOOLCLOSE of a continuous report and at SPOOLOPEN RESUME. If the terminal operator cannot match the security key, the NOTAUTH condition is raised.

Data definition: Signed binary halfword.

After JCL reports are closed, RSL is set to 1

SCS

Specifies that the report being opened will use the SNA character string. It is the responsibility of the application programmer to ensure correct carriage control characters.

For SCS, a SPOOLWRITE is formatted into a whole number of MCC lines. When an SCS (with NOCONV) report is printed, the destination is checked for SCS support. If no SCS support is specified for the destination, message DFH5459 is sent to CSPW.

SEP

Specifies that 2 separator pages should be printed preceding the report, and 2 following. Separator page are described on page 309.

TITLE(data-area)

Specifies the title to be displayed on the CEMS/CEOS report characteristics panel. The title is a 32-character field which is not subject to any restrictions or validation. The default title is the REPORT name. USERDATA defaults to the first 16 characters of TITLE.

Note that DBCS characters can be used in TITLE, but if you do not code USERDATA, and let it default to the first 16 characters of TITLE, DBCS characters will be translated when they appear on the screen.

TOKEN(data-value)

A token is used by CICS to identify the report during creation. The option is required on all commands.

SPOOLOOPEN

You must specify an 8-character area (that must be word aligned) and store the value returned by CICS after a successful SPOOLOOPEN.

SPOOLWRITE and SPOOLCLOSE

You must specify an 8-character area that contains the value returned after SPOOLOOPEN (except for LOG reports, when a token area must be supplied but the returned value is not required.)

T3270

Specifies that the report to be created is in terminal 3270 format. A report written in T3270 format can contain any control characters normally supported by the 3270-type printers. It is the responsibility of the application to ensure that the control characters being used are valid as the report controller attempts no validation.

For T3270, SPOOLWRITE is formatted into a whole number of MCC lines. When a T3270 (with NOCONV) report is printed, the destination is checked for 3270 support. If no 3270 support is available, a message (DFH5459) is sent to CSPW.

USERDATA(data-area)

Specifies a 16-character report description or other reference information to be displayed on the CEMS/CEOS report list panel. If this option is omitted, the first 16 printable characters of the title are used. If TITLE is also omitted, the report name is used.

Note that DBCS characters should not be used in USERDATA, because they will be translated when they appear on the screen.

USERID(data-value)

This is a synonym for DESTINATION, as described on page 314. If you specify this option, you cannot specify DESTINATION.

Report Controller Exceptional Conditions

The exceptional conditions that can occur with the report controller are listed below. Each condition has a RESP value associated with it, and each RESP value has one or more further values, known as RESP2 values. A RESP2 value is an unique number (in decimal) corresponding to more detailed information that may help explain why the RESP condition has been raised.

The RESP values are listed in numerical order under field EIBRESP in Appendix A, "EXEC Interface Block" on page 413.

CCERROR RESP = 76

occurs when an invalid control character has been specified.

Default action: terminate the task abnormally.

ILLOGIC RESP = 21

occurs when there is an error in the validation of a parameter.

RESP2 Reason

1	Invalid class.
2	Reserved
3	Invalid report name
4	Invalid number of copies
5	Invalid escape name
6	Invalid priority
7	Invalid forms
8	Invalid destination
9	Invalid node
10	Invalid RSL value
11	Invalid lines per page
12	Invalid line length
13	Invalid map set name
14	Invalid map name
15	Invalid userid
16	Invalid token
17	Reserved
18	Invalid userdata field

Default action: terminate the task abnormally.

INVREQ RESP = 16

occurs when a request is invalid.

RESP2	Reason
1	Reserved
2	Token specified on SPOOLWRITE does not match the token received on SPOOLOPEN for a given report name.
3	A SPOOLWRITE has been attempted without a preceding valid SPOOLOPEN.
4	A SPOOLCLOSE has been attempted without a preceding valid SPOOLOPEN.
5	Token specified on SPOOLCLOSE does not match the token received on SPOOLOPEN for a given report name.
6-16	Reserved
17	A SPOOLCLOSE RESUME has been attempted for a report previously opened as a JCL type.
18	A SPOOLCLOSE RESUME has been attempted for a report that is on the XMT queue (NODE specified on SPOOLOPEN).
19-21	Reserved
22	Unsupported language (RPGII)

Default action: terminate the task abnormally.

IOERR RESP = 17

occurs when an error response is returned from POWER. RESP2 contains the POWER return and feedback codes. See the *CICS/DOS/VS Problem Determination Guide* for help in interpreting the RESP2 values.

Default action: terminate the task abnormally.

LENGERR RESP = 22

occurs when there is a length error.

RESP2	Reason
1	Reserved
2	Length of SPOOLWRITE for ESC, BMS, SCS(NOCNV), or 3270(NOCNV) type report is greater than the maximum allowable 32767 bytes.
3	Reserved
4	Invalid length for SPOOLWRITE. Length must be greater than 0.

Default action: terminate the task abnormally.

MAPERROR RESP = 77

occurs when there is an error during execution of a SPOOLWRITE MAPNAME command.

RESP2	Reason
1	No map set name specified for a report opened without a map set name.
2	No map name specified for a report opened without a map name.
3	Map set cannot be found.
4	Map name specified not found in specified map set.
5	No data fields in the specified map.
6	GETMAIN for the specified map set failed.

Default action: terminate the task abnormally.

NAMEERROR RESP = 74

occurs when an application has specified the wrong report name when issuing a command.

Default action: terminate the task abnormally.

NOSPACE RESP = 18

This condition occurs when there is no space on the POWER spool file to add records. Open logical reports are deleted by POWER, physical reports are kept.

Default action: terminate the task abnormally.

NOSPOOL RESP = 80

occurs when part, or all, of the POWER interface is unavailable.

RESP2	Reason
1	No subsystem present
2	No report controller interface present
3-5	Reserved
6	Interface being terminated
7	Interface not active
8	Interface being disabled
12	Interface has been stopped

Default action: terminate the task abnormally.

NOTAUTH RESP = 70

occurs when a resource security check has failed. The reasons for the failure are the same as for abend code AEY7, as listed in the *CICS/DOS/VS Messages and Codes* manual.

Default action: terminate the task abnormally.

WRONGSTAT RESP = 73

occurs when a SPOOLOPEN RESUME command is issued but the report has not previously been closed as resumable.

Default action: terminate the task abnormally.

Chapter 3.5. 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 publications *CICS/DOS/VS IBM 3767/3770/6670 Guide* and *CICS/DOS/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. Refer to "Chapter 1.5. Exceptional Conditions" on page 31 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.

Note: In the ISSUE END, ISSUE ABORT, ISSUE SEND, and ISSUE WAIT commands, the options CONSOLE, PRINT, CARD, and WPMEDIA1-4 are alternatives to DESTID and DESTIDLENG.

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, since 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 since 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. RIDFLD 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.

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.

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, IREQCD, 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.

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.

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 since 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 (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.

VOLUMEENG(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.

| Chapter 3.6. Structured Query Language/Data System (SQL/DS)

Many query applications arise from specific needs of certain groups of end users. Programmed queries - such as CICS transactions - let you tailor your query operations to specific needs. An application program implementing preplanned queries makes better use of system resources than unplanned queries issued from a terminal.

You can find further information in the *SQL/DS Application Programming* manual, order number SH24-5018.

The programmed SQL/DS commands are preprocessed once only for multiple executions and are held in executable form. Thus each program execution saves the extra processing needed for an unplanned query, such as an ISQL request.

SQL commands in a CICS application program are interpreted before the application program is compiled (or assembled). This is done by submitting the program source code to one of the SQL/DS preprocessors. CICS application programs that include SQL/DS commands are first processed by the appropriate CICS language translator before the SQL/DS preprocessor takes over.

The preprocessor produces modified source code for that language's normal compiler (or assembler) and saves an access module for that program in the SQL/DS data base. The access module holds the code needed to implement the SQL commands in the application program. Access modules are optimized for query and data manipulation operations. They are called every time the application program accesses SQL/DS data while executing.

During application development, many SQL syntax errors can be caught and corrected at the preprocessor stage. Programs don't have to be fully compiled (or assembled) and tested just to catch such errors.

You can use the CICS execution (command level) diagnostic facility (EDF) to debug application programs containing EXEC SQL commands. EDF will intercept your program's calls to SQL, but will not identify them as EXEC SQL commands. Instead it will show them as calls to the SQL adapter program.

Embedded SQL Commands

Each of the examples in Figure 19 shows the declaration of variables and a simple SQL query (SELECT) command. Notice that the program variables to be shared by the program and SQL/DS are declared as usual in a "declaration" section of the program. This section is preceded and ended by special SQL commands. You can also see that the program variables are always preceded by a colon (:) when an embedded SQL command refers to them.

The EXEC SQL prefix for SQL commands and the colons in front of each program variable are used by the SQL/DS preprocessors. The SELECT commands in these examples get the DESCRIPTION and QUANTITY data from a table named INVENTORY only where the value in the PARTNUMBER field in that table matches the contents of variable ZZ. The retrieved data is copied into variables XX and YY.

COBOL example

```
DATA DIVISION.  
WORKING-STORAGE SECTION.  
  EXEC SQL BEGIN DECLARE SECTION END-EXEC.  
01  XX.  
    49 XX-LENGTH PICTURE S9(4) COMPUTATIONAL.  
    49 XX-VALUE PICTURE X(24).  
77  YY PICTURE S9(9) COMPUTATIONAL.  
77  ZZ PICTURE S9(4) COMPUTATIONAL.  
  EXEC SQL END DECLARE SECTION END-EXEC.  
  .  
  .  
PROCEDURE DIVISION.  
  EXEC SQL SELECT DESCRIPTION, QUANTITY INTO :XX, :YY  
    FROM INVENTORY WHERE PARTNUMBER = :ZZ END-EXEC.
```

PL/I example

```
EXEC SQL BEGIN DECLARE SECTION;  
  DCL XX CHAR(24) VAR;  
  DCL YY BIN FIXED(31);  
  DCL ZZ BIN FIXED(15);  
EXEC SQL END DECLARE SECTION;  
  .  
  .  
EXEC SQL SELECT DESCRIPTION, QUANTITY INTO :XX, :YY  
    FROM INVENTORY WHERE PARTNUMBER = :ZZ;
```

Assembler example

```
XX  EXEC SQL BEGIN DECLARE SECTION  
  DS  H,CL24  
YY  DS  F  
ZZ  DS  H  
EXEC SQL END DECLARE SECTION  
  .  
  .  
EXEC SQL SELECT DESCRIPTION, QUANTITY INTO :XX, :YY  
    FROM INVENTORY WHERE PARTNUMBER = :ZZ
```

C

Figure 19. SQL commands embedded in application programs

Preparing a Program

The SQL commands that you have embedded in an application program must be analyzed and converted by SQL/DS **before** the normal language compilation (or assembly). This analysis and conversion is done by the SQL/DS preprocessing facility called PREP.

PREP does two jobs. First, it generates a new version of your source code, suitable for normal compilation (or assembly). This version contains the SQL commands in comment form and standard

host language code (for example, COBOL) to invoke an interface routine in place of the SQL commands.

Then, PREP converts the SQL commands into an access module, putting the module into the SQL/DS data base. This module contains machine code designed specifically for the SQL commands you've used.

After PREP, you input the new version of the source code to the appropriate compiler (or assembler) and link edit the resultant program with

a small SQL/DS routine to form the final load module.

When SQL/DS receives a request from a CICS application program, the access module created by the preprocessor is called to handle each SQL command. The SQL/DS interface routine handles the actual linkage and communications with SQL/DS.

The first thing SQL/DS does when it loads the access module is to check that it is still valid. SQL/DS will have stored the original SQL commands with the access module at PREP time. If the access module is no longer valid, SQL/DS will use these commands to try to PREP the program again. Only if this fails will you be told of any problem - otherwise the program will continue to run without any intervention.

Part 4. Control Operations

Chapter 4.1. Introduction to Control Operations	337
Chapter 4.2. Interval Control	339
Chapter 4.3. Task Control	353
Chapter 4.4. Program Control	355
Chapter 4.5. Storage Control	365
Chapter 4.6. Transient Data Control	367
Chapter 4.7. Temporary Storage Control	373

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 31.

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.

Since each end of an intersystem link may be in a different time zone, you should use the INTERVAL form of expiration time when the transaction to be started is in a remote system.

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). You should 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 1 January 1900. Refer to Appendix A, "EXEC Interface Block" on page 413 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-value)]
[DAYCOUNT(data-area)]
[DAYOFWEEK(data-area)]
[DAYOFMONTH(data-area)]
[MONTHOFYEAR(data-area)]
[YEAR(data-area)]
[TIME(data-area)]
[TIMESEP(data-value)])
```

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 353.

- 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. Since 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.

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.

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.

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/DOS/VS Resource Definition (Macro)* manual, or by means of the LOCALQ operand of the TRANSACTION definition as described in the *CICS/DOS/VS 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 the section "Expiration Times" earlier in the chapter. 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.

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. Since 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)

You use the RETRIEVE command to retrieve data stored by expired START commands. It is the only method available for accessing such data.

The syntax of the command is as follows:

```
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

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 1 January 1900, is to be stored.

For the FORMATTIME command, "parameter" specifies the data value, in milliseconds since 0000 hours on 1 January 1900, that 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 should normally use this option only when a date is needed for output purposes. Where a date is needed for analysis, you should request the date in explicit form, for example, 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 1 January 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.

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 will be canceled. If the REQID option is specified as well, the request identifier should be a name defined as recoverable to temporary storage.

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.

TERMINID(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 a local terminal in the TCT on the system in which the START command is executed.

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 16 megabytes.

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.

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/DOS/VS 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 if the terminal identifier in a START command cannot be found in the terminal control table.

Default action: terminate the task abnormally.

TRANSIDERR

occurs if the transaction identifier specified in a START command cannot be found in the program control table.

Default action: terminate the task abnormally.

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, ENQBUSY is raised. A HANDLE CONDITION ENQBUSY command will return control to the task at the ENQBUSY label, without waiting for the resource to become available.

DEQ 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 releases the resource during sync point processing or when the task is terminated.

If more than one ENQ is issued for the same resource by a task, the resource remains owned by that task until the task issues a matching number of DEQs.

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 DEQ, the resource to be dequeued from must be identified by the method used when enqueueing 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.

EXEC CICS ENQ RESOURCE(RESNAME)

or

**EXEC CICS ENQ RESOURCE(SOCSECN0)
LENGTH(9)**

Substituting "DEQ" for "ENQ" in these examples illustrates the ways in which a resource can be released.

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 LENGTH is specified in an ENQ, it must also be specified in the DEQ for that resource, and the values of these options must be the same. This option is only required if the resource is specified as a character string.

NOSUSPEND

specifies that application program suspension for ENQBUSY is to be inhibited. This condition will be handled as described on page 31.

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.

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 31.

The HANDLE ABEND command can be used to deal with abnormal terminations. Refer to

"Chapter 5.2. Abnormal Termination Recovery" on page 385 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, PGMDERR

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 358, the description of COMMAREA on page 363, and the description of TRANSID on page 363.

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.

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 20 on page 357 illustrates the concept of logical levels.

Transfer Program Control (XCTL)

```
XCTL  
PROGRAM(name)  
[COMMAREA(data-area)  
[LENGTH(data-value)]]
```

Condition: NOTAUTH, PGMDERR

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 358.

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.

Return Program Control (RETURN)

```
RETURN  
[TRANSID(name)  
[COMMAREA(data-area)  
[LENGTH(data-value)]]]
```

Condition: INVREQ, NOTAUTH

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 command is issued in a lower-level program, the program to which control is returned will have relinquished control by issuing a LINK command and will reside one logical level higher than the program returning control.

When the command is issued in a program at the highest logical level, control returns to CICS.

If the task is associated with a terminal, the TRANSID option can be used to specify the transaction identifier for the next program to be associated with that terminal; this causes subsequent input entered from the terminal to be interpreted wholly as data. In addition, the COMMAREA option can be used to pass data to

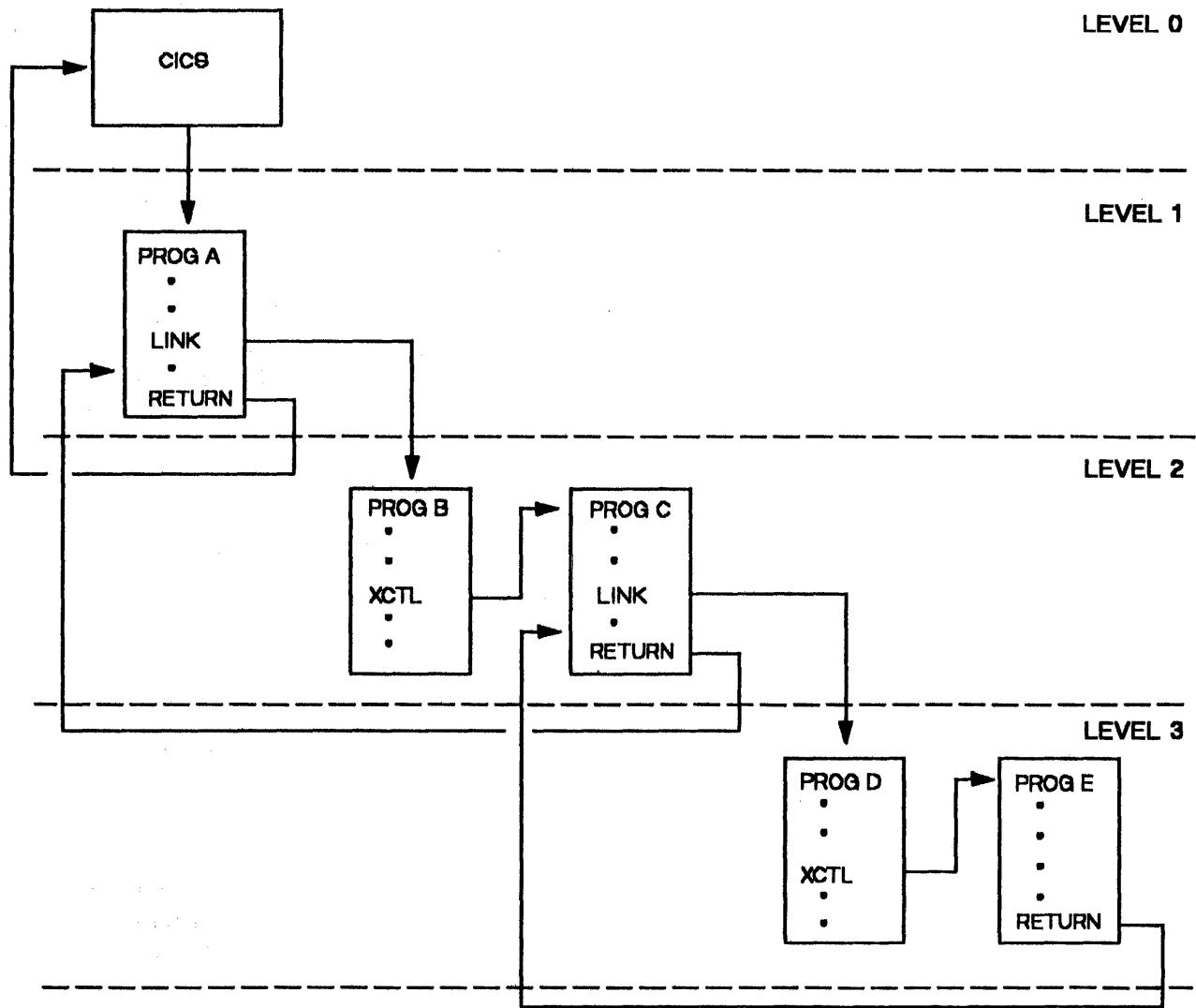


Figure 20. Application Program Logical Levels
 the new task that will be started. For further details, see "Passing Data to Other Programs" on page 358.

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.

The COMMAREA and LENGTH options can be used only when the RETURN command is returning control to CICS; the INVREQ condition will occur otherwise.

Load a Program (LOAD)

```

LOAD
PROGRAM(name)
[SET(ptr-ref)]
[LENGTH(data-area)]
  FLENGTH(data-area)]
[ENTRY(ptr-ref)]
[HOLD]
  
```

Condition: NOTAUTH, PGMDERR

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 131 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/DOS/VS Resource Definition (Macro)* manual, or the RELOAD attribute for the PPT as described in the *CICS/DOS/VS 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, PGMDERR

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 a COBOL program, you should specify the data area in the working storage section of the passing program and in the linkage section of the receiving program. 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 363. (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 363. 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, the data area must be specified in the linkage section of the receiving program, and must be called DFHCOMMAREA.

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.

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 41 for details.

Another way is to store the data in temporary storage; see "Chapter 4.7. Temporary Storage Control" on page 373 for details.

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. The examples also show how the RETURN command is used to pass data to a new transaction.

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
```

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'
DS 0H
.
.
.
LABEL1  DS 0H
.
.
.
LABEL2  DS 0H
.
.
.
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'

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 - 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;
```

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 a LINK command, a pointer to the data area is passed.

For an XCTL command, a pointer to the data area is passed or a copy of it (see "Passing Data to Other Programs" on page 358)

For a RETURN command, 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.

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.

For the LOAD command, FLENGTH and LENGTH are mutually exclusive.

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 alphabetic 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 either 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 TRANSID option is issued in a task that is not associated with a terminal.

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/DOS/VS 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.

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 31.

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 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]
```

Conditions: LENGTHERR, NOSTG

Storage should be released when no longer needed; it will then be available to other tasks. Other storage not released will be released by CICS when the task is terminated.

The LENGTH and FLENGTH options are mutually exclusive.

The following example shows how to obtain a 1024-byte area of main storage. You should 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)
```

Storage Control Options

DATA(data-area)

specifies that the main storage associated with the data area is to be released. This storage must have been acquired previously by a GETMAIN command and the length of data 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.

FLENGTH and LENGTH are mutually exclusive.

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 31.

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.

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 partition.

Destinations are **extrapartition** if the data is directed to a destination that is external to the CICS partition.

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 373).

Exceptional conditions that occur during execution of a transient data control command are handled as described in "Chapter 1.5. Exceptional Conditions" on page 31.

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, data base 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.

Extrapolition Destinations

Extrapolition 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 partition. In general, sequential extrapolation destinations are used for storing and retrieving data outside the CICS partition.

For example, one task may read data from a remote terminal, edit the data, and write the results to a data set for subsequent processing in another partition. Logging data, statistics, and transaction error messages are examples of data that can be written to extrapolation destinations. In general, extrapolation 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 extrapolation data and consists of sequential records that are fixed-length or variable-length, blocked or unblocked. The record format for an extrapolation destination must be defined in the DCT by the system programmer. (Refer to the *CICS/DOS/VS Resource Definition (Macro)* manual for details.)

Indirect Destinations

Intrapartition and extrapolation 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.

Since 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/DOS/VS 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 discussion of the DFHDCT TYPE=INTRA macro in the *CICS/DOS/VS 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: 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.

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 LENGTH option must be specified for a destination other than disk; length is not checked. If LENGTH is omitted, the LENGERR condition will occur.

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)

This command is used to read transient data from a predefined symbolic source. The source (queue) is identified in the QUEUE option.

```
READQ TD
QUEUE(name)
{INTO(data-area)|SET(ptr-ref)}
[LENGTH(data-area)]
[SYSID(name)]
[NOSUSPEND]
```

Conditions: IOERR, ISCINVREQ, LENGERR, NOTAUTH, NOTOPEN, QBUSY, QIDERR, QZERO, SYSIDERR

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 368, 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 LENGTH option must be specified for a destination other than disk, when the INTO option

is specified. If the LENGTH option is omitted, the LENGERR condition will occur.

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: 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 31.

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

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 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/DOS/VS 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 (see above) 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.

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 31.

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, 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 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-value)|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 31.

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 alphabetic, up to eight characters in length, and must be unique within the CICS system. Do not use hexadecimal 'FA' through 'FF' as the first character of the name; these characters are reserved for CICS use.

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 refers to data whose length is zero, or when a WRITEQ TS command specifies a queue which is locked and awaiting ISC session recovery. (Refer to the appropriate *CICS/DOS/VS Installation and Operations Guide* for details.) This condition occurs also 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.

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/DOS/VS 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	381
Chapter 5.2. Abnormal Termination Recovery	385
Chapter 5.3. Trace Control	389
Chapter 5.4. Dump Control	395
Chapter 5.5. Journal Control	399
Chapter 5.6. Recovery (Sync Points)	405

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 49.
- 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. (If running sequential support on FBA devices under VSE, the user must employ LIOCS, because no utility exists for this environment.)

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 CSMT SHUTDOWN with appropriate responses following the initial message to respond to the CSMT 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 CSMT SHUTDOWN 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. (EOF applies only to a card reader.) 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, refer to "Chapter 3.3. Terminal Control" on page 239.

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 355.)

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 21 on page 386, 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.

Refer to the section dealing with creating a program abend exit in the *CICS/DOS/VS Customization Guide* for additional information about exit routines, and to the *CICS/DOS/VS 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 31.

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

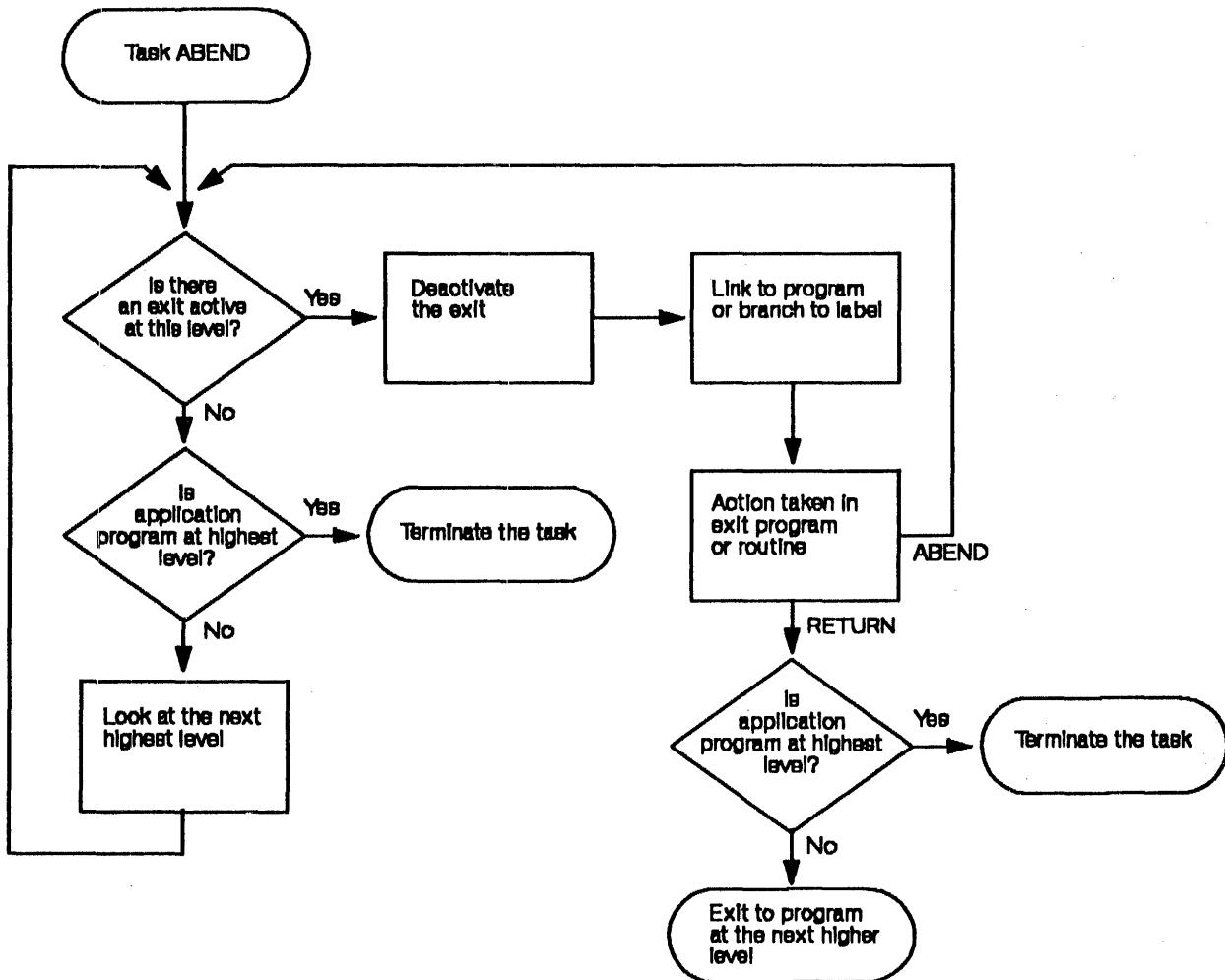


Figure 21. ABEND Exit Processing

such request in any application program at the same logical level.

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.

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.

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.

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 name may consist of up to 4 characters.

CANCEL

specifies that exits established by HANDLE ABEND and ABEND commands are to be ignored. HANDLE ABEND CANCEL cancels a previously established exit at the logical level of the application program in control. ABEND CANCEL cancels all exits at any level in the task (and terminates the task abnormally).

LABEL(label)

specifies the label to which control will branch if abnormal termination occurs. LABEL cannot be used in PL/I 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 8 alphanumeric characters and must have been defined in the 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.

The 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/DOS/VS 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 393.

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.

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 395.

Trace Table Format

The trace table consists of a trace header and a variable number of fixed-length entries produced by the trace control commands.

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. Note that 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/DOS/VS 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. 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/DOS/VS 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.

Note that 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/DOS/VS Installation and Operations Guide*.

User Trace Entry Point and Event Monitoring Point (ENTER)

The ENTER 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).

```
ENTER
TRACEID(data-value)
[FROM(data-area)]
[RESOURCE(name)]
[ENTRYNAME(name)]
[ACCOUNT]
[MONITOR]
[PERFORM]
```

Condition: INVREQ

The ENTER 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 390 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.

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 you should 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.

| Note that 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 and all of the CICS tables.

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 partition. 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.
- |
- 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.

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 data base 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/DOS/VS Resource Definition (Macro)* 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 31.

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 (LI.bb), 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 started writing to 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 command 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 NOBUFS 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, NOBUFS, 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 400. 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, since 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 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 NOJBUFFSP condition is to be inhibited. This condition will be handled as described on page 31.

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/DOS/VS 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/DOS/VS Facilities and Planning Guide* and the *CICS/DOS/VS 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, you should not rely on this because a logical message whose first page is incomplete will be lost. You should also code an explicit SEND PAGE command before the SYNCPOINT command or before termination of the transaction.

The system programmer should be consulted 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 data base 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.

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. Note that 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. Note that a field of one 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

Note that a decimal point is an EBCDIC special character and as such is removed.

There are no exceptional conditions with DEEDIT.

Appendices

- 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 G Report Controller Sample Programs**

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 41. 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 alphabetical 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 POWER, are primarily for the use of the system programmer; they are not described in this book. For details of the commands, refer to the *CICS/DOS/VS 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	1C02	DUMP
043A	ISSUE PASS	1E02	ISSUE ADD
043C	EXTRACT LOGONMSG	1E04	ISSUE ERASE
0602	READ	1E06	ISSUE REPLACE
0604	WRITE	1E08	ISSUE ABORT
0606	REWRITE	1E0A	ISSUE QUERY
0608	DELETE	1E0C	ISSUE END
060A	UNLOCK	1E0E	ISSUE RECEIVE
060C	STARTBR	1E10	ISSUE NOTE
060E	READNEXT	1E12	ISSUE WAIT
0610	READPREV	1E14	ISSUE SEND
0612	ENDBR	2002	BIF DEEDIT
0614	RESETBR	2202	ENABLE
0802	WRITEQ TD	2204	DISABLE
0804	READQ TD	2206	EXTRACT EXIT
0A02	WRITEQ TS	4A02	ASKTIME ABSTIME
0A04	READQ TS	4A04	FORMATTIME
CA06	DELETEQ TS	4C02	INQUIRE DATASET
0C02	GETMAIN	4C04	SET DATASET
0C04	FREEMAIN	4E02	INQUIRE PROGRAM
0E02	LINK	4E04	SET PROGRAM
0E04	XCTL	5002	INQUIRE TRANSACTION
0E06	LOAD	5004	SET TRANSACTION
0E08	RETURN	5202	INQUIRE TERMINAL
0E0A	RELEASE	5204	SET TERMINAL
0E0C	ABEND	5206	INQUIRE NETNAME
0E0E	HANDLE ABEND	5402	INQUIRE SYSTEM
1002	ASKTIME	5404	SET SYSTEM
1004	DELAY	5602	SPOOLOPEN
1006	POST	5604	SPOOLREAD
1008	START	5606	SPOOLWRITE
100A	RETRIEVE	5610	SPOOLCLOSE
100C	CANCEL	5620	SPOOLOPEN REPORT
1202	WAIT EVENT	5622	SPOOLOPEN MAPNAME
1204	ENQ	5624	SPOOLOPEN ESCAPE
1206	DEQ	5626	SPOOLOPEN RESUME
1208	SUSPEND	5628	SPOOLWRITE REPORT
1402	JOURNAL	562A	SPOOLWRITE MAPNAME
1404	WAIT JOURNAL	562C	SPOOLCLOSE REPORT
1602	SYNCPOINT	5802	INQUIRE CONNECTION
1802	RECEIVE MAP	5804	SET CONNECTION
1804	SEND MAP	5A02	INQUIRE MODENAME
1806	SEND TEXT	5A04	SET MODENAME
1808	SEND PAGE		
180A	PURGE MESSAGE		
180C	ROUTE		
180E	RECEIVE PARTN		
1810	SEND PARTNSET		
1812	SEND CONTROL		
1A02	TRACE		
1A04	ENTER		

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, refer to the *CICS/DOS/VS 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 POWER are primarily for the use of the system programmer; they are

not described in this book. However, the conditions relating to the spool commands of the CICS interface to POWER are included in the list. For details of the commands, refer to the *CICS/DOS/VS Customization Guide*.

EIBFN (Byte 0)

Byte (of EIBRCODE)		EIBRCODE Value	Condition
02	0		
02	0	E0	INVREQ
04	0	04	EOF
04	0	10	EODS
04	0	C1	EOF
04	0	C2	ENDINPT
04	0	D0	SYSIDERR ²
04	0	D2	SESSIONERR ²
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 ²
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	18 1 10	INVLD C
08 0 C0	QBUSY	18 1 20	UNEXPIN
08 0 D0	SYSIDERR ³	18 1 40	IGREQCD
08 0 D1	ISCINVREQ	18 1 80	TSIOERR
08 0 D6	NOTAUTH	18 2 01	OVERFLOW
08 0 E1	LENGERR	18 2 04	EODS
0A 0 01	ITEMERR	18 2 08	EOC
0A 0 02	QIDERR	18 2 10	IGREQID
0A 0 04	IOERR	1A 0 E0	INVREQ
0A 0 08	NOSPACE	1E 0 04	DSSTAT
0A 0 20	INVREQ	1E 0 08	FUNCERR
0A 0 D0	SYSIDERR ³	1E 0 0C	SELNERR
0A 0 D1	ISCINVREQ	1E 0 10	UNEXPIN
0A 0 D6	NOTAUTH	1E 0 E1	LENGERR
0A 0 E1	LENGERR	1E 1 11	EODS
0C 0 E1	LENGERR	1E 1 2B	IGREQCD
0C 0 E2	NOSTG	1E 2 20	EOC
0E 0 01	PGMIDERR	4A 3 01	ERROR
0E 0 D6	NOTAUTH	56 3 0D	NOTFND
0E 0 E0	INVREQ	56 3 10	INVREQ
10 0 01	ENDDATA	56 3 11	IOERR
10 0 04	IOERR	56 3 12	NOSPACE
10 0 11	TRANSIDERR	56 3 14	ENDFILE
10 0 12	TERMIDERR	56 3 15	ILLOGIC
10 0 14	INVTSREQ	56 3 16	LENGERR
10 0 20	EXPIRED	56 3 46	NOTAUTH
10 0 81	NOTFND	56 3 49	WRONGSTAT
10 0 D0	SYSIDERR ³	56 3 4A	NAMEERROR
10 0 D1	ISCINVREQ	56 3 4C	CCERROR
10 0 D6	NOTAUTH	56 3 4D	MAPERROR
10 0 E1	LENGERR	56 3 50	NOSPOOL
10 0 E9	ENVDEFERR		
10 0 FF	INVREQ		
12 0 32	ENQBUSY		
12 0 E0	INVREQ		
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		

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 1,2 = DAM response
byte 1 = VSAM return code
byte 2 = VSAM error code*

Details of these response codes are given in the VSE/SP Messages and Codes manual and in the VSE/VSAM Messages and Codes manual.

2. When *INVMPSZ* occurs during *BMS* operations, byte 3 of field *EIBRCODE* contains the terminal code; see "Terminal Code Table" on page 202.

3. When SESSIONERR or SYSIDERR occurs, further information is provided in byte 1 of EIBRCODE, as follows:

04 Name not that of system entry.
08 Link out of service.
0C Name unknown to CICS.

Further information on SYSIDERR can be found in the CICS/DOS/VS Problem Determination Guide.

4. 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.

5. When LENERR 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)

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 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 POWER, are primarily for the use of the system programmer; they are not described in this book. For details of the commands, refer to the CICS/DOS/VS Customization Guide. However, the EIBRESP codes for these commands are included in the following list.

ASM: F
COBOL: PIC S9(8) 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 INVLDCL

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

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 S9(8) 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 POWER, are primarily for the use of the system programmer; they are not described in this book. For details of these commands, refer to the *CICS/DOS/VS Customization Guide*. For details of the commands that are used with the separately ordered feature called the report controller, see page 293.

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 No.	ENGRAVING Top(LC) Front(UC)	LINE Code	PROCESSOR CODE Numeric(LC) Alpha(UC)	HLL ID
0	MSG ACK	1	AA	
1	SEND AGAIN	Q	D9	
2	CORR	A	C1	
3	HOLD OVRDE	2	C8	
4	VOID	Z	F2	
5	ACCT INQ	W	E9	
6	ACCT TFR	S	E6	
7	CIF	S	E2	
8	MISC	X	F3	
9	CLSD ACCT	E	E7	
10	NO BOOK	D	C5	234
11	MORT LOAN	4	AE	5
12		C	AF	6
13	NEW ACCT	R	C4	7
14	BOOK BAL	F	B0	8
15	INST LOAN	5	B1	9
16	SPEC TRAN	V	B2	10
17	SAV BOND	T	B3	11
18	SAV	G	B4	12
19	XMAS CLUB	T	B5	13
20	*	G	B6	14
21	DDA	B	B7	
22	00	Y	B8	
23	MON ORD	H	B9	
24	0	7	BA	
25	7	N	F0	15
26	4	U	F7	16
27	CSHR CHK	J	D5	17
28	1	M	E4	
29	8	I	D1	
30	5	K	F8	18
31	CASH RECD	9	D4	
32	2	M	C9	
33	9	I	D2	19
34	6	K	F5	
35	UTIL BILL	9	D6	
36	3	O	F5	
37	DEP +	0	E4	
38	WITH -	O	F3	
39	FEES	O	4B	
40	TOTL	O	D7	
41	CASH IN	O	5B	
42	CASH CHK	O	60	
43	VAL	O	C6	
44	TAB	O	61	
45	ALPHA ENTRY	O	E3	
46	NUM ENTRY	O	BD	20
47	SEND	O	BE	21
48	RETURN	0	5C	
49	NUM ENTRY	6	50	
50	SPACE	6	05	
58	MSGLIGHT	0	05	TABCHAR
		26-ETB 03-ETX	STATION ID 05	
			15	JRNLCR
			40	MSGLITE
			17	

Figure 22. 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	Z	F2	F2 (2)	4C (<)
4	W	E9	A9 (z)	E9 (Z)
5	S	E6	A6 (w)	E6 (W)
6	S	E2	A2 (s)	E2 (S)
7	;	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	:	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	Z	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	!	F6	F6 (6)	7D (')
20	Y	C2	82 (b)	C2 (B)
21	H	E8	A8 (y)	E8 (Y)
22	>	C8	88 (h)	C8 (H)
23	N	F7	F7 (7)	6E (>)
24	U	D5	95 (n)	D5 (N)
25	J	E4	A4 (u)	E4 (U)
26	X	D1	91 (j)	D1 (J)
27	M	F8	F8 (8)	5C (*)
28	I	D4	94 (m)	D4 (M)
29	K	C9	89 (i)	C9 (I)
30	L	D2	92 (k)	D2 (K)
31	;	F9	F9 (9)	4D (()
32	,	6B	6B (,)	4F ()
33	O	D6	96 (o)	D6 (O)
34	L	D3	93 (1)	D3 (L)
35)	F0	F0 (0)	5D (()
36	0	4B	4B (.)	5F (~)
37	-	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
48	RETURN	15	15	15
49	SHIFT	06	06	06
50	(SPACE)	40	40	40
53	SEND	26-ETB 03-ETX		BCKSPACE

Figure 23. 2980-2 Administrative Station Character Set/Translate Table

KEY No.	ENGRAVING Top(LC) Front(UC)	LINE Code	PROCESSOR CODE Numeric(LC) Alpha(UC)	HLL ID	
0	CK \$	- Q A O Z W S I X E D 2 C R F 3 V T G 4 B Y H 5 N U J 6 N I K 7	D9 D3 C1 C9 E9 E6 E2 5B E7 C5 C4 4B C3 60 C6 E8 E5 E3 C7 5C C2 61 D7 D8 D5 E4 D1 C8 D4 D6 D2 F7 6B F4 1 8 0 5 2 9 40 41 42 43 44 45 46 47	BC D3 C1 B7 4B 5C 5B 4F AE C5 6F BF C3 60 C6 BB A0 A1 C7 BE C2 61 D7 B2 D5 AF D1 7B E7 D6 D2 F7 BLANK F4 F1 F8 F0 F5 F2 F9 B0 F6 F3 50 05 36 06 26-ETB 03-ETX	19
1					
2	CK #				
3					
4					
5					
6					
7	IMD 2				
8					
9					
10					
11	IMD 1				
12					
13					
14					
15	CODE				
16					
17					
18	AMT				
19					
20					
21					
22	OB				
23					
24					
25					
26					
27	ACCT #				
28					
29					
30					
31	7				
32	..				
33	4				
34	1				
35	8				
36	0				
37	5				
38	2				
39	9				
40	..				
41	6				
42	3				
43	VAL				
44	TAB				
45	ALPHA				
46	NUMERIC				
47	SEND				
48	RETURN		15	15	
49	NUMERIC		06		
50	SPACE		40	40	
51	FEED OPEN		04	OPENCH	

Figure 24. 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	LIST=SEGMENT TYPE=TRANSACTION	DUMP FROM DUMP[TASK]
DFHBFTA	-	DFHDI TYPE=ABORT TYPE=ADD TYPE=CHECK TYPE=END TYPE=ERASE TYPE=NOTE TYPE=QUERY TYPE=RECEIVE TYPE=REPLACE TYPE=SEND TYPE=WAIT	ISSUE ABORT ISSUE ADD HANDLE CONDITION ISSUE END ISSUE ERASE ISSUE NOTE ISSUE QUERY ISSUE RECEIVE ISSUE REPLACE ISSUE SEND ISSUE WAIT
DFHBIF TYPE=DEEDIT	BIF DEEDIT	DFHFC TYPE=CHECK TYPE=DELETE (DL/I types) TYPE=ESETL TYPE=GET TYPE=GET, TYPOPER=UPDATE TYPE=GETAREA TYPE=GETNEXT TYPE=GETPREV TYPE=PUT, TYPOPER=DELETE TYPE=PUT, TYPOPER=NEWREC TYPE=PUT, TYPOPER=UPDATE TYPE=RELEASE SEND {MAP TEXT} SET ROUTE SEND {MAP TEXT} PAGING SEND TEXT ACCUM	HANDLE CONDITION DELETE RIDFLD - ENDBR READ READ UPDATE - READNEXT READPREV DELETE WRITE REWRITE UNLOCK RESETBR STARTBR
DFHBMS TYPE=CHECK TYPE=IN TYPE=MAP TYPE=OUT TYPE=OUT,MAP= TYPE=PAGEBLD TYPE=PAGEOUT TYPE=PURGE TYPE=RETURN TYPE=ROUTE TYPE=STORE TYPE=TEXTBLD	HANDLE CONDITION RECEIVE MAP RECEIVE MAP FROM SEND TEXT SEND MAP SEND MAP ACCUM SEND PAGE PURGE MESSAGE SEND {MAP TEXT} SET ROUTE SEND {MAP TEXT} PAGING SEND TEXT ACCUM	DFHIC TYPE=CANCEL TYPE=CHECK TYPE=GET TYPE=GETIME TYPE=INITIATE TYPE=POST TYPE=PUT TYPE=RETRY TYPE=WAIT	CANCEL HANDLE CONDITION RETRIEVE ASKTIME START POST START FROM RETRIEVE DELAY
DFHDC TYPE=CICS TYPE=COMPLETE TYPE=PARTIAL LIST=PROGRAM LIST=TERMINAL LIST=TRANSACTION	DUMP TABLES DUMP COMPLETE DUMP PROGRAM DUMP TERMINAL DUMP STORAGE		

DFHJC TYPE=CHECK TYPE=GETJCA TYPE=PUT TYPE=WAIT TYPE=WRITE	HANDLE CONDITION - JOURNAL WAIT WAIT JOURNAL JOURNAL	TYPE=COPY TYPE=DISCONNECT TYPE=EODS TYPE=ERASEAUP TYPE=GET TYPE=PAGE TYPE=PASSBK TYPE=PRINT TYPE=PROGRAM TYPE=PUT TYPE=READ TYPE=READB TYPE=READL TYPE=RESET TYPE=SIGNAL TYPE=WAIT TYPE=WRITE TYPE=WRITEL	ISSUE COPY ISSUE DISCONNECT ISSUE EODS ISSUE ERASEAUP RECEIVE - SEND PASSBK ISSUE PRINT ISSUE LOAD SEND WAIT RECEIVE(WAIT assumed) RECEIVE BUFFER RECEIVE LEAVEKB ISSUE RESET WAIT SIGNAL WAIT TERMINAL SEND SEND LEAVEKB
DFHKC TYPE=ATTACH TYPE=CHAP TYPE=DEQ TYPE=ENQ TYPE=NOPURGE TYPE=PURGE TYPE=WAIT TYPE=WAIT, ECADDR	- - DEQ ENQ - - SUSPEND WAIT EVENT	TYPE=READ TYPE=READB TYPE=READL TYPE=RESET TYPE=SIGNAL TYPE=WAIT TYPE=WRITE TYPE=WRITEL	RECEIVE(WAIT assumed) RECEIVE BUFFER RECEIVE LEAVEKB ISSUE RESET WAIT SIGNAL WAIT TERMINAL SEND SEND LEAVEKB
DFHMDF	-	DFHTD TYPE=CHECK TYPE=FEOV TYPE=GET TYPE=PURGE TYPE=PUT	HANDLE CONDITION -
DFHMDI	-		READQ TD
DFHMSD	-		DELETEQ TD WRITEQ TD
DFHPC TYPE=ABEND TYPE=CHECK TYPE=COBADDR TYPE=DELETE TYPE=LINK TYPE=LOAD TYPE=RESETXIT TYPE=RETURN TYPE=SETXIT TYPE=XCTL	ABEND HANDLE CONDITION - RELEASE LINK LOAD HANDLE ABEND RESET RETURN HANDLE ABEND XCTL	DFHTR TYPE=ENTRY TYPE=OFF TYPE=ON DFHTS TYPE=CHECK TYPE=GET ¹ TYPE=GETQ TYPE=PURGE TYPE=PUT ¹ TYPE=PUTQ TYPE=RELEASE	ENTER TRACE OFF TRACE ON HANDLE CONDITION READQ TS READQ TS DELETEQ TS WRITEQ TS WRITEQ TS DELETEQ TS
DFHSC TYPE=FREEMAIN TYPE=GETMAIN	FREEMAIN GETMAIN		
DFHSP TYPE=USER TYPE=ROLLBACK	SYNCPOINT SYNCPOINT ROLLBACK		
DFHTC TYPE=CBUFF TYPE=CONVERSE	SEND CBUFF CONVERSE		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/DOS/VS Installation and Operations 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 HIGHLIGHT 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.

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 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 below. The plus (+) sign in this and subsequent displays shows the position of the attribute byte. In an actual display, this position contains a blank.

+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+

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$AMNU - CICS/VS SAMPLE FILEA OPERATOR INSTRUCTION ME*  
NU - ASSEMBLER'  
DFH$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. You should insert editing steps needed to ensure valid changes to the file. Those fields which 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 the Inquiry/Update Sample Program

```
TITLE 'DFH$AALL - CICS/VS SAMPLE FILEA INQUIRY/UPDATE - ASSEMBLER'

DFHEISTG DSECT
COPY DFH$AGA
COPY DFH$AGB
RETREG EQU 2
R06 EQU 6
R07 EQU 7
R08 EQU 8
R09 EQU 9
FILEDS DS 0C
COPY DFH$AFIL
COMPTR EQU 4
COPY DFH$ALOG
COPY DFHBMSCA
MESSAGES DS CL39
KEYNUM DS CL9
COMLEN DS 1H
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'?
```

MAP A
MAP B
SET UP REGISTER USAGE

RECORD DESCRIPTION FOR FILEA
pointer to COMMAREA
LOG FILE RECORD DESCRIPTION
BMS ATTRIBUTE BYTES
TEMP STORE FOR MESSAGES
TEMP STORE FOR FILE RECORD KEY
LENGTH OF COMMAREA

Source Listing for the Inquiry/Update Sample Program (Continued)

```

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   OH          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 MAPC('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   TITLE0,=CL(L'TITLE0)'FILE ADD' SET UP TITLE
MVC   MSG30,=CL(L'MSG30)'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   AMOUNT0,=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   OH          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
10    MVC   TITLE0,=CL(L'TITLE0)'FILE INQUIRY' SET UP TITLE ON MAP
MVC   MSG30,=CL(L'MSG30)'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   OH          UPDATE ROUTINE
13    MVC   TITLE0,=CL(L'TITLE0)'FILE UPDATE' SET UP MAP TITLE
MVC   MSG30,=CL(L'MSG30)'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
*                 HERE A COMMAREA HAS BEEN RETURNED, AND IS THEREFORE SECOND*****
*                 INVOCATION OF THIS PROGRAM
*                 HERE A COMMAREA HAS BEEN RETURNED, AND IS THEREFORE SECOND*****
*                 INVOCATION OF THIS PROGRAM
COMRETND DS   OH          HERE COMMAREA HAS BEEN RETURNED
16    L     COMPTR,DFHEICAP  GET ADDRESSABILITY TO COMMAREA
17    EXEC CICS HANDLE CONDITION MAPFAIL(NOTMODF) ERROR(ERRORS)

```

Source Listing for the Inquiry/Update Sample Program (Continued)

```

        DUPREC(DUPREC) NOTFND(NOTFOUND)
18      EXEC CICS RECEIVE MAP('DFH$AGB')
        CLC  EIBTRNID,=CL(L'EIBTRNID)'AUPD' IS INVOKING T-ID AUPD?
        BNE  SECADD    ..NO, SO BRANCH TO SECOND ADD ROUT
19      EXEC CICS READ UPDATE DATASET('FILEA') INTO(FILEA)      *
        RIDFLD(NUMB-FILEDS(COMPTR))
        CLC  FILEREC,FILEREC-FILEDS(COMPTR) RECORD CHANGED ON FILE?
        BE   OKREC     ..NO, SO BRANCH AND CONTINUE
20      MVC  MSG10,=CL(L'MSG10)'RECORD UPDATED BY OTHER USER, TRY AGA*
        IN'
        MVI  MSG1A,DFHBMASB    BRIGHTEN MESSAGE ON SCREEN
        MVI  MSG3A,DFHPROTN   DARK AND PROTECTED ATTRIBUTE
        BAL  RETREG,MAPBUILD  GO BUILD MAP
        EXEC CICS SEND MAP('DFH$AGB') DATAONLY
        MVC  COMLEN,=H'80'     SET UP LENGTH OF COMMERA
        B    CICSCONT      GO RETURN CONTROL TO CICS
OKREC   DS   OH          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.
SECADD   DS   OH          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   OH          THIS ROUTINE RETURNS CONTROL TO CICS
27      EXEC CICS RETURN TRANSID(EIBTRNID) COMMAREA(FILEDS)      *
        LENGTH(COMLEN)
        AMNU  DS   OH          ENDING ROUTINE
28      XC   DFH$AGAO(DFH$AGAE-DFH$AGAO),DFH$AGAO CLEAR MAP
        MVI  MSGA,DFHBMASB    BRIGHTEN MESSAGE FIELD ON MAP
        MVC  MSG0,MESSAGES   MOVE ANY MESSAGE TO MAP AREA
29      EXEC CICS SEND MAP('DFH$AGA') ERASE
30      EXEC CICS RETURN
*****
*                                     *
*          GENERAL ROUTINES          *
*                                     *
*****
MAPBUILD DS   OH          ROUTINE TO BUILD MAP DFH$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   OH          ROUTINE TO SEND MAP DFH$AGB
32      EXEC CICS SEND MAP('DFH$AGB') ERASE
        BR   RETREG        RETURN
CHECK    DS   OH          ANY INPUT FROM SCREEN? ROUTINE
33      LA   R06,DFH$AGB0    R6 POINTS TO START OF MAP DFH$AGB
        LA   R07,(DFH$AGBE-DFH$AGB0) R7 CONTAINS LENGTH OF MAP B
        LA   R08,HEXZERO     R8 POINTS TO HEXZERO

```

Source Listing for the Enquiry/Update Sample Program (Continued)

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 'AADD'?
BE	ADNAMCHK	.YES, SO GO TO 'AADD' NAME CHECK
UPNAMCHK	DS OH	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 OH	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 ADDR1,ADDR1	HAS ADDRESS BEEN ENTERED?
	BZ PHNTST	.NO, BRANCH
	MVC ADDR2,ADDR2	.YES, PUT IN IN FILE AREA
PHNTST	OC PHONE1,PHONE1	HAS PHONE BEEN ENTERED?
	BZ DATTST	.NO, BRANCH
	MVC PHONE,PHONE1	.YES, PUT IN IN FILE AREA
DATTST	OC DATE1,DATE1	HAS DATE BEEN ENTERED?
	BZ AMTTST	.NO, BRANCH
	MVC DATEX,DATE1	.YES, PUT IN IN FILE AREA
AMTTST	OC AMOUNT1,AMOUNT1	HAS AMOUNT BEEN ENTERED?
	BZ CHEKTRAN	.NO, BRANCH
	TRT AMOUNT1,CHEKTAB	IS AMOUNT NUMERIC
	BNZ DATAERR	NO, ASK FOR CORRECT AMOUNT
	MVC AMOUNT,AMOUNT1	.YES, PUT IN IN FILE AREA
	B COMTST	
CHEKTRAN	CLC EIBTRNID,=CL(L'EIBTRNID)'AADD'	IS INVOKING T-ID 'AADD'?
	BNE COMTST	
*		PUT VALID AMOUNT IN NEW RECORD
COMTST	MVC AMOUNT,=CL8'\$0000.00'	
	OC COMMENT1,COMMENT1	HAS COMMENT BEEN ENTERED?
	BZ CONTINUE	.NO, CONTINUE
	MVC COMMENT,COMMENT1	.YES, PUT IN IN FILE AREA
CONTINUE	DS OH	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 OH	ADD FUNCTION REQUIRED
38	EXEC CICS WRITE DATASET('FILEA') FROM(FILEA)	*
	RIDFLD(NUMB-FILEDS(COMPTR))	
	BR RETREG	FINISHED, SO RETURN
DATAERR	DS OH	GENERAL ROUTINES
39	MVI NAMEA,DFHBMFSE	PRESERVE CONTENTS OF SCREEN
	MVI ADDR1,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

Source Listing for the Inquiry/Update Sample Program (Continued)

```

40      MVI    MSG1A, DFHPROTN      DARK AND PROTECTED ATTRIBUTE
41      MVC    MSG30, =CL(L'MSG30)'DATA ERROR - CORRECT AND PRESS ENTER'
41      EXEC   CICS SEND MAP('DFH$AOB') DATAONLY
41      CLC    EIBTRNID, =CL(L'EIBTRNID)'AUPD' UPDATE REQUIRED?
41      BE     UPDTERR          ..YES, SO BRANCH
42      MVC    COMLEN, =H'7'      ..NO, SET UP COMLEN
42      B      CICSCONT        GO RETURN CONTROL TO CICS
42      UPDTERR DS    OH
42      MVC    COMLEN, =H'80'     UPDATE, SET UP REQUIRED COMLEN
42      B      CICSCONT
43      NOTMODF DS    OH          SCREEN NOT CHANGED
43      MVC    MESSAGES, =CL(L'MESSAGES)'RECORD NOT MODIFIED' MESSAGE
43      B      AMNU           COMPLETE, GO FINISH
43      DUPREC  DS    OH          DUPLICATE RECORD
43      MVC    MESSAGES, =CL(L'MESSAGES)'DUPLICATE RECORD' MESSAGE
43      B      AMNU           COMPLETE, GO FINISH
43      BADLENG DS    OH
43      MVC    MESSAGES, =CL(L'MESSAGES)'PLEASE ENTER AN ACCOUNT NUMBER'
43      B      AMNU
43      BADCHARS DS    OH
43      MVC    MESSAGES, =CL(L'MESSAGES)'ACCOUNT NUMBER MUST BE NUMERIC'
43      B      AMNU
43      NOTFOUND DS    OH          RECORD NOT FOUND
43      MVC    MESSAGES, =CL(L'MESSAGES)'INVALID NUMBER - PLEASE REENTER'
43      B      AMNU           COMPLETE, GO FINISH
43      MFAIL   DS    OH
43      MVC    MESSAGES, =CL(L'MESSAGES)'PRESS CLEAR TO EXIT'
43      B      AMNU
44      ERRORS  DS    OH          GENERAL ERROR ROUTINE
44      EXEC   CICS DUMP DUMPCODE('ERRS')
44      MVC    MESSAGES, =CL(L'MESSAGES)'TRANSACTION TERMINATED'
44      B      AMNU           COMPLETE, GO FINISH
44      HEXZERO DS    X'00'        CONSTANT FOR COMPARISONS
44      TAB    DC    256X'FF'      TRANSLATE TABLE FOR NAME
44      ORG   TAB+C' '
44      DC    X'00'
44      ORG   TAB+C'.'        CHAR '.'
44      DC    X'00'
44      ORG   TAB+C'--'       CHAR '--'
44      DC    X'00'
44      ORG   TAB+C''''       CHAR QUOTE
44      DC    X'00'
44      ORG   TAB+C'A'        CHARS 'A' - 'I'
44      DC    9X'00'
44      ORG   TAB+C'J'        CHARS 'J' - 'R'
44      DC    9X'00'
44      ORG   TAB+C'S'        CHARS 'S' - 'Z'
44      DC    8X'00'
44      ORG
44      CHEKTAB DS    256X'FF'      TRANSLATE TABLE FOR AMOUNT
44      ORG   CHEKTAB+C'¤'      ALTERNATE CURRENCY
44      DC    X'00'
44      ORG   CHEKTAB+C'.'      ALLOW FULL STOP
44      DC    X'00'
44      ORG   CHEKTAB+C'$'      ALLOW CURRENCY
44      DC    X'00'
44      ORG   CHEKTAB+C'0'      NUMBERS 0-9
44      DC    10X'00'
44      ORG
44      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 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 the Browse Sample Program

```
TITLE 'DFH$ABRW - CICS/VС SAMPLE FILEA BROWSE - ASSEMBLER'

DFHEISTG DSECT
HEXZERO DS X'00'
CURROP DS CL1
LASTOP DS CL1
STATUS DS CL1      FILE STATUS - HI OR LO END OR NORMAL (H/L/N)
FLAGS DS X
PFIRST EQU X'80'
KEYS DS OCL18
RID DS CL6
RIDB DS CL6
RIDF DS CL6
MESSAGES DS CL80
*
COPY DFHBMSCA
COPY DFH$AFIL
COPY DFH$AGA
COPY DFH$AGC
*
DFH$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 KEY1,=H'0'          WAS ACCOUNT NUMBER OMITTED?
    BE NOACCTNUM            YES - FRONT OF FILE BY DEFAULT.
*
6   TRT KEY1,CHEKTAB
    BNZ BADCHARS            CHECK ACCOUNT NUMBER IS NUMERIC,
                            IT ISN'T - GO DISPLAY MESSAGE.
```

Source Listing for the Browse Sample Program (Continued)

```

MVC RID,KEYI
CLC RID,=C'999999'      IF ACCOUNT NUMBER IS MAXIMUM
BNE SETRID
MVC RID,=6X'FF'          SET RECORD KEY HIGH TO
SETRID  MVC RIDF,RID
MVC RIDB,RID
B  BRWSNOW
*x
BADCHARS MVC MESSAGES,=CL(L'MESSAGES)'ACCOUNT NUMBER MUST BE NUMERIC'
B  AMNU
*x
NOACCCNUM MVC RID,=C'000000'
MVC RIDF,=C'000000'      0 DIGITS ENTERED
B  BRWSNOW
*****ESTABLISH START POINT*****
*x
*****ESTABLISH START POINT*****
*x
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
*x
OI  FLAGS,PFIRST
MVI STATUS,C'H'          AND
B  PAGEB                PAGE BACKWARDS
*****BUILD NEXT FORWARD PAGE*****
*x
*****BUILD NEXT FORWARD PAGE*****
*x
PAGEF   DS 0H
MVI CURROP,C'F'          IN CASE PF1 WAS USED
8  EXEC CICS HANDLE CONDITION
ENDFILE(TOOHIGH)
*x
LA  4,1
LA  6,DFH$AGCO
LA  7,(DFH$AGCE-DFH$AGCO)
LA  8,HEXZERO
LA  9,L'HEXZERO
ICM 9,B'100',HEXZERO
MVCL 6,8
*x
MVC RID,RIDF
*x
NEXTLINE DS 0H
9  EXEC CICS READNEXT
INTO(FILEA)
DATASET('FILEA')
RIDFLD(RID)
*x
10  CH  4,=H'1'          FIRST LINE?
BNE NEXT2
MVC NUMBER10,NUMB
MVC NAME10,NAME
MVC AMOUNT10,AMOUNT
MVC RIDB,RID
B  NEXTCONT
*x
NEXT2   DS 0H

```

Source Listing for the Browse Sample Program (Continued)

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	
*	NEXT3	DS	0H	THIRD LINE?
	CH	4,=H'3'	.NO, TEST FOR FOURTH LINE	
	BNE	NEXT4	MOVE NUMBER TO MAP AREA	
	MVC	NUMBER30,NUMB	MOVE NAME TO MAP AREA	
	MVC	NAME30,NAME	MOVE AMOUNT TO MAP AREA	
	B	NEXTCONT	GOTO NEXTCONT	
*	NEXT4	DS	0H	FOURTH LINE?
	CH	4,=H'4'	.NO, GOTO NEXTCONT	
	BNE	NEXTCONT	MOVE NUMBER TO MAP AREA	
	MVC	NUMBER40,NUMB	MOVE NAME TO MAP AREA	
	MVC	NAME40,NAME	MOVE AMOUNT TO MAP AREA	
	MVC	AMOUNT40,AMOUNT		
*	NEXTCONT	DS	0H	INCREMENT COUNT
	LA	4,1(,4)	FINISHED?	
	CH	4,=H'5'	.NO, GO BUILD NEXTLINE	
	BNE	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'		
	EXEC	CICS HANDLE CONDITION		
		ENDFILE(TOOLOW)	IN CASE PF2 WAS USED	
*		LA	4,1	CLEAR MAP
		LA	6,DFH\$AGCO	SET COUNTER TO 1
		LA	7,(DFH\$AGCE-DFH\$AGCO)	R6->START OF MAP DFH\$AGC
		LA	8,HEXZERO	R7-> LENGTH OF DFH\$AGC
		LA	9,L'HEXZERO	R8-> X'00'
		ICM	9,B'100',HEXZERO	R9-> LENGTH OF HEXZERO
		MVCL	6,8	X'00' INTO TOP BYTE OF R9
				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
*		CLI	STATUS,C'H'	YES - DO EXTRA READPREV
	BE	PREVLINE	- UNLESS AT HI EOF	
PREVXTRA	DS	0H		
	EXEC	CICS READPREV		
		INTO(FILEA)		
		DATASET('FILEA')		
		RIDFLD(RID)		

Source Listing for the Browse Sample Program (Continued)

```

* PREVLINE DS 0H
13 EXEC CICS READPREV
           INTO(FILEA)
           DATASET('FILEA')
           RIDFLD(RID)

*          CH 4,=H'4'
          BNE PREV2
          MVC NUMBER10,NUMB
          MVC NAME10,NAME
          MVC AMOUNT10,AMOUNT
          B  PREVCONT

          PUT FIELDS IN ASCENDING ORDER
          FOURTH LINE?
          ..NO, GO TEST FOR 3RD LINE
          MOVE NUMBER TO MAP AREA
          MOVE NAME TO MAP AREA
          MOVE AMOUNT TO MAP AREA
          GOTO PREVCONT

* PREV2   DS 0H
          CH 4,=H'3'
          BNE PREV3
          MVC NUMBER20,NUMB
          MVC NAME20,NAME
          MVC AMOUNT20,AMOUNT
          B  PREVCONT

          THIRD LINE?
          ..NO, TEST FOR 2ND LINE
          MOVE NUMBER TO MAP AREA
          MOVE NAME TO MAP AREA
          MOVE AMOUNT TO MAP AREA
          GOTO PREVCONT

* PREV3   DS 0H
          CH 4,=H'2'
          BNE PREV4
          MVC NUMBER30,NUMB
          MVC NAME30,NAME
          MVC AMOUNT30,AMOUNT
          B  PREVCONT

          SECOND LINE?
          ..NO, TEST FOR FIRST LINE
          MOVE NUMBER TO MAP AREA
          MOVE NAME TO MAP AREA
          MOVE AMOUNT TO MAP AREA
          GOTO PREVCONT

* PREV4   DS 0H
          CH 4,=H'1'
          BNE PREVCONT
          MVC NUMBER40,NUMB
          MVC NAME40,NAME
          MVC AMOUNT40,AMOUNT

          FIRST LINE?
          ..NO, PREVCONT
          MOVE NUMBER TO MAP AREA
          MOVE NAME TO MAP AREA
          MOVE AMOUNT TO MAP AREA

* PREVCONT DS 0H
          LA 4,1(,4)
          CH 4,=H'5'
          BNE PREVLINE

          INCREMENT COUNT
          FINISHED?
          ..NO, GO BUILD NEXT LINE

*          MVC RIDB,RID
          TM FLAGS,PFIRST
          BO HIGHMSG

          RIDB->NEXT FORWARD PAGE
          DID WE START AT HI-EOF?
          YES TELL HIM.

          EXEC CICS SEND MAP('DFH$AGC') ERASE
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
*          RECEIVE NEXT PAGING REQUEST
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

RECEIVE DS 0H
         NI FLAGS,X'FF'-PFIRST
         MVC LASTOP,CURROP
         EXEC CICS RECEIVE MAP('DFH$AGC')
         CLI DDIRI,C'F'
         BE PAGEF
         CLI DDIRI,C'B'

         SET OF FIRST TIME FLAG.
         REMEMBER LAST OPERATION

14          PAGE FORWARD REQUIRED?
         ..YES, GO TO PAGEF ROUTINE
         PAGE BACK REQUIRED?

         BE PAGEB
         ..YES, GO TO PAGEB ROUTINE
         EXEC CICS SEND CONTROL FREEKB FRSET. IGNORE - RESET KEYBOARD.

```

Source Listing for the Browse Sample Program (Continued)

```
B      RECEIVE
*****HANDLE END OF FILE CONDITIONS*****
TOOHIGH DS  OH
15    MVI  STATUS,C'H'           SET STATUS 'HI END'
      MVC  RIDF,RID
      MVC  RIDB,RID
      MVI  DIRO,X'40'
HIGHMSG MVC  MSG10,=CL(L'MSG10)'HI END OF FILE'
      MVC  MSG1A,DFHBMASB
      EXEC CICS SEND MAP('DFH$AGC') ERASE
      B    RECEIVE
*
TOOLOW DS  OH
16    MVI  STATUS,C'L'           SET STATUS 'LO END'
      MVC  RIDF,=C'000000'
      MVC  RIDB,=C'000000'
      MVI  DIRO,X'40'
      MVI  MSG2A,DFHBMASB
      MVC  MSG20,=CL(L'MSG20)'LO END OF FILE'
      EXEC CICS SEND MAP('DFH$AGC') ERASE
      B    RECEIVE
*****HANDLE GENERAL CONDITIONS*****
NOTFOUND DS  OH
17    MVC  MESSAGES,=CL(L'MESSAGES)'END OF FILE - PLEASE RESTART '
      B    AMNU
*
SMSG   DS  OH
18    MVC  MESSAGES,=CL(L'MESSAGES)'PRESS CLEAR TO EXIT'
      B    AMNU
*
ERRORS DS  OH
19    EXEC CICS DUMP DUMPCODE('ERRS')
      MVC  MESSAGES,=CL(L'MESSAGES)'TRANSACTION TERMINATED'
*****DISPLAY GENERAL MENU THEN EXIT*****
AMNU   DS  OH
20    XC   DFH$AGAO(DFH$AGAE-DFH$AGAO),DFH$AGAO  CLEAR MAP A
      MVI  MSGA,DFHBMASB
      MVC  MSG0,MESSAGES
      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/DOS/VS Installation and Operations Guide*.

The trigger level may be changed using the CEMT command, as follows:

CEMT SET QUEUE(L86O) TRIGGER(n)

where n is the destination trigger level (any integer from 0 through 32767).

Source Listing for the Order Entry Sample Program

```
TITLE 'DFH$AREN - CICS/VS SAMPLE FILEA ORDER ENTRY - ASSEMBLER*'

DFHEISTG DSECT
COPY DFH$AGK
COPY DFH$AL86
COPY DFH$AFIL
COPY DFHBMSCA
*
FLAGS DS 1B
FLERR EQU X"40"
*
DFH$AREN CSECT
1 EXEC CICS HANDLE AID CLEAR(ENDA)
*
2 EXEC CICS HANDLE CONDITION MAPFAIL(MAPFAIL)
NOTFND(NOTFOUND) ERROR(ERRORS)
*
*
XC DFH$AGKO(DFH$AGKE-DFH$AGKO),DFH$AGKO
*
*
3 EXEC CICS SEND MAP('DFH$AGK') ERASE
***** PROCESS THE INPUTTED FIELDS *****
RECEIVE DS OH
*
```

MAP DEFINITION
TD Q REC DESCRIPT
FILE OF ACCOUNTS
STD BMS ATTRIBUTE

ERROR FLAGS
FLERR-INPUTCHECKS

CLEAR MAP VARS

ERASE+DISPLAY MAP

RECEIVE INPUTDATA

Source Listing for the Order Entry Sample Program (Continued)

```

4      EXEC CICS RECEIVE MAP('DFH$AGK')
      NI      FLAGS,X'FF'-FLERR
*                               SET FLERR TO 0
*                               SET MDT=1 IN CASE
*                               FIELDS NEED TO BE
*                               RE-ENTERED
*                               CHECK FOR NUMERIC
5      TRT    CUSTNOA,DFHUNNUM
      MVI    PARTNOA,DFHUNNUM
      MVI    QUANTA,DFHUNNUM
*                               ATTR=BRI+UNPROT'D+NUMERIC
*                               SET FLERR TO 1
*                               CHECK FOR NUMERIC
TSTPART TRT    PARTNOI,CHEKTAB
      BZ     TSTPART
      MVI    CUSTNOA,DFHUNINT
      OI     FLAGS,FLERR
*                               ATTR=BRI+UNPROT'D+NUMERIC
*                               SET FLERR TO 1
*                               CHECK FOR NUMERIC
TSTQUANT TRT    QUANTI,CHEKTAB
      BZ     CHKFLEERR
      MVI    QUANTA,DFHUNINT
      OI     FLAGS,FLERR
*                               ATTR=BRI+UNPROT'D+NUMERIC
*                               SET FLERR TO 1
***** SIMPLE VALIDATION OF INPUT DATA *****
*                               CHECK CUSTOMER NUMBER EXISTS
***** WRITE VALID ORDER TO TD QUEUE *****
7      QBUILD EXEC CICS READ DATASET('FILEA') INTO(FILEA) RIDFLD(CUSTNOI)
8      MVC    CUSTNO,CUSTNOI
      MVC    PARTNO,PARTNOI
      MVC    QUANTITY,QUANTI
      MVC    TERMID,EIBTRMID
***** HANDLE ERRORS AND RESTART *****
9      EXEC CICS WRITEQ TD QUEUE('L860') FROM(L860) LENGTH(22)
10     EXEC CICS SEND MAP('DFH$AGK') MAPONLY ERASE
      B      RECEIVE
*                               GET MORE INPUT
***** HANDLE ERRORS AND RESTART *****
*                               INVALID CUSTOMER ACCOUNT NO.
11     NOTFOUND DS    OH
      MVI    CUSTNOA,DFHUNINT
      MVI    PARTNOA,DFHUNNUM
      MVI    QUANTA,DFHUNNUM
      MVI    MSG1A,DFHBMASB
      EXEC CICS SEND MAP('DFH$AGK')
      B      RECEIVE
*                               GET MORE INPUT
*                               NO DATA ENTERED IN INPUT FIELDS
MAPFAIL DS    OH

```

Source Listing for the Order Entry Sample Program (Continued)

```

12      XC      DFH$AGKO(DFH$AGKE-DFH$AGKO),DFH$AGKO      CLEAR MAP
        MVI     MSG2A,DFHBMASB                         ERROR MSG=BRIGHT
        EXEC   CICS SEND MAP('DFH$AGK')
        B      RECEIVE
*****
*                                         GET MORE INPUT
*****
*                                         EXIT PROGRAM
*****
*                                         GENERAL ERROR CONDITIONS
*                                         ERROR MSG=BRIGHT
13      DS      OH
        MVI     MSG2A,DFHBMASB
        MVC     MSG20,=C' TRANSACTION TERMINATED'
        EXEC   CICS SEND MAP('DFH$AGK')
        EXEC   CICS DUMP DUMPCODE('ERRS')
        B      EXIT
*****
ENDA   DS      OH
14      EXEC   CICS SEND TEXT FROM (PRESMSG) ERASE      QUIT PROGRAM
*
        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, 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 the Order Entry Queue Print Sample Program

```
TITLE 'DFH$ACOM - CICS/VС SAMPLE FILEA ORDER ENTRY QUEUE PRINT'
- ASSEMBLER'

DFHEISTG DSECT
COPY DFH$AGL
COPY DFH$AL86
COPY DFH$AFIL
LOGORD DS 0CL92
LDATE DS PL7
LTIME DS PL7
LITEM DS CL22
COMMNT DS CL11
FILLER DS CL51
QLENGTH DS 1H      SIZE OF Q RECORD
DFH$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(L86O)LENGTH(QLENGTH)QUEUE('L860')
4 EXEC CICS READ DATASET('FILEA') INTO(FILEA) RIDFLD(CUSTNO)
```

Source Listing for the Order Entry Queue Print Sample (Continued)

```

5      CLC  AMOUNT(8),=C'$0100.00' IS ORDER VALID?
BNH  LWRITE  IF->100  BRANCH AND WRITE LOG
6      MVC  ADDR0,ADDRX  SET UP MAP
MVC  NAM0,NAME
MVC  PART0,PARTNO
MVC  NUM0,CUSTNO
MVC  LITEM,ITEM
MVC  QUANT0,QUANTITY
7      EXEC CICS SEND MAP('DFH$AGL') ERASE PRINT L80
B  QREAD  GET NEXT RECORD
LWRITE  DS  OH
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  OH
10     EXEC CICS DUMP DUMPCODE('ERRS')
B  FIN  BRANCH TO END
ENDA  DS  OH
XC  DFH$AGLO(DFH$AGLE-DFH$AGLO),DFH$AGLO  CLEAR MAP
11     MVC  TITLE0,=CL(L'TITLE0)'ORDER QUEUE IS EMPTY' SET UP TITLE
EXEC CICS SEND MAP('DFH$AGL') DATAONLY ERASE L80 PRINT
TIME  DS  OH
*          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
*          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  OH
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 which 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 the Low Balance Report Sample Program

```

TITLE 'DFH$AREP - CICS/VIS SAMPLE FILEA LOW BALANCE INQUIRY - A*'
SSEMLBL'

DFHEISTG DSECT
KEYNUM DS CL6
TERMLENG DS H
TERMDATA DS CL1
*
*
*
EDVAL DS CL3
PAGEN DS CL2
WORKREG EQU 7
RETREG EQU 4
COPY DFH$AGD
COPY DFH$AFIL
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 OH
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 ACCOUNT NUMBER TO MAP
      MVC NAMEO,NAME MOVE NAME TO MAP

```

Source Listing for the Low Balance Report Sample Program (Continued)

```

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  OH      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
ENDFILE DS  OH      END OF FILE CONDITION RAISED
10     EXEC CICS SEND MAP ('FINAL') MAPSET ('DFH$AGD') MAPONLY      *
      ACCUM PAGING
11     EXEC CICS SEND PAGE
12     EXEC CICS SEND TEXT FROM (OPINSTR) ERASE
13     EXEC CICS ENDBR DATASET('FILEA')
***** A RECEIVE COMMAND IS ISSUED TO GIVE
***** THE TERMINAL OPERATOR A CHANCE TO
***** READ THE PROMPTING MESSAGE.
***** THE TRANSACTION WILL TERMINATE WHEN
***** THE OPERATOR PRESSES THE ENTER KEY.
***** PAGING COMMANDS CAN THEN BE ISSUED.
***** NO HARM IS DONE IF THE OPERATOR
***** TYPES IN DATA BEFORE PRESSING THE
***** ENTER KEY.
14     LA    WORKREG,1
      STH  WORKREG,TERMLENG
      EXEC CICS RECEIVE INTO(TERMDATA) LENGTH(TERMLENG)
      EQU  *
15     ENDTASK
      EXEC CICS RETURN
      DS   OH
16     EXEC CICS HANDLE CONDITION ERROR
      EXEC CICS PURGE MESSAGE
      EXEC CICS ABEND ABCODEC('ERRS')
      DS   OH      PAGE BUILT HERE
17     EXEC CICS SEND MAP('FOOTING') MAPSET('DFH$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', *
                  HIGHLIGHT=UNDERLINE
          DFHMDF POS=(3,1),LENGTH=29,INITIAL='OPERATOR INSTR - ENTER AMN*x
                  U'
          DFHMDF POS=(4,1),LENGTH=38,INITIAL='FILE INQUIRY - ENTER AIN*x
                  Q AND NUMBER'
          DFHMDF POS=(5,1),LENGTH=38,INITIAL='FILE BROWSE - ENTER ABR*x
                  W AND NUMBER'
          DFHMDF POS=(6,1),LENGTH=38,INITIAL='FILE ADD - ENTER AAD*x
                  D AND NUMBER'
          DFHMDF POS=(7,1),LENGTH=38,INITIAL='FILE UPDATE - ENTER AUP*x
                  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,
                  HIGHLIGHT=REVERSE
          DFHMDF POS=(12,25),LENGTH=6,INITIAL='NUMBER'
          DFHMDF POS=(12,32),LENGTH=6,ATTRB=NUM,COLOR=GREEN,
                  HIGHLIGHT=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 DFHSAGA

```
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
MSG0   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$AGAE EQU  *      END OF MAP DEFINITION
* * * END OF MAP DEFINITION * * *
              SPACE 3
              ORG
MAP$SETAT 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+

DFHSAGB 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 DFHSAGB			
DFH\$AGBI	DS	0H	ENSURE ALIGNMENT
DFH\$AGBO	DS	0C .	INPUT MAP ORIGIN
	DS	12C .	OUTPUT MAP ORIGIN
	SPACE		TIOA PREFIX
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
ADDRD	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    OCL39 .        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    OCL39 .        INPUT DATA FIELD
MSG3O    DS    CL39 .        OUTPUT DATA FIELD
SPACE
DFH$AGBE EQU  *
*** 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: +XXXXXXX+
+DATE: +XXXXXXX+
+AMOUNT: +XXXXXXX+
+COMMENT: +XXXXXXX+
+XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
+XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

DFHSAGC 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,HIGHLIGHT=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 DFHSAGC

DFH\$AGCI	DS	0H	ENSURE ALIGNMENT
DFH\$AGCO	DS	0C .	INPUT MAP ORIGIN
	DS	12C .	OUTPUT MAP ORIGIN
	TIOA PREFIX		
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
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
NUMBER10	DS	CL6 .	OUTPUT DATA FIELD
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
NAME10	DS	CL20 .	OUTPUT DATA FIELD
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
AMOUNT10	DS	CL8 .	OUTPUT DATA FIELD
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
NUMBER20	DS	CL6 .	OUTPUT DATA FIELD
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
NAME20	DS	CL20 .	OUTPUT DATA FIELD
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
AMOUNT20	DS	CL8 .	OUTPUT DATA FIELD
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
NUMBER30	DS	CL6 .	OUTPUT DATA FIELD
NAME3L	DS	CL2 .	INPUT DATA FIELD LENGTH

DSECT generated by DFHSAGC (Continued)

```

NAME3F    DS    0C .
NAME3A    DS    C .
NAME3I    DS    0CL20 .
NAME3O    DS    CL20 .
SPACE
AMOUNT3L  DS    CL2 .
AMOUNT3F  DS    0C .
AMOUNT3A  DS    C .
AMOUNT3I  DS    0CL8 .
AMOUNT3O  DS    CL8 .
SPACE
NUMBER4L  DS    CL2 .
NUMBER4F  DS    0C .
NUMBER4A  DS    C .
NUMBER4I  DS    0CL6 .
NUMBER4O  DS    CL6 .
SPACE
NAME4L    DS    CL2 .
NAME4F    DS    0C .
NAME4A    DS    C .
NAME4I    DS    0CL20 .
NAME4O    DS    CL20 .
SPACE
AMOUNT4L  DS    CL2 .
AMOUNT4F  DS    0C .
AMOUNT4A  DS    C .
AMOUNT4I  DS    0CL8 .
AMOUNT4O  DS    CL8 .
SPACE
MSG0L    DS    CL2 .
MSG0F    DS    0C .
MSG0A    DS    C .
MSG0I    DS    0CL39 .
MSG0O    DS    CL39 .
SPACE
MSG1L    DS    CL2 .
MSG1F    DS    0C .
MSG1A    DS    C .
MSG1I    DS    0CL39 .
MSG1O    DS    CL39 .
SPACE
MSG2L    DS    CL2 .
MSG2F    DS    0C .
MSG2A    DS    C .
MSG2I    DS    0CL39 .
MSG2O    DS    CL39 .
SPACE
DFH$AGCE  EQU   * .
* * * END OF MAP DEFINITION * * *
SPACE 3
ORG
MAPSETCT EQU *      * END OF MAP SET
* * * END OF MAP SET DEFINITION * * *
SPACE 3

```

DFH\$AGC Screen Layout

<u>+FILE BROWSE</u>		
+NUMBER	+NAME	+AMOUNT
+XXXXXX	+XXXXXXXXXXXXXXXXXXXX	+XXXXXXX

+PRESS CLEAR TO END BROWSE OPERATION
+PRESS PF1 OR TYPE F TO PAGE FORWARD
+PRESS PF2 OR TYPE B TO PAGE BACKWARD

DFHSAGD 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',   *
          HIGHLIGHT=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 DFHSAGD

```
        DS  0H          ENSURE ALIGNMENT
DFH$AGDO DS  0C .
        DS 12C .
        SPACE
        DS  CL2 .
NUMBERA  DS  C .
NUMBER  DS  CL6 .
        SPACE
        DS  CL2 .
NAMEA   DS  C .
NAMEO   DS  CL20 .
        SPACE
        DS  CL2 .
AMOUNTA DS  C .
AMOUNTO DS  CL8 .
        SPACE
DFH$AGDE EQU  *          END OF MAP DEFINITION
* * * END OF MAP DEFINITION * * *
        SPACE 3
        DS  0H          ENSURE ALIGNMENT
HEADINGO DS  0C .
        DS 12C .
        SPACE
        DS  CL2 .
PAGENA  DS  C .
PAGENO  DS  CL3 .
        SPACE
HEADINGE EQU  *          END OF MAP DEFINITION
* * * END OF MAP DEFINITION * * *
        SPACE 3
        DS  0H          ENSURE ALIGNMENT
FOOTINGO DS  0C .
        DS 12C .
        SPACE
FOOTINGE EQU  *          END OF MAP DEFINITION
* * * END OF MAP DEFINITION * * *
        SPACE 3
        DS  0H          ENSURE ALIGNMENT
FINALO  DS  0C .
        DS 12C .
        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

DFHSAGK 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 DFHSAGK

```
DS    0H          ENSURE ALIGNMENT
DFH$AGKI DS  0C .          INPUT MAP ORIGIN
DFH$AGKO DS  0C .          OUTPUT MAP ORIGIN
              DS 12C .
              SPACE          TIOA PREFIX
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
MSG10  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
MSG20  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
```

DFHSAGL 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 DFHSAGL

```
        DS  0H          ENSURE ALIGNMENT
DFH$AGLO DS  0C .
        DS  12C .
        SPACE
        DS  CL2 .
TITLEA  DS  C .
        DS  CL43 .
        SPACE
        DS  CL2 .
NUMBA  DS  C .
        DS  CL6 .
        SPACE
        DS  CL2 .
NAMA   DS  C .
NAMO   DS  CL20 .
        SPACE
        DS  CL2 .
ADDRA  DS  C .
        DS  CL20 .
        SPACE
        DS  CL2 .
PARTA  DS  C .
        DS  CL6 .
        SPACE
        DS  CL2 .
QUANTA DS  C .
        DS  CL6 .
        SPACE
DFH$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	OCL80
FILEREC	DS	OCL80
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	OCL92
LOGHDR	DS	OCL12
LDAY	DS	PL4
LTIME	DS	PL4
LTERML	DS	CL4
LOGREC	DS	OCL80
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	OCL22
ITEM	DS	OCL22
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/DOS/VS Installation and Operations 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 HIGHLIGHT 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.

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 below. The plus (+) sign in this and subsequent displays

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+

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 (1) erases the screen and displays map DFH\$CGA in response to the EXEC CICS SEND MAP command, then (2) ends the program using the RETURN 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$CMNU - CICS/VIS SAMPLE FILEA OPERATOR INSTRUCTION MENU *  
*****  
IDENTIFICATION DIVISION.  
PROGRAM-ID. FILECMNU.  
ENVIRONMENT DIVISION.  
DATA DIVISION.  
*  
PROCEDURE DIVISION.  
1  EXEC CICS SEND MAP('MENU') MAPSET('DFH$CGA')  
    MAPONLY ERASE END-EXEC.  
2  EXEC CICS RETURN END-EXEC.  
GOBACK.
```

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. You should insert editing steps needed to ensure valid changes to the file. Those fields which 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 the Inquiry/Update Sample Program

```
*****  
* 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.
```

Source Listing for the Inquiry/Update Sample Program (Continued)

```

          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
   MOVE DFHCOMMAREA TO COMMAREA GO TO READ-INPUT.
3
4  EXEC CICS HANDLE CONDITION MAPFAIL(MFAIL)
   ERROR(ERRORS) END-EXEC.
5  EXEC CICS RECEIVE MAPC('MENU') MAPSET('DFH$CGA') END-EXEC.
6  IF KEYL = ZERO THEN GO TO BADLENG.
7  MOVE KEYI TO KEYNUM.
8  IF KEYI IS NOT NUMERIC THEN GO TO BADCHARS.
9  MOVE LOW-VALUES TO DETAIL0.
10 IF EIBTRNID = 'ADDS' THEN
11   MOVE 'FILE ADD' TO TITLE0
   MOVE 'ENTER DATA AND PRESS ENTER KEY' TO MSG30
12   MOVE KEYI TO NUMB IN COMMAREA, NUMBO
13   MOVE DFHBMMUNN TO AMOUNTA
   MOVE '$0000.00' TO AMOUNTO
   MOVE 7 TO COMLEN GO TO MAP-SEND.
14  EXEC CICS HANDLE CONDITION NOTFND(NOTFOUND) END-EXEC.
15  EXEC CICS READ DATASET('FILEA') INTO(FILEA) RIDFLD(KEYNUM)
   END-EXEC.

16  IF EIBTRNID = 'INQY' THEN
17    MOVE 'FILE INQUIRY' TO TITLE0
    MOVE 'PRESS ENTER TO CONTINUE' TO MSG30
*     PROTECT ALL THE MAP FIELDS
18    MOVE DFHBMPRO TO NAMEA
    MOVE DFHBMPRO TO ADDRA
19    MOVE DFHBMPRO TO PHONEA
    MOVE DFHBMPRO TO DATEA
    MOVE DFHBMPRO TO AMOUNTA
    MOVE DFHBMPRO TO COMMENTA
20    PERFORM MAP-BUILD THRU MAP-SEND
    EXEC CICS RETURN TRANSID('MENU') END-EXEC.

21  IF EIBTRNID = 'UPDT' THEN
22    MOVE 'FILE UPDATE' TO TITLE0
    MOVE 'CHANGE FIELDS AND PRESS ENTER' TO MSG30
23    MOVE FILEREc IN FILEA TO FILEREc IN COMMAREA
    MOVE 80 TO COMLEN.

MAP-BUILD.
24    MOVE NUMB    IN FILEA TO NUMBO.
25    MOVE NAME    IN FILEA TO NAMEO.
26    MOVE ADDRX   IN FILEA TO ADDRO.
27    MOVE PHONE   IN FILEA TO PHONEO.
28    MOVE DATEX   IN FILEA TO DATEO.
29    MOVE AMOUNT  IN FILEA TO AMOUNTO.
30    MOVE COMMENT IN FILEA TO COMMENTO.

MAP-SEND.
31    EXEC CICS SEND MAP('DETAIL') MAPSET('DFH$CGB')
   ERASE END-EXEC.

FIN.
32    GO TO CICS-CONTROL.

READ-INPUT.
33    EXEC CICS HANDLE CONDITION MAPFAIL(NOTMODF) NOTFND(NOTFOUND)

```

Source Listing for the Inquiry/Update sample Program (Continued)

```
          ERROR(ERRORS) DUPREC(DUPREC) END-EXEC.  
22      EXEC CICS RECEIVE MAP('DETAIL') MAPSET('DFH$CGB') END-EXEC.  
23      IF EIBTRNID = 'UPDT' THEN  
24          EXEC CICS READ UPDATE DATASET('FILEA') INTO(FILEA)  
                  RIDFLD(NUMB IN COMMAREA) END-EXEC  
24      IF FILEREC IN FILEA NOT = FILEREC IN COMMAREA THEN  
                  MOVE 'RECORD UPDATED BY OTHER USER, TRY AGAIN' TO MSG10  
                  MOVE DFHBMASB TO MSG1A  
                  MOVE DFHPROTN TO MSG3A  
                  PERFORM MAP-BUILD  
                  EXEC CICS SEND MAP('DETAIL') MAPSET('DFH$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.  
27      IF NAMEI      = LOW-VALUES AND  
          ADDRI       = LOW-VALUES AND  
          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.  
          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.
```

Source Listing for the Inquiry/Update Sample Program (Continued)

```
28    IF NAMEI NOT = LOW-VALUE MOVE NAMEI TO NAME IN FILEA.
      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(FILEA) LENGTH(92)
          END-EXEC.
      IF EIBTRNID = 'UPDT' THEN
        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 MSG30
      *          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.
      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.
```

Source Listing for the Inquiry/Update Sample Program (Continued)

```
MOVE LOW-VALUE TO MENU0.  
MOVE DFHBMASB TO MSGA.  
MOVE MESSAGES TO MSG0.  
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 KEY1 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 above.
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 the Browse Sample Program

```
*****  
* DFH$CBRW - CICS/VIS SAMPLE FILEA BROWSE - COBOL *  
*****  
IDENTIFICATION DIVISION.  
PROGRAM-ID. FILECBRW.  
ENVIRONMENT DIVISION.  
DATA DIVISION.  
WORKING-STORAGE SECTION.  
77 I      PIC 999  USAGE IS COMP.  
77 MESSAGES PIC X(39) VALUE ' '.  
* GOING FWD OR BACK... F/B, AT LO/HI END OF FILE  
77 CURROP  PIC X(1)  VALUE ' '.  
77 LASTOP   PIC X(1)  VALUE ' '.  
77 STATS    PIC X(1)  VALUE ' '.  
*  
77 RID      PIC 9(6)  VALUE IS ZERO.          DATA-AREAS FOR RIDFLD  
*  
77 RIDB     PIC 9(6)  VALUE IS ZERO.          BUILDS PREV BACK PAGE  
*  
77 RIDF     PIC 9(6)  VALUE IS ZERO.          BUILDS NEXT FWD PAGE  
*  
*  
*          COPY DFHBMSCA.  
*  
01 FILEA.  COPY DFH$CFIL.  
*  
*          COPY DFH$CGA.  
*  
*          COPY DFH$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.
```

Source Listing for the Browse Sample Program (Continued)

```
*****  
* SIMPLE CHECKS OF INPUT DATA *  
*****  
4 IF KEYL NOT = ZERO THEN  
  IF KEYI IS NUMERIC THEN  
    *  
    MOVE KEYI TO RID  
    MOVE KEYI TO RIDF  
    MOVE KEYI TO RIDB  
    ELSE  
    *  
    MOVE  
    'ACCOUNT NUMBER MUST BE NUMERIC'  
    TO MESSAGES  
    GO TO MENU  
    ELSE  
    *  
    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.  
*  
7 EXEC CICS HANDLE CONDITION  
  ENDFILE(TOOHIGH) END-EXEC.  
*  
  MOVE LOW-VALUES TO BROWSEO.  
*  
  MOVE RIDF TO RID.  
  MOVE I TO I.  
  NEXT-LINE.  
*  
8 EXEC CICS READNEXT INTO(FILEA) DATASET('FILEA')  
  RIDFLD(RID) END-EXEC.  
*  
9 IF I = 1 THEN  
  MOVE NUMB TO NUMBER10  
  MOVE NAME TO NAME10  
  MOVE AMOUNT TO AMOUNT10  
*  
  MOVE RID TO RIDB  
  ELSE IF I = 2 THEN  
    MOVE NUMB TO NUMBER20  
    MOVE NAME TO NAME20  
    MOVE AMOUNT TO AMOUNT20  
    ELSE IF I = 3 THEN  
      MOVE NUMB TO NUMBER30  
      MOVE NAME TO NAME30  
*  
  RIDB NEEDS EXISTING A/C
```

Source Listing for the Browse Sample Program (Continued)

```

MOVE AMOUNT TO AMOUNT30
ELSE IF I = 4 THEN
  MOVE NUMB TO NUMBER40
  MOVE NAME TO NAME40
  MOVE AMOUNT TO AMOUNT40.
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.
***** 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 NUMBER10
  MOVE NAME TO NAME10
  MOVE AMOUNT TO AMOUNT10
ELSE IF I = 3 THEN
  MOVE NUMB TO NUMBER20
  MOVE NAME TO NAME20
  MOVE AMOUNT TO AMOUNT20
ELSE IF I = 2 THEN
  MOVE NUMB TO NUMBER30
  MOVE NAME TO NAME30
  MOVE AMOUNT TO AMOUNT30
ELSE IF I = 1 THEN
  MOVE NUMB TO NUMBER40
  MOVE NAME TO NAME40

```

Source Listing for the Browse Sample Program (Continued)

```
MOVE AMOUNT TO AMOUNT40.  
*  
* ADD 1 TO I.  
* IF I NOT EQUAL 5 THEN GO TO PREV-LINE.  
* MOVE RID TO RIDB.  
* EXEC CICS SEND MAP('BROWSE') MAPSET('DFH$CGC')  
* ERASE END-EXEC.  
* GET NEXT RQUEST  
* GO TO PROMPT.  
*****  
* PROMPT FOR NEXT PAGING REQUEST*  
*****  
* PROMPT.  
MOVE CURROP TO LASTOP.  
12 EXEC CICS RECEIVE MAP('BROWSE') MAPSET('DFH$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$CGC') END-EXEC.  
* GO TO PROMPT.  
*****  
* HANDLE END OF FILE CONDITIONS *  
*****  
13 TOOHIGH.  
MOVE 'H' TO STATS.  
MOVE RID TO RIDF.  
MOVE RID TO RIDB.  
MOVE ' ' TO DIRO.  
MOVE 'HI-END OF FILE' TO MSG10.  
* BRT+PROT ATTR  
MOVE DFHBMASB TO MSG1A.  
EXEC CICS SEND MAP('BROWSE') MAPSET('DFH$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 MSG20.  
* 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.  
*****
```

Source Listing for the Browse Sample Program (Continued)

```
*****  
* DISPLAY GENERAL MENU THEN EXIT *  
*****  
MENU.  
*  
MOVE LOW-VALUE TO MENU0.  
MOVE DFHBMASB TO MSGA.  
MOVE MESSAGES TO MSG0.  
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/DOS/VS Installation and Operations Guide*. Note that 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)

where n is the destination trigger level (any integer from 0 through 32767).

Source Listing for the Order Entry Sample Program

```
*****  
* DFH$CREN - CICS/VС SAMPLE FILEA ORDER ENTRY - COBOL *  
*****  
IDENTIFICATION DIVISION.  
PROGRAM-ID. FILECREN.  
ENVIRONMENT DIVISION.  
DATA DIVISION.  
WORKING-STORAGE SECTION.  
77 ERROR-FLAG PIC 9.  
77 PREMSG PICTURE X(20) VALUE 'PROCESSING COMPLETED'.  
      COPY DFH$CGK.  
01 FILEA.    COPY DFH$CFIL.  
01 L86O.    COPY DFH$CL86.  
            COPY DFHBMSCA.  
*  
PROCEDURE DIVISION.  
*  
*  
1   EXEC CICS HANDLE AID CLEAR(ENDA) END-EXEC.  
2   EXEC CICS HANDLE CONDITION MAPFAIL(MAPFAIL)  
      NOTFND(NOTFOUND)  
      ERROR(ERRORS) END-EXEC.  
*  
*  
MOVE LOW-VALUES TO ORDER0.  
*  
3   EXEC CICS SEND MAP('ORDER') MAPSET('DFH$CGK')  
      ERASE END-EXEC.  
*  
HANDLE CONDITIONS  
CLEAR MAP
```

Source Listing for the Order Entry Sample Program (Continued)

```
*****  
*  
*  
*  
* RECEIVM.  
4  EXEC CICS RECEIVE MAP('ORDER') MAPSET('DFH$CGK') END-EXEC.  
*  
*  
* MOVE 0 TO ERROR-FLAG.  
* MOVE DFHBMFSE TO CUSTNOA, PARTNOA, QUANTA.  
*****  
*  
*  
*  
*  
* 5  IF CUSTNOI NOT NUMERIC THEN  
*     MOVE DFHUNINT TO CUSTNOA MOVE 1 TO ERROR-FLAG.  
*  
*  IF PARTNOI NOT NUMERIC THEN  
*     MOVE DFHUNINT TO PARTNOA MOVE 1 TO ERROR-FLAG.  
*  
*  IF QUANTI NOT NUMERIC THEN  
*     MOVE DFHUNINT TO QUANTA MOVE 1 TO ERROR-FLAG.  
*  
*  
*  
* 6  IF ERROR-FLAG = 1 THEN  
*     MOVE DFHBMASB TO MSG2A  
*     EXEC CICS SEND MAP('ORDER') MAPSET('DFH$CGK') END-EXEC  
*     GO TO RECEIVM.  
*****  
*  
*  
*  
* 7  EXEC CICS READ DATASET('FILEA') INTO(FILEA) RIDFLD(CUSTNOI)  
*     END-EXEC.  
*  
* 8  MOVE CUSTNOI TO CUSTNO.  
*  MOVE PARTNOI TO PARTNO.  
*  MOVE QUANTI TO QUANTITY.  
*  MOVE EIBTRMID TO TERMID.  
*****  
*  
*  
* 9  EXEC CICS WRITEQ TD QUEUE('L860') FROM(L860) LENGTH(22)  
*     END-EXEC.  
10  EXEC CICS SEND MAP('ORDER') MAPSET('DFH$CGK')  
*     MAPONLY ERASEAUP END-EXEC.  
*     GO TO RECEIVM.  
*  
*  
*  
* 11 NOTFOUND.  
*     MOVE DFHBMASB TO MSG1A.  
*     EXEC CICS SEND MAP('ORDER') MAPSET('DFH$CGK') END-EXEC.  
*     GO TO RECEIVM.  
*  
* 12 MAPFAIL.  
*     MOVE LOW-VALUES TO ORDER0.  
*     MOVE DFHBMASB TO MSG2A.  
*     EXEC CICS SEND MAP('ORDER') MAPSET('DFH$CGK') END-EXEC.  
*     GO TO RECEIVM.
```

Source Listing for the Order Entry Sample Program (Continued)

```
*****  
*  
***** EXIT FROM PROGRAM  
*  
*  
13  ERRORS.  
    MOVE 'TRANSACTION TERMINATED' TO MSG20.  
    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). 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 (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. Note that 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 the Order Entry Queue Print Sample Program

```
*****  
* DFH$CCOM - CICS/VС SAMPLE FILEA ORDER ENTRY QUEUE PRINT *  
*****  
IDENTIFICATION DIVISION.  
PROGRAM-ID. FILECOM.  
ENVIRONMENT DIVISION.  
DATA DIVISION.  
WORKING-STORAGE SECTION.  
77 Q-LENGTH      PIC 9(4)  COMP VALUE 22.  
77 PRESMSG      PIC X(20) VALUE 'PROCESSING COMPLETED'.  
01 LOGORD.  
 02 LOGTIME.  
    03 LDAY  PIC S9(7) COMP-3.  
    03 LTIME PIC S9(7) COMP-3.  
 02 LITEM      PIC X(22).  
 02 COMMENT    PIC X(11) VALUE 'ORDER ENTRY'.  
 02 FILLER     PIC X(51) VALUE SPACES.  
      COPY DFH$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
```

Source Listing for the Order Entry Queue Print Sample (Continued)

```
        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.
11      MOVE LOW-VALUES TO PRINTO
        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
*                                         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 L860. (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 the Low Balance Report Sample Program

```
*****  
* DFH$CREP - CICS/VIS 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
```

Source Listing for the Low Balance Report Sample Program (Continued)

```
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.
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 ABCODEC('ERRS') END-EXEC.
OFLW.
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 catalogued 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.

DFHSCGA Map Definition

```
DFH$CGA  TITLE 'FILEA - MAP FOR OPERATOR INSTRUCTIONS - COBOL'  
          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', *  
                 HIGHLIGHT=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'  
          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,  
                 HIGHLIGHT=REVERSE  
          DFHMDF POS=(12,25),LENGTH=6,INITIAL='NUMBER'  
          DFHMDF POS=(12,32),LENGTH=6,ATTRB=NUM,COLOR=GREEN,  
                 HIGHLIGHT=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 DFHSCGA

```
01  MENU1.  
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  MENU0 REDEFINES MENU1.  
02  FILLER PIC X(12).  
02  FILLER PICTURE X(3).  
02  MSG0  PIC X(39).  
02  FILLER PICTURE X(3).  
02  KEY0  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

```
DFH$CGB  TITLE 'FILEA - MAP FOR FILE INQUIRY/UPDATE - COBOL'  
          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 DFHSCGB

```
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  ADDR1  COMP PIC S9(4).
02  ADDR2  PICTURE X.
02  FILLER REDEFINES ADDR2.
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).
02  NUMBO  PIC X(6).
02  FILLER PICTURE X(3).
```

DSECT generated by DFHSCGB (Continued)

```
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 MSG10  PIC X(39).
02 FILLER PICTURE X(3).
02 MSG30  PIC X(39).
```

DFH\$CGB Screen Layout

```
+XXXXXXXXXXXX
+NUMBER: +XXXXXX+
+NAME:  +XXXXXXXXXXXXXXXXXXXXXX+
+ADDRESS: +XXXXXXXXXXXXXXXXXXXXXX+
+PHONE:  +XXXXXXX+
+DATE:  +XXXXXXX+
+AMOUNT: +XXXXXXX+
+COMMENT: +XXXXXXX+
+XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
+XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

DFH\$CGC Map Definition

```
DFH$CGC  TITLE 'FILEA - MAP FOR FILE BROWSE - COBOL'  
          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 DFHSCGC

```
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).  
02 NAME4L   COMP PIC S9(4).  
02 NAME4F   PICTURE X.
```

DSECT generated by DFHSCGC (Continued)

```
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 NUMBER10 PIC X(6).  
02 FILLER PICTURE X(3).  
02 NAME10 PIC X(20).  
02 FILLER PICTURE X(3).  
02 AMOUNT10 PIC X(8).  
02 FILLER PICTURE X(3).  
02 NUMBER20 PIC X(6).  
02 FILLER PICTURE X(3).  
02 NAME20 PIC X(20).  
02 FILLER PICTURE X(3).  
02 AMOUNT20 PIC X(8).  
02 FILLER PICTURE X(3).  
02 NUMBER30 PIC X(6).  
02 FILLER PICTURE X(3).  
02 NAME30 PIC X(20).  
02 FILLER PICTURE X(3).  
02 AMOUNT30 PIC X(8).  
02 FILLER PICTURE X(3).  
02 NUMBER40 PIC X(6).  
02 FILLER PICTURE X(3).  
02 NAME40 PIC X(20).  
02 FILLER PICTURE X(3).  
02 AMOUNT40 PIC X(8).  
02 FILLER PICTURE X(3).  
02 MSG00 PIC X(39).  
02 FILLER PICTURE X(3).  
02 MSG10 PIC X(39).  
02 FILLER PICTURE X(3).  
02 MSG20 PIC X(39).
```

DFH\$CGC Screen Layout

+FILE BROWSE		
+NUMBER	+NAME	+AMOUNT
+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		

DFHSCGD Map Definition

```
DFH$CGD  TITLE 'FILEA - MAPSET FOR LOW BALANCE REPORT - COBOL'  
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', *  
         HIGHLIGHT=UNDERLINE  
PAGEN   DFHMDF POS=(1,30),LENGTH=4,INITIAL='PAGE'  
         DFHMDF POS=(1,35),LENGTH=3  
         DFHMDF POS=(3,1),LENGTH=6,INITIAL='NUMBER'  
         DFHMDF POS=(3,17),LENGTH=4,INITIAL='NAME'  
FOOTING DFHMDF POS=(3,32),LENGTH=6,INITIAL='AMOUNT'  
         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 DFHSCGD

```
01 LINE0.  
  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 HEADING0.  
  02 FILLER PIC X(12).  
  02 FILLER PICTURE X(2).  
  02 PAGENA  PICTURE X.  
  02 PAGENO  PIC X(3).  
01 FOOTING0.  
  02 FILLER PIC X(12).  
01 FINAL0.  
  02 FILLER PIC X(12).
```

DFH\$CGD Screen Layout

DFHSCGK Map Definition

```
DFH$CGK  TITLE 'FILEA - MAP FOR ORDER ENTRY - COBOL'  
DFHMSD TYPE=&SYSPARM,MODE=INOUT,CTRL=(FREEKB,FRSET),  
        TIOAPFX=YES,LANG=COBOL,EXTATT=MAPONLY      x  
ORDER    DFHMDI SIZE=(12,40)  
        DFHMDF POS=(01,10),LENGTH=11,ATTRB=(BRT,ASKIP),  
                INITIAL='ORDER ENTRY',COLOR=BLUE,HIGHLIGHT=UNDERLINE x  
MSG1     DFHMDF POS=(03,04),LENGTH=26,ATTRB=(DRK,ASKIP),  
                INITIAL='NUMBER NOT FOUND - REENTER',  
                COLOR=RED,HIGHLIGHT=BLINK      x  
MSG2     DFHMDF POS=(04,04),LENGTH=22,ATTRB=(DRK,ASKIP),  
                INITIAL='DATA ERROR - REENTER',  
                COLOR=RED,HIGHLIGHT=BLINK      x  
        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 DFHSCGK

```
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 MSG10  PIC X(26).  
02 FILLER PICTURE X(3).  
02 MSG20  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).
```

DFHSCGK 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

```
DFH$CGL  TITLE 'FILEA - MAP FOR ORDER ENTRY QUEUE PRINT - COBOL'  
DFHMSD  TYPE=&SYSPARM, MODE=OUT, *  
        TIOAPIX=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/DOS/VS Installation and Operations 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 HIGHLIGHT 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.

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 below. The plus (+) sign in this and subsequent displays shows the position of the attribute byte. In an actual display, this position contains a blank.

+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+

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 (1) erases the screen and displays map DFH\$PGA in response to the EXEC CICS SEND MAP command, then (2) ends the program using the RETURN 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;
```

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. You should insert editing steps needed to ensure valid changes to the file. Those fields which 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 the Inquiry/Update Sample Program

```
/*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;
```

Source Listing for the Inquiry/Update Sample Program (Continued)

```
SUBSTR(ADDR(DFH$PGB0)->CHSTR,1,STG(DFH$PGB0))
      =LOW(STG(DFH$PGB0));
SELECT(EIBTRNID);
WHEN('PADD') DO;
  5      TITLE0='FILE ADD';
         MSG30='ENTER DATA AND PRESS ENTER KEY';
         NUMBO,COMMAREA.NUMB=KEYI;
         AMOUNTA=DFHBMUNN;
         AMOUNTO='$0000.00';
         COMLEN=7;
  6      CALL MAP_SEND;
         GO TO CICS_CONTROL;
         END;
WHEN('PINQ',
      'PUPD') DO;
  7      EXEC CICS HANDLE CONDITION NOTFND(NOTFOUND);
  8      EXEC CICS READ DATASET('FILEA') INTO(FILEA)
                  RIDFLD(KEYNUM);
         IF EIBTRNID='PINQ' THEN
           DO;
  9          TITLE0='FILE INQUIRY';
          MSG30='PRESS ENTER TO CONTINUE';
          CALL MAP_BUILD;
          /* PROTECT ALL FIELDS ON MAP */
          NAMEA,ADDRA,PHONEA,DATEA,AMOUNTA,
          COMMENTA = DFHBMPRO;
 10         CALL MAP_SEND;
          EXEC CICS RETURN TRANSID('PMNU');
          END;
         ELSE DO;
 12          TITLE0='FILE UPDATE';
          MSG30='CHANGE FIELDS AND PRESS ENTER';
          COMMAREA.FILEREC=FILEA.FILEREC;
 13          CALL MAP_BUILD;
          CALL MAP_SEND;
 14          COMLEN=80;
          GO TO CICS_CONTROL;
          END;
 16          OTHERWISE GO TO ERRORS;
         END;
MAP_BUILD:  PROC;
 17          NUMBO    = FILEA.NUMB;
          NAMEO    = FILEA.NAME;
          ADDRO    = FILEA.ADDRX;
          PHONEO   = FILEA.PHONE;
          DATEO    = FILEA.DATEX;
          AMOUNTO  = FILEA.AMOUNT;
          COMMENTO = FILEA.COMMENT;
          RETURN;
 18          END;
MAP_SEND:  PROC;
 18          EXEC CICS SEND MAP('DFH$PGB') ERASE;
          RETURN;
 19          END;
READ_INPUT:
 20          EXEC CICS HANDLE CONDITION MAPFAIL(NOTMODF) DUPREC(DUPREC)
                  ERROR(ERRORS) NOTFND(NOTFOUND);
 21          EXEC CICS RECEIVE MAP('DFH$PGB');
```

Source Listing for the Inquiry/Update Sample Program (Continued)

```
21      SELECT(EIBTRNID);
22      WHEN('PUPD')
23          DO;
24              EXEC CICS READ UPDATE DATASET('FILEA') INTO(FILEA)
25                  RIDFLD(COMMAREA.NUMB);
26              IF STRING(FILEA.FILEREC)~=STRING(COMMAREA.FILEREC) THEN
27                  DO;
28                      MSG10='RECORD UPDATED BY OTHER USER, TRY AGAIN';
29                      MSG1A=DFHBMASB;
30                      MSG3A=DFHPROTN;
31                      CALL MAP_BUILD;
32                      EXEC CICS SEND MAP('DFH$PGB') DATAONLY;
33                      COMMAREA.FILEREC=FILEA.FILEREC;
34                      COMLEN=80;
35                      GO TO CICS_CONTROL;
36                  END;
37              ELSE
38                  DO;
39                      FILEA.STAT='U';
40                      MESSAGES='RECORD UPDATED';
41                  END;
42              END;
43          WHEN('PADD') DO;
44              FILEA.STAT='A';
45              MESSAGES='RECORD ADDED';
46          END;
47      OTHERWISE GO TO ERRORS;
48  END;
49  IF NAMEL    = 0 &
50      ADDR    = 0 &
51      PHONEL  = 0 &
52      DATEL   = 0 &
53      AMOUNTL = 0 &
54      COMMENTL = 0 THEN
55      GO TO NOTMODF;
56
57  SELECT(EIBTRNID);
58  WHEN('PADD') IF
59      VERIFY(NAMEI,'ABCDEFGHIJKLMNOPQRSTUVWXYZ .-''')~=0 THEN
60          GO TO DATA_ERROR;
61  WHEN('PUPD') DO;
62      IF NAMEL~=0 THEN
63          IF VERIFY(NAMEI,'ABCDEFGHIJKLMNOPQRSTUVWXYZ .-''')~=0
64              THEN
65                  GO TO DATA_ERROR;
66          IF AMOUNTL~=0 THEN
67              IF VERIFY(AMOUNTI,'0123456789.$¢')~=0 THEN
68                  GO TO DATA_ERROR;
69              END;
70      OTHERWISE;
71  END;
72  IF EIBTRNID='PADD' THEN
73      FILEA.NUMB=COMMAREA.NUMB;
74      IF NAMEL  ~= 0 THEN FILEA.NAME=NAMEI;
75      IF ADDR   ~= 0 THEN FILEA.ADDRX=ADDRI;
76      IF PHONEL ~= 0 THEN FILEA.PHONE=PHONEI;
77      IF DATEL  ~= 0 THEN FILEA.DATEX=DATEI;
78      IF AMOUNTL ~= 0 THEN FILEA.AMOUNT=AMOUNTI;
79      ELSE IF EIBTRNID = 'PADD' THEN FILEA.AMOUNT='$0000.00';
80      IF COMMENTL~=0 THEN FILEA.COMMENT=COMMENTI;
```

Source Listing for the Inquiry/Update Sample Program (Continued)

```
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;
        MSG30='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$PGB') DATAONLY;
37      IF EIBTRNID='PADD' THEN COMLEN=7;
        ELSE COMLEN=80;
CICS_CONTROL:
38      EXEC CICS RETURN TRANSID(EIBTRNID) COMMAREA(COMMAREA)
                           LENGTH(COMLEN);
NOTMODF:
39      MESSAGES='RECORD NOT MODIFIED';
        GO TO PMNU;
DUPREC:
        MESSAGES='DUPLICATE RECORD';
        GO TO PMNU;
BADLENG:
        MESSAGES='PLEASE ENTER AN ACCOUNT NUMBER';
        GOTO PMNU;
BADCHARS:
        MESSAGES='ACCOUNT NUMBER MUST BE NUMERIC';
        GO TO PMNU;
NOTFOUND:
        MESSAGES='INVALID NUMBER - PLEASE REENTER';
        GO TO PMNU;
MFAIL:
        MESSAGES='PRESS CLEAR TO EXIT';
        GOTO PMNU;
ERRORS:
40      EXEC CICS DUMP DUMPCODE('ERRS');
        MESSAGES='TRANSACTION TERMINATED';
PMNU:
41      SUBSTR(ADDR(DFH$PGAO)->CHSTR,1,STG(DFH$PGAO))
           =LOW(STG(DFH$PGAO));
        MSGA=DFHBMASB;
        MSG0=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 above.
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 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 the Browse Sample Program

```
/*DFH$PBRW - CICS/VIS SAMPLE FILEA BROWSE - PL/I*/
/*BROWSE: PROC OPTIONS(MAIN);
  %INCLUDE DFHBMSCA;
  %INCLUDE DFH$PFIL;
  %INCLUDE DFH$PGA;
  %INCLUDE DFH$PGC;

  DCL
    (ADDR,
     HIGH,
     LOW,
     STG,
     SUBSTR,
     VERIFY) BUILTIN;

  DCL I      FIXED BIN(15);

  DCL (RID,
        RIDB,
        RIDF)
      PIC'999999' INIT(0);
      /*USED AS RIDFLD PARAM */
      /*FOR BUILDING PREV PAGE*/
      /*FOR BUILDING NEXT PAGE*/

  DCL (CURROP,
        LASTOP,
        STATUS) CHAR(1) INIT('');
      /* NOTE CURRENT OPERATION */
      /* LAST OPERATION */
      /* F = GOING FORWARDS, */
      /* B = GOING BACKWARDS. */
      /*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. */
/*
```

Source Listing for the Browse Sample Program (Continued)

```
/******  
/* 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;  
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$PGC0)->STRING,1,STG(DFH$PGC0)) = LOW(STG(DFH$PGC0));  
/*RESET FIELDS + ATTRB IN MAP C*/  
RID =RIDF; /*IF LAST REQUEST=BACKPAGE THEN*/  
/*NEED RIDF FOR FORWARD PAGING */  
9 CALL BUILDNEXT;  
RIDF=RID; /*USE RIDF FOR NEXT PAGE */  
10 EXEC CICS SEND MAP('DFH$PGC') ERASE;  
GOTO RECEIVE;  
PAGE_BACKWARD:  
11 CURROP='B';  
EXEC CICS HANDLE CONDITION ENDFILE(TOOLOW);  
SUBSTR(ADDR(DFH$PGC0)->STRING,1,STG(DFH$PGC0)) = LOW(STG(DFH$PGC0));  
/*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;
```

Source Listing for the Browse Sample Program (Continued)

```
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$PGC');  
  SELECT(DIRI);  
    WHEN('F') GOTO PAGE_FORWARD;  
    WHEN('B') GOTO PAGE_BACKWARD;  
    OTHERWISE  
      DO;  
        EXEC CICS SEND MAP('DFH$PGC');  
        GOTO RECEIVE;  
      END;  
    END;  
  ****  
  /*                                         HANDLE END OF FILE CONDITIONS */  
  ****  
TOOHIGH:  
13 STATUS='H';  
  RIDF = RID;  
  RIDB = RID;  
  DIRO = ' ';  
  MSG10='HI-END OF FILE';  
  MSG1A=DFHBMASB;  
  EXEC CICS SEND MAP('DFH$PGC') ERASE;  
  GOTO RECEIVE;  
TOOLOW:  
14 STATUS='L';  
  RIDF = 000000;  
  RIDB = 000000;  
  DIRO = ' ';  
  MSG20='LO-END OF FILE';  
  MSG2A=DFHBMASB;  
  EXEC CICS SEND MAP('DFH$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';
```

Source Listing for the Browse Sample Program (Continued)

```
/******  
/* DISPLAY GENERAL MENU THEN EXIT*/  
/******  
  
PMNU:  
18 SUBSTR(ADDR(DFH$PGA0)->STRING,1,STG(DFH$PGA0)) = LOW(STG(DFH$PGA0));  
      MSGA=DFHBMASB;  
      MSG0=MESSAGES;  
      EXEC CICS SEND MAP('DFH$PGA') ERASE;  
19 EXEC CICS RETURN;  
  
/******  
/* INTERNAL PROCEDURES           MOVE REQUIRED FIELDS TO MAP  */  
/******  
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;
```

Source Listing for the Browse Sample Program (Continued)

```
        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/DOS/VS Installation and Operations Guide*. Note that 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(L86O) TRIGGER(n)

where n is the destination trigger level (any integer from 0 through 32767).

Source Listing for the Order Entry Sample Program

```
/*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*/  
/* DFH$PREN - CICS/VS SAMPLE FILEA ORDER ENTRY - PL/I */  
/*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*/  
ORDER: PROC OPTIONS(MAIN);  
  
%INCLUDE DFHBMSCA;  
%INCLUDE DFH$PFIL;  
%INCLUDE DFH$PL86;  
%INCLUDE DFH$PGK;  
  
DCL  
  (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' */
```

Source Listing for the Order Entry Sample Program (Continued)

```
2 EXEC CICS HANDLE CONDITION MAPFAIL(MAPFAIL) /*EXITS FOR ERRORS */
      ERROR (ERRORS)
      NOTFND (NOTFOUND);

/*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*/
/*          ZEROIZE PLI STRUCTURE=DFH$PGK*/
/*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*/

      SUBSTR(ADDR(DFH$PGK0)->CHSTR,1,STG(DFH$PGK0))=LOW(STG(DFH$PGK0));

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*/
/*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*/
/*          SIMPLE VALIDATION OF DATA */
/*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*/

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;

  IF VERIFY(QUANTI,'1234567890')~=0 THEN
  DO;
    QUANTA = DFHUNINT;
    ERROR_FLAG = '1'B;
  END;

  IF ERROR_FLAG THEN
  DO;
    MSG2A=DFHBMASB;
    EXEC CICS SEND MAP('DFH$PGK') ERASE;
    GOTO RECEIVE;
  END;

/*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*/
/*          READ RECORD,CHECK CUSTNO EXISTS*/
/*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*/

7 EXEC CICS READ DATASET('FILEA') INTO(FILEA) RIDFLD(CUSTNOI);
8 CUSTNO = CUSTNOI;
PARTNO = PARTNOI;
QUANTITY = QUANTI;
TERMID = EIBTRMID;
```

Source Listing for the Order Entry Sample Program (Continued)

```
/******  
/* WRITE VALID ORDER TO TD QUEUE */  
/******  
9 EXEC CICS WRITEQ TD QUEUE ('L860') FROM (L860) LENGTH(22);  
10 EXEC CICS SEND MAP('DFH$PGK') MAPONLY ERASEUP;  
    GOTO RECEIVE;  
/******  
/* HANDLE ERRORS THEN RESTART */  
/******  
11 NOTFOUND:  
    CUSTNOA = DFHUNINT;                                /*INVALID ACCOUNT NO */  
    MSG1A   = DFHBMASB;                                /*NUMBER NOT FOUND */  
    EXEC CICS SEND MAP('DFH$PGK');  
    GOTO RECEIVE;  
12 MAPFAIL:                                         /*NO DATA IN FIELDS */  
    SUBSTR(ADDR(DFH$PGK0)->CHSTR,1,STG(DFH$PGK0))=LOW(STG(DFH$PGK0));  
    MSG2A=DFHBMASB;                                    /*DATA ERROR -REENTER*/  
    EXEC CICS SEND MAP('DFH$PGK');  
    GOTO RECEIVE;  
/******  
/* EXIT FROM PROGRAM */  
/******  
13 ERRORS:                                         /*GENERAL ERROR COND */  
13 MSG20='TRANSACTION TERMINATED';  
    MSG2A=DFHBMASB;                                    /*DATA ERROR -REENTER*/  
    EXEC CICS SEND MAP('DFH$PGK');  
    EXEC CICS DUMP DUMPCODE('ERRS');  
    GOTO EXIT;  
14 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
5. The order details are checked, invalid orders are redisplayed for correction. Error fields are highlighted and have MDIT set on. The user should add further editing steps necessary to ensure only valid orders are accepted.

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).

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. Note that 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 the Order Entry Queue Print Sample Program

```
*****  
/* DFH$PCOM - CICS/VS SAMPLE FILEA ORDER ENTRY QUEUE PRINT */  
*****  
QPRINT: PROC OPTIONS(MAIN);  
  %INCLUDE DFH$PFILE;  
  %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(' ');\br/>  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=='L86O' THEN  
    GO TO TIME;  
    SUBSTR(ADDR(DFH$PGL0)->CHSTR,1,STG(DFH$PGL0))  
    =LOW(STG(DFH$PGL0));  
  
Q_READ:
```

Source Listing for the Order Entry Queue Print Program (Cont)

```
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>'$0100.00' THEN DO;
6          ADDR0 = ADDRX;
          PART0 = PARTNO;
          NAM0 = NAME;
          NUM0 = CUSTNO;
          QUANT0 = QUANTITY;
7          EXEC CICS SEND MAP('DFH$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:
11     SUBSTR(ADDR(DFH$PGL0)->CHSTR,1,STG(DFH$PGL0))
          =LOW(STG(DFH$PGL0));
12     TITLE0='ORDER QUEUE IS EMPTY';
        EXEC CICS SEND MAP('DFH$PGL') DATAONLY ERASE PRINT L80;
TIME:
13     /* 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 */
     /* 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 the Low Balance Report Sample Program

```
/******  
* DFH$PREP - CICS/VIS 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\$PGD0)->STRING,1,STG(DFH\$PGD0))=
 LOW(STG(DFH\$PGD0));
7 AMOUNTO = AMOUNT;
 NUMERO = 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;

Source Listing for the Low Balance Report Sample Program (Continued)

```
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:
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 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', *
           HIGHLIGHT=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, *
           HIGHLIGHT=REVERSE
KEY DFHMDF POS=(12,25),LENGTH=6,INITIAL='NUMBER'
DFHMDF POS=(12,32),LENGTH=6,ATTRB=NUM,COLOR=GREEN, *
           HIGHLIGHT=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 DFHSPGA

```
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 DFHSPGB

```
DECLARE 1 DFH$PGBI AUTOMATIC UNALIGNED,
2 DFHMS1 CHARACTER (12),
2 TITLEL  FIXED BINARY (15,0),
2 TITLEF  CHARACTER (1),
2 TITLEI  CHARACTER (12),
2 NUMBL  FIXED BINARY (15,0),
2 NUMBF  CHARACTER (1),
2 NUMBI  CHARACTER (6),
2 NAMEL  FIXED BINARY (15,0),
2 NAMEF  CHARACTER (1),
2 NAMEI  CHARACTER (20),
2 ADDR1  FIXED BINARY (15,0),
2 ADDR2  CHARACTER (1),
2 ADDRI  CHARACTER (20),
2 PHONE1  FIXED BINARY (15,0),
2 PHONEF  CHARACTER (1),
2 PHONEI  CHARACTER (8),
2 DATEL  FIXED BINARY (15,0),
2 DATEF  CHARACTER (1),
2 DATEI  CHARACTER (8),
2 AMOUNTL  FIXED BINARY (15,0),
2 AMOUNTF  CHARACTER (1),
2 AMOUNTI  CHARACTER (8),
2 COMMENTL  FIXED BINARY (15,0),
2 COMMENTF  CHARACTER (1),
2 COMMENTI  CHARACTER (9),
2 MSG1L  FIXED BINARY (15,0),
2 MSG1F  CHARACTER (1),
2 MSG1I  CHARACTER (39),
2 MSG3L  FIXED BINARY (15,0),
2 MSG3F  CHARACTER (1),
2 MSG3I  CHARACTER (39),
2 FILL0092 CHARACTER (1);
DECLARE 1 DFH$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  MSG10    CHARACTER (39),
2  DFHMS12 FIXED BINARY (15,0),
2  MSG3A    CHARACTER (1),
2  MSG30    CHARACTER (39),
2  FILL0092 CHARACTER (1);
/* END OF MAP DEFINITION */
```

DFH\$PGB Screen Layout

```
+XXXXXXXXXXXX  
+NUMBER: +XXXXXXX+  
+NAME:  +XXXXXXXXXXXXXXXXXXXXXXXXX+  
+ADDRESS: +XXXXXXXXXXXXXXXXXXXX+  
+PHONE:  +XXXXXXX+  
+DATE:   +XXXXXXX+  
+AMOUNT: +XXXXXXX+  
+COMMENT: +XXXXXXX+  
+XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  
+XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

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,HIGHLIGHT=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$PGCI AUTOMATIC UNALIGNED,
2 DFHMS1 CHARACTER (12),
2 DIRL  FIXED BINARY (15,0),
2 DIRF  CHARACTER (1),
2 DIRI  CHARACTER (1),
2 NUMBER1L  FIXED BINARY (15,0),
2 NUMBER1F  CHARACTER (1),
2 NUMBERII  CHARACTER (6),
2 NAME1L  FIXED BINARY (15,0),
2 NAME1F  CHARACTER (1),
2 NAMEII  CHARACTER (20),
2 AMOUNTIL  FIXED BINARY (15,0),
2 AMOUNT1F  CHARACTER (1),
2 AMOUNTII  CHARACTER (8),
2 NUMBER2L  FIXED BINARY (15,0),
2 NUMBER2F  CHARACTER (1),
2 NUMBER2I  CHARACTER (6),
2 NAME2L  FIXED BINARY (15,0),
2 NAME2F  CHARACTER (1),
2 NAME2I  CHARACTER (20),
2 AMOUNT2L  FIXED BINARY (15,0),
2 AMOUNT2F  CHARACTER (1),
2 AMOUNT2I  CHARACTER (8),
2 NUMBER3L  FIXED BINARY (15,0),
2 NUMBER3F  CHARACTER (1),
2 NUMBER3I  CHARACTER (6),
2 NAME3L  FIXED BINARY (15,0),
2 NAME3F  CHARACTER (1),
2 NAME3I  CHARACTER (20),
2 AMOUNT3L  FIXED BINARY (15,0),
2 AMOUNT3F  CHARACTER (1),
2 AMOUNT3I  CHARACTER (8),
2 NUMBER4L  FIXED BINARY (15,0),
2 NUMBER4F  CHARACTER (1),
2 NUMBER4I  CHARACTER (6),
2 NAME4L  FIXED BINARY (15,0),
2 NAME4F  CHARACTER (1),
2 NAME4I  CHARACTER (20),
2 AMOUNT4L  FIXED BINARY (15,0),
2 AMOUNT4F  CHARACTER (1),
2 AMOUNT4I  CHARACTER (8),
2 MSG0L  FIXED BINARY (15,0),
2 MSG0F  CHARACTER (1),
2 MSG0I  CHARACTER (39),
2 MSG1L  FIXED BINARY (15,0),
2 MSG1F  CHARACTER (1),
2 MSG1I  CHARACTER (39),
2 MSG2L  FIXED BINARY (15,0),
2 MSG2F  CHARACTER (1),
2 MSG2I  CHARACTER (39),
2 FILL0084  CHARACTER (1);
DECLARE 1 DFH$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 NUMBER10  CHARACTER (6),
```

DSECT generated by DFHSPGC (continued)

```
2 DFHMS5 FIXED BINARY (15,0),
2 NAME1A CHARACTER (1),
2 NAME10 CHARACTER (20),
2 DFHMS6 FIXED BINARY (15,0),
2 AMOUNT1A CHARACTER (1),
2 AMOUNT10 CHARACTER (8),
2 DFHMS7 FIXED BINARY (15,0),
2 NUMBER2A CHARACTER (1),
2 NUMBER20 CHARACTER (6),
2 DFHMS8 FIXED BINARY (15,0),
2 NAME2A CHARACTER (1),
2 NAME20 CHARACTER (20),
2 DFHMS9 FIXED BINARY (15,0),
2 AMOUNT2A CHARACTER (1),
2 AMOUNT20 CHARACTER (8),
2 DFHMS10 FIXED BINARY (15,0),
2 NUMBER3A CHARACTER (1),
2 NUMBER30 CHARACTER (6),
2 DFHMS11 FIXED BINARY (15,0),
2 NAME3A CHARACTER (1),
2 NAME30 CHARACTER (20),
2 DFHMS12 FIXED BINARY (15,0),
2 AMOUNT3A CHARACTER (1),
2 AMOUNT30 CHARACTER (8),
2 DFHMS13 FIXED BINARY (15,0),
2 NUMBER4A CHARACTER (1),
2 NUMBER40 CHARACTER (6),
2 DFHMS14 FIXED BINARY (15,0),
2 NAME4A CHARACTER (1),
2 NAME40 CHARACTER (20),
2 DFHMS15 FIXED BINARY (15,0),
2 AMOUNT4A CHARACTER (1),
2 AMOUNT40 CHARACTER (8),
2 DFHMS16 FIXED BINARY (15,0),
2 MSG0A CHARACTER (1),
2 MSG00 CHARACTER (39),
2 DFHMS17 FIXED BINARY (15,0),
2 MSG1A CHARACTER (1),
2 MSG10 CHARACTER (39),
2 DFHMS18 FIXED BINARY (15,0),
2 MSG2A CHARACTER (1),
2 MSG20 CHARACTER (39),
2 FILL0084 CHARACTER (1);
/* END OF MAP DEFINITION */
```

DFH\$PGC Screen Layout

```
+FILE BROWSE
+NUMBER      +NAME      +AMOUNT
+XXXXXXX +XXXXXXXXXXXXXXXXXXXX+XXXXXX
+XXXXXXX +XXXXXXXXXXXXXXXXXXXX+XXXXXX
+XXXXXXX +XXXXXXXXXXXXXXXXXXXX+XXXXXX
+XXXXXXX +XXXXXXXXXXXXXXXXXXXX+XXXXXX

+PRESS CLEAR TO END BROWSE TRANSACTION
+PRESS PF1 OR TYPE F TO PAGE FORWARD
+PRESS PF2 OR TYPE B TO PAGE BACKWARD
```

DFHSPGD 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',      *
           HIGHLIGHT=UNDERLINE
PAGEN DFHMDF POS=(1,30),LENGTH=4,INITIAL='PAGE'
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$PCDO 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

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 DFHSPGK

```
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 MSG10   CHARACTER (26),
        2 DFHMS4 FIXED BINARY (15,0),
        2 MSG2A   CHARACTER (1),
        2 MSG20   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 TITLE0    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 ADDR0     CHARACTER (20),
        2 DFHMS6 FIXED BINARY (15,0),
        2 PARTA     CHARACTER (1),
        2 PART0     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	+xxxxxxxxxxxxxxxxxxxxxx	+xxxxxxxxxxxxxxxxxxxxxx
+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);
```

Appendix G. Report Controller Sample Programs

The report controller sample programs described in this appendix are included in source form on the report controller feature tape and are only executable when the report controller feature is installed.

The *CICS/DOS/VS Installation and Operations 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 a CICS sample application program based on the Low Balance Report sample program in appendixes D, E, and F. The sample program is presented in assembler language, COBOL, and PL/I.

You can invoke the three versions of the sample program by entering their transaction identifiers (ARPS, FRPS, and PRPS respectively) onto a clear screen.

Description

The low balance report sample program creates a report called DFH\$ARPS, DFH\$CRPS, or DFH\$PRPS respectively, which 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 ARPS, FRPS, or PRPS 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\$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 for the Low Balance Report Sample Program (ASM)

```

* $SEG(DFH$ARPS),COMP(SAMPLES),PROD(CICS/DOS):
TITLE 'DFH$ARPS - CICS/VS SAMPLE FILEA LOW BALANCE INQUIRY - -*
ASSEMBLER - WITH SPOOLER FEATURE'

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
*
1
TOK DS D TOKEN FIELD
PRTLINE DS 0CL50 PRINT LINE
SPACE1 DS CL5' '
PLNUMB DS CL6 NUMBER
SPACE2 DS CL5' '
PLNAME DS CL20 NAME
SPACE3 DS CL5' '
PLAMNT DS CL8 AMOUNT
*
COPY DFH$AGD OUTPUT MAP
COPY DFH$AFIL FILEA'S RECORD DESCRIPTION
DFH$ARPS CSECT
MVC KEYNUM(6),=C'000000' SET RECORD KEY TO ZERO
EXEC CICS HANDLE CONDITION ERROR(ERRORS) OVERFLOW(OFLOW) *
ENDFILE(ENDFILE) LENGERR(ENDTASK)
MVI PAGENA,X'00' MOVE X'00' TO ATTRIBUTE
MVC PAGEN,PAGE1 INITIALIZE PAGE NUMBER TO 1
BAL RETREG,MAPNUM MOVE PAGENUMBER TO MAP AREA
EXEC CICS SEND MAP('HEADING') MAPSET('DFH$AGD') ACCUM PAGING *
ERASE
*
2
EXEC CICS SPOOLOPEN REPORT('DFH$ARPS') TOKEN(TOK) *
TITLE('LOW BALANCE REPORT') HEAD(HEAD) DATETIME *
NOSEP NOCC HEADNUM
EXEC CICS STARTBR DATASET('FILEA') RIDFLD(KEYNUM)
REPEAT DS 0H
EXEC CICS READNEXT INTO(FILEA) DATASET('FILEA') *
RIDFLD(KEYNUM)
CLC AMOUNT,LOWLIM COMPARE AMOUNT ON RECORD WITH LIM
BH REPEAT ..OK, GREATER THAN $50, TRY NEXT
XC DFH$AGDO(DFH$AGDE-DFH$AGD0),DFH$AGDO CLEAR MAP
MVC AMOUNTO,AMOUNT MOVE AMOUNT ON FILE TO MAP
MVC NUMBERO,NUMB MOVE ACCOUNT NUMBER TO MAP
MVC NAMEO,NAME MOVE NAME TO MAP
*
3
MVC PLAMNT,AMOUNT MOVE AMOUNT ON FILE TO PRINT LINE
MVC PLNUMB,NUMB MOVE ACCOUNT NUMBER TO PRINT LINE
MVC PLNAME,NAME MOVE NAME TO PRINT LINE
4
EXEC CICS SPOOLWRITE REPORT('DFH$ARPS') TOKEN(TOK) *
FROM(PRTLINE) FLENGTH(50)
*
EXEC CICS SEND MAP('DFH$AGD') MAPSET('DFH$AGD') ACCUM PAGING

```

Source for the Low Balance Report Sample Program (ASM) (Cont'd)

```

B      REPEAT          GO BUILD NEXT MAP
***** END ROUTINE AND GENERAL ROUTINES *****
* ENDFILE DS  OH          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
* ENDFILE DS  OH          END OF FILE CONDITION RAISED
*          EXEC CICS SEND MAP ('FINAL') MAPSET ('DFH$AGD') MAPONLY      *
*          ACCUM PAGING
*          EXEC CICS SEND PAGE
*          EXEC CICS SEND TEXT FROM (OPINSTR) ERASE
*          EXEC CICS ENDBR DATASET('FILEA')
*
* 5      EXEC CICS SPOOLWRITE REPORT('DFH$ARPS') TOKEN(TOK)      *
*          FROM(LASTLINE) FLENGTH(26)
* 6      EXEC CICS SPOOLCLOSE REPORT('DFH$ARPS') TOKEN(TOK) HOLD
*
*          A RECEIVE COMMAND IS ISSUED TO GIVE
*          THE TERMINAL OPERATOR A CHANCE TO
*          READ THE PROMPTING MESSAGE.
*
*          THE TRANSACTION WILL TERMINATE WHEN
*          THE OPERATOR PRESSES THE ENTER KEY.
*
*          PAGING COMMANDS CAN THEN BE ISSUED.
*
*          NO HARM IS DONE IF THE OPERATOR
*          TYPES IN DATA BEFORE PRESSING THE
*          ENTER KEY.
*
* ENDTASK LA  WORKREG,1
*          STH WORKREG,TERMLENG
*          EXEC CICS RECEIVE INTO(TERMDATA) LENGTH(TERMLENG)
*          EQU *
*          EXEC CICS RETURN
* ERRORS  DS  OH
*          EXEC CICS HANDLE CONDITION ERROR
*          EXEC CICS PURGE MESSAGE
*          EXEC CICS ABEND ABCODE('ERRS')
* OFLOW   DS  OH          PAGE BUILT HERE
*          EXEC CICS SEND MAP('FOOTING') MAPSET('DFH$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
*          EXEC CICS SEND MAP('HEADING') MAPSET('DFH$AGD') ACCUM PAGING *
*          ERASE
*          EXEC CICS SEND MAP('DFH$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
* HEAD    DC  CL132'          NUMBER          NAME          AMOUNT
* LASTLINE DC  CL26'END OF LOW BALANCE REPORT'
* END

```

Source for the Low Balance Report Sample Program (COBOL)

```
* $SEG(DFH$CRPS),COMP(SAMPLES),PROD(CICS/DOS): *  
*****  
* DFH$CRPS - CICS/VS SAMPLE FILEA LOW BALANCE INQUIRY *  
* - COBOL WITH SPOOLER FEATURE *  
*****  
IDENTIFICATION DIVISION.  
PROGRAM-ID. FILECRPS.  
ENVIRONMENT DIVISION.  
DATA DIVISION.  
WORKING-STORAGE SECTION.  
1 77 TOK          PIC X(8) VALUE LOW-VALUES.  
77 HEAD          PIC X(132) VALUE '      NUMBER      NAME  
                  '          AMOUNT'.  
77 LASTLINE      PIC X(26) VALUE 'END OF LOW BALANCE REPORT'.  
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$CFIL.  
*  
01 PRTLINE.  
02 FILLER        PIC X(5).  
02 PLNUMB        PIC X(6).  
02 FILLER        PIC X(5).  
02 PLNAME        PIC X(20).  
02 FILLER        PIC X(5).  
02 PLAMNT        PIC X(8).  
*  
PROCEDURE DIVISION.  
    EXECUTE CICS HANDLE CONDITION ERROR(ERRORS)  
          OVERFLOW(OFLOW) ENDFILE(ENDFILE)  
          LENGERR(ENDTASK) END-EXEC  
    MOVE LOW-VALUE TO PAGENA  
    MOVE PAGEN      TO PAGENO  
    EXEC CICS SEND MAP('HEADING') MAPSET('DFH$CGD') ACCUM  
          PAGING ERASE END-EXEC  
*  
2  EXEC CICS SPOOPEN REPORT('DFH$CRPS') TOKEN(TOK)  
          TITLE('LOW BALANCE REPORT') HEAD(HEAD) DATETIME  
          NOSEP NOCC HEADNUM END-EXEC  
*  
    EXEC CICS STARTBR DATASET('FILEA') RIDFLD(KEYNUM) END-EXEC.  
REPEAT.  
    EXEC CICS READNEXT INTO(FILEA) RIDFLD(KEYNUM)  
          DATASET('FILEA') END-EXEC  
    MOVE AMOUNT      TO AMOUNTO  
    IF AMOUNTO GREATER THAN LOWLIM GO TO REPEAT.  
    MOVE LOW-VALUE TO LINEO  
    MOVE AMOUNT      TO AMOUNTO  
    MOVE NUMB        TO NUMBERO  
    MOVE NAME        TO NAMEO
```

Source for the Low Balance Report Sample Program (COBOL) (Cont'd)

```
      MOVE NAME      TO NAMEO
  *
 3   MOVE AMOUNT    TO PLAMNT
     MOVE NUMB     TO PLNUMB
     MOVE NAME     TO PLNAME
 4   EXEC CICS SPOOLWRITE REPORT('DFH$CRPS') TOKEN(TOK)
          FROM(PRTLINE) FLENGTH(50) END-EXEC
  *
 5   EXEC CICS SEND MAP('LINE') MAPSET('DFH$CGD')
          ACCUM PAGING END-EXEC
     GO TO REPEAT.
  ENDFILE.
  EXEC CICS SEND MAP('FINAL') MAPSET('DFH$CGD')
          MAPONLY ACCUM PAGING END-EXEC
  EXEC CICS SEND PAGE END-EXEC
  EXEC CICS SEND TEXT FROM(OPINSTR) LENGTH(52) ERASE END-EXEC
  EXEC CICS ENDBR DATASET('FILEA') END-EXEC
  *
 5   EXEC CICS SPOOLWRITE REPORT('DFH$CRPS') TOKEN(TOK)
          FROM(LASTLINE) FLENGTH(26) END-EXEC
 6   EXEC CICS SPOOLCLOSE REPORT('DFH$CRPS') TOKEN(TOK)
          HOLD END-EXEC
  *
  *
  *          A RECEIVE COMMAND IS ISSUED TO GIVE THE
  *          TERMINAL OPERATOR A CHANCE TO READ THE
  *          PROMPTING MESSAGE.
  *
  *          THE TRANSACTION WILL TERMINATE WHEN THE
  *          OPERATOR PRESSES THE ENTER KEY.
  *
  *          PAGING COMMANDS CAN THEN BE ISSUED
  *
  *          NO HARM IS DONE IF THE OPERATOR TYPES IN
  *          DATA BEFORE PRESSING THE ENTER KEY.
  EXEC CICS RECEIVE INTO(TERMDATA) LENGTH(TERMLENG) END-EXEC.
  ENDTASK.
  EXEC CICS RETURN END-EXEC.
  GOBACK.
  ERRORS.
  EXEC CICS HANDLE CONDITION ERROR END-EXEC
  EXEC CICS PURGE MESSAGE END-EXEC
  EXEC CICS ABEND ABCODEC('ERRS') END-EXEC.
  OFLOW.
  EXEC CICS SEND MAP('FOOTING') MAPSET('DFH$CGD')
          MAPONLY ACCUM PAGING END-EXEC
  ADD 1      TO PAGEN
  MOVE PAGEN TO PAGENO
  EXEC CICS SEND MAP('HEADING') MAPSET('DFH$CGD')
          ACCUM PAGING ERASE END-EXEC.
  EXEC CICS SEND MAP('LINE') MAPSET('DFH$CGD')
          ACCUM PAGING END-EXEC
  GO TO REPEAT.
```

Source for the Low Balance Report Sample Program (PL/I)

```
/* $SEG(DFH$PRPS),COMP(SAMPLES),PROD(CICS/DOS): */  
/* DFH$PRPS - CICS/VS SAMPLE FILEA LOW BALANCE INQUIRY - PL/I */  
/* -- PLI WITH SPOOLER FEATURE */  
/* REPORT: PROC OPTIONS(MAIN); */  
1 DCL TOK CHAR(8) INIT(' ');  
DCL HEAD CHAR(132) STATIC  
    INIT('      NUMBER          NAME          AMOUNT');  
DCL LASTLINE CHAR(26) STATIC  
    INIT('END OF LOW BALANCE REPORT');  
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$PFIL;  
DCL 1 PRTLINE,  
    2 FILL01 CHAR(5) INIT(' '),  
    2 PLNUMB PIC'(6)9',  
    2 FILL02 CHAR(5) INIT(' '),  
    2 PLNAME CHAR(20),  
    2 FILL03 CHAR(5) INIT(' '),  
    2 PLAMNT CHAR(20);  
EXEC CICS HANDLE CONDITION ERROR(ERRORS) OVERFLOW(OFLOW)  
    ENDFILE(ENDFILE) LENGERR(END_TASK);  
PAGENA=LOW(1);  
PAGENO=PAGEN;  
EXEC CICS SEND MAP('HEADING') MAPSET('DFH$PGD') ACCUM PAGING ERASE;  
2 EXEC CICS SPOOLOPEN REPORT('DFH$PRPS') TOKEN(TOK)  
    TITLE('LOW BALANCE REPORT') HEAD(HEAD) DATETIME  
    NOSEP NOCC HEADNUM;  
EXEC CICS STARTBR DATASET('FILEA') RIDFLD(KEYNUM);  
REPEAT:  
EXEC CICS READNEXT INTO(FILEA) DATASET('FILEA') RIDFLD(KEYNUM);  
    IF AMOUNT<=LOWLIM THEN  
        DO;  
            SUBSTR(ADDR(DFH$PGD0)-->STRING,1,STG(DFH$PGD0))=  
                LOW(STG(DFH$PGD0));  
            AMOUNTO = AMOUNT;  
            NUMBERO = NUMB;  
            NAMEO = NAME;  
3        PLAMNT = AMOUNT;  
        PLNUMB = NUMB;  
        PLNAME = NAME;  
4        EXEC CICS SPOOLWRITE REPORT('DFH$PRPS') TOKEN(TOK)  
            FROM(PRTLINE) FLENGTH(50);  
        EXEC CICS SEND MAP('DFH$PGD') MAPSET('DFH$PGD') ACCUM PAGING;  
    END;  
    GOTO REPEAT;  
ENDFILE:  
EXEC CICS SEND MAP('FINAL') MAPSET('DFH$PGD') MAPONLY ACCUM PAGING;  
EXEC CICS SEND PAGE;  
EXEC CICS SEND TEXT FROM(OPINSTR) ERASE;  
EXEC CICS ENDBR DATASET('FILEA');
```

Source for the Low Balance Report Sample Program (PL/I) (Cont'd)

```
5 EXEC CICS SPOOLWRITE REPORT('DFH$PRPS') TOKEN(TOK)
  FROM(LASTLINE) FLENGTH(26);
6 EXEC CICS SPOOLCLOSE REPORT('DFH$PRPS') TOKEN(TOK) HOLD;
  /* 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;
EXEC CICS RECEIVE INTO(TERM DATA) LENGTH(TERM LENG);
END_TASK:
EXEC CICS RETURN;
ERRORS:
EXEC CICS HANDLE CONDITION ERROR;
EXEC CICS PURGE MESSAGE;
EXEC CICS ABEND ABCODE('ERRS');
OFLOW:
EXEC CICS SEND MAP('FOOTING') MAPSET('DFH$PGD')
  MAPONLY ACCUM PAGING;
PAGEN = PAGEN+1;
PAGENA = LOW(1);
PAGENO = PAGEN;
EXEC CICS SEND MAP('HEADING') MAPSET('DFH$PGD')
  ACCUM PAGING ERASE;
EXEC CICS SEND MAP('DFH$PGD') MAPSET('DFH$PGD') ACCUM PAGING;
GOTO REPEAT;
END;
```

Program Notes

1. The TOKEN field is defined on a word boundary.
2. The SPOOLOPEN command opens a report on the POWER spool file, a report token is allocated and returned in the TOK field. You decide at this stage which options the report will have.

In this case, the report will have a title of LOW BALANCE REPORT, will have a heading in each page consisting of NUMBER - NAME - AMOUNT, with the page number, the date, and the time inserted on the right.

3. Fields are moved from the selected customer record to the area that will be written to the report.
4. The customer details line is written to the report.
5. The end of report message is written to the report.
6. The report is closed. At this stage the status of the report may be decided. In this case, the report called DFII\$ARPS, DFII\$CRPS, or DFII\$PRPS will have been created and will be on the POWER spool file with a disposition of L.

Index

Special Characters

& (CL interpreter) 65
&DFHEIMX (macro global bit) 15
*ASM statement (assembler language) 16
*PROCESS statement (PL/I) 16
? (CL interpreter) 62

A

ABCODE option 43, 387
ABEND (abnormal termination) exit 385
ABEND command 387
abend user task (EDF) 54
abnormal termination
 exceptional condition 387
 options 387
 reactivate an exit 385
 recovery 385
absolute expression 6
ABSTIME option 346
access to DL/I data base
 DL/I CALL statement 115
 EXEC DL/I command 105
access to system information
 ADDRESS command 41
 ASSIGN command 42
 CICS storage areas 41
 EXEC interface block (EIB) 41
ACCOUNT option 393
ACCUM option 186, 229
activate an ABEND exit 385
active partition 165, 168
ACTPARTN option 173, 230
adding records
 to BDI data set 325
 to DAM data set 90
 to VSAM data set 82
address
 cursor 253
 PCB 116
ADDRESS command 41
AID (see attention identifier)
AIX (alternate index) 76
ALARM option 153, 230
ALIGNED attribute (PL/I) 9
ALL option 230
ALLOCATE command 258, 261
alternate facility 239
alternate index (AIX) 76
alternate index path 76

alternate key 76
alternate screen size 140
ALTSCRN operand 221
ampersand (CL interpreter) 65
ANYKEY option 156, 254
API (application programming interface) 41
APOST option 19
application data area of screen 139
application partition set 168, 172
application program logical levels 355
application program using commands and macros 15
application programming interface (API) 41
application-oriented information (LU6) 259
APPLID option 43
argument value
 assembler language 6
 COBOL 7
 PL/I 8
ASA option 313
ASIS option
 basic mapping support 155, 230
 terminal control 283
ASKTIME command 340
assembler language
 argument value 6
 coding conventions 5
 LENGTH option default 7
 program exit 12
 programming techniques 24
 register contents 12
 restrictions 24
 sample programs 427, 549
 translated code 12
assembling a map 145
assembling a partition set 171
ASSIGN command 42
asynchronous interrupt 242
asynchronous journal output 400
asynchronous page build 158
ATT (see automatic task initiation)
ATTACHID option 283
attention condition (SIGNAL) 248
attention identifier (AID)
 constants 224
 HANDLE AID command 156
 input without data 253
 list (DFHAIID) 254
 3270 input operation 136
ATTTRB operand 207, 221
attribute character 136, 150
attribute constants 150
attribute control character list (DFHBMSCA) 222, 255
attribute, extended 162
audible alarm (MSR) 176
Audio Response Unit (7770) 282

audio terminal (2721) 282
autoanswer transaction (3735) 276
autocall transaction (3735) 276
automated features of the spooler 301
automatic task initiation (ATI) 368
AUTOPAGE option 183, 230
autoskip field 137
AUXILIARY option 376
auxiliary storage temporary data 373
auxiliary trace facility 392

B

background transparency 139
backout of resources 405
base color 137
base locator for linkage (BLL)
 chained storage areas 26
 large storage areas 27
 OCCURS DEPENDING ON clauses 27
 optimization feature 27
 storage addressing 25
BASE operand 208
base state 169
basic mapping support (BMS)
 an introduction 131
 assembling maps 145
 block data format 179
 BMS and GDDM 143
 cataloging maps 145
 CMSG message switching transaction 201
 completing a logical message 183
 coordinating BMS and another screen
 manager 154
 cumulative mapping 186
 cumulative output processing 186
 cursor position 153
 deleting a logical message 183
 determining the actual input partition (RECEIVE PARTN) 225
 device control options 153
 exceptional conditions 157, 236
 field data format 148, 149
 field definition macro 144, 206
 field group 144
 floating maps 186
 full function BMS 181
 GDDM coordination 143, 154
 header and trailer maps 186
 input field suffix 148
 input partition (RECEIVE PARTN) 173
 invalid data 150
 loading a partition set 172
 logical device components 175
 logical message 181
 map definition macro 142, 206

map positioning 187
map set definition macro 141, 205
map set suffixing 145
map set termination 144
map sets 132
map size 24
mapping input data 154, 225
mapping output data 150
maps 147
minimum function BMS 135
null map 142
options 229
outboard formatting 179
output field suffixes 149
page overflow 186, 190
partition definition macro 170, 220
partition set definition macro 170, 220
physical map 145
pregenerated versions 134
printer support 163
returning mapped data to a program 201
routing a logical message (ROUTE) 195
SCS and non-3270 printers 164
sending a user defined data stream (SEND TEXT NOEDIT) 203
sending data previously mapped by BMS (SEND TEXT MAPPED) 203
sending data to a display 150
sending device controls without data (SEND CONTROL) 153
sending text data (SEND TEXT) 162
standard function BMS 161
symbolic map 145, 147
terminal code table 202
terminal operator paging 184
3270 printer using NLEOM 163
3270 printer without NLEOM 163
batch data interchange 323
 add record to data set 325
 delete a record from data set 326
 destination identification 323
 exceptional conditions 329
 interrogate a data set 324
 options 327
 read record from data set 324
 request next record number 326
 send data to output device 326
 terminate data set 326
 update a record in data set 325
 wait for function completion 327
batch logical unit (3770) 278
batch mode application (3740) 277
BIF DEEDIT (built-in function) 409
blank lines and 3270 printer 159
BLL (see base locator for linkage)
block data format (BMS) 179
block reference 89
blue parameter of COLOR operand 209

BMS (see basic mapping support)
 BMS logical message (see logical messages)
 BOTHNUM option 313
 BOTTOM command (CEBR) 70
 bracket protocol (LAST option) 248
 bright intensity field 137
 browse operation
 DAM 89
 ending 97
 read next record during 96
 read previous record 96
 reset starting point 97
 specify starting point 95
 VSAM 79
 browse temporary storage (EDF) 54
 browse transaction (CEBR) 69
 BTAM programmable device 250
 BTRANS option 43
 BUFFER option 283
 BUFSZE operand 221
 BUILD ATTACH command 259
 built-in function (BIF DEEDIT) 409

C

CALL statement 24
 CALLDLI macro 115
 CANCEL command 345
 CANCEL option 387
 CARD option 327
 cataloging a map 145
 cataloging a partition set 171
 CBIDERR condition 290
 CBL statement (COBOL) 16
 CBUFF option 283
 CEBR (browse transaction) 69
 CECI (see command level interpreter) 61
 CECS (see command level interpreter) 61
 CEDF transaction 51
 cell size for partitions 174
 chained storage area, COBOL 26
 chaining of data 244, 246
 character attribute 162
 character cell size 169, 174
 CHARSIZE operand 221
 checking a DL/I call 119
 checkout, program 49
 CICS option 17
 CLASS option 313
 CLEAR key 156, 165
 CLEAR option 156, 254
 CLEAR PARTITION key 165
 closing a report 299
 CLRPARTN option 254
 CMSG message switching transaction 201
 CNOTCOMPL option 283

COBOL
 argument value 7
 base locator for linkage (BLI) 25
 compilers supported 25
 program segments 28
 restrictions 24
 sample programs 471
 translated code 15
 CODEREG argument 14
 coding conventions 5
 COLOR operand 209
 COLOR option 43
 COLUMN command (CEBR) 70
 COLUMN operand 187, 210
 command
 argument values 6
 end-of-command delimiter 5
 execution (CL interpreter) 64
 format 5
 macro equivalent 425
 printer spooling 311
 report controller 311
 syntax check 63
 syntax notation 4
 command language translator
 data set 11
 optional facilities 16
 translated code 12
 command level interpreter (CECI/CECS)
 an introduction 61
 command input area 62
 information area 63
 installing 68
 invoking 61
 PF key values area 66
 program control 67
 screen layout 62
 security rules 68
 status area 62
 terminal sharing 67
 variables 65
 COMMAREA option 363
 common buffer, output to (2980) 267
 communication area (DFHCOMMAREA) 26
 communication line, relinquishing 242
 compiler options 17
 compilers supported
 COBOL 25
 PL/I 29
 COMPLETE option 396
 conditions (exceptional conditions) 31
 CONFIRM option 283
 CONNECT PROCESS command 261
 CONSOLE option 327
 constants
 AID values (DFHAID) 224
 attribute values (DFHBMSCA) 222
 for examining EIBAID field 157, 224

for MSR control values 224
 for printer format controls 222
 for 3270 attributes 222
 MSR control (DFHMSRCA) 224
 printer control values (DFHBMSCA) 222
 3270 attributes 150

control
 exclusive, DAM 91
 exclusive, VSAM 88
 pass with return 355
 pass without return 356
 return 356
 trace 393

control area, VSAM 76
 control interval, VSAM 76
 CONVERSE command 242
 CONVERSE option 283
 converse with terminal or LU 242
 CONVID option 284
 COPIES option 313

copy
 displayed information 253
 symbolic description map 147

copy book
 DFHAID 224
 DFHBMSCA 150, 222
 DFHEIBLK 13
 DFHMSRCA 224

copybook DFIIAID 157

create a journal record 401

CSA option 42

CTLCHAR option 284

CTRL operand 210

cumulative output processing 186

cumulative text processing 192

CURRENT option 230

CURSOR option 153, 230

cursor position
 basic mapping support 153
 terminal control 253

CURSOR SELECT key 157

CWA option 42

CWALENG option 44

D

DAM
 browsing operations 89
 data sets 89, 90
 exclusive control 91

data
 chaining 244
 temporary storage 373

data communication operations 129

data comparison 167

data definition 22

data fields on screen 139
 data initialization 22
 data integrity 77
 data interchange (see batch data interchange)
 DATA operand 210
 DATA option 366
 data set
 access from CICS application programs 78
 batch data interchange 323
 blocked 77
 DAM 89
 identification 78
 translator 11
 VSAM 87

data-area argument 6
 data-value argument 6
 data, passing to other program 358
 DATAONLY option 151, 230
 DATAREG argument 14
 DATASET option 98
 DATASTR option 284
 date feature of the spooler 303
 date field of EIB 41
 DATE option 346
 DATEFORM option 346
 DATESEP option 346
 DATETIME option 313
 DAYCOUNT option 346
 DAYOFMONTH option 346
 DAYOFWEEK option 346
 DCT option 396
 DDMYY option 346
 deadlock prevention 88
 DEBKEY option 98
 deblocking argument 89
 DEBREC option 98
 DEBUG option 17
 debugging 49, 379
 default action for conditions 31
 default screen size 140
 deferred journal output 400
 deferred light pen field 157
 defining a map set 141
 defining partition sets 170
 definite response protocol
 batch data interchange 324
 terminal control 247

DEFRESP option
 batch data interchange 327
 terminal control 247, 284

DELAY command 340

delay processing of task 340

DELETE command 95

DELETE option 313

DELETE option, report controller 299

DELETEQ TD command 370

DELETEQ TS command 375

deleting

batch data interchange record 326
 file control record 95
 loaded program 358
 temporary storage queue 375
 transient data queue 370
DELIMITER option 44
 delimiter, end-of-command 5
DEQ command 353
 dequeue from resource 353
DEST option 284
DESTCOUNT option 44, 201, 229
DESTID option
ASSIGN command 44
 batch data interchange 327
DESTIDLENG option
ASSIGN command 44
 batch data interchange 327
 destination
 extrapartition 368
 identification 323
 indirect 368
 intrapartition 367
 destination for a report 307
DESTINATION option 314
 detect an attention condition 248
 device control options (BMS) 153, 194
 device dependent data stream 131
DFHAID copy book 224, 254
DFHBMSCA copy book 150, 222, 255
DFHCOMMAREA (communication area) 26
DFHEAI interface processor 12
DFHECALL macro 12
DFHEIBLK copy book 13
DFHEICAL macro - use **DFHECALL** 12
DFHEIEND macro 12
DFHEIENT macro 12
 CODEREG 14
 DATAREG 14
 defaults 14
 EIBREG 14
DFHEIPLR symbolic register 14
DFHEIRET macro 12
DFHEISTG macro 12
DFHMDF field definition macro 143, 206
DFHMDI map definition macro 142, 206
DFHMSD map set definition macro 141, 205
DFHMSRCA copy book 176, 224
DFHPDI partition definition macro 170, 220
DFHPSD partition set definition macro 220
DFHRESP builtin function 32
DFH2980 structure 267
DIB (DL/I interface block) 108
 direct terminal 197
DISABLED condition 100
 disconnect a switched line 242
 display device operations 252
 attention identifier (AID) 253
 attention identifier list (DFHAID) 254
 copy displayed information 253
 cursor address 253
 erase all unprotected fields 253
 input operation without data 253
 pass control on receipt of an AID 254
 print displayed information 252
 standard attribute/printer control character list
 (DFHBMSCA) 222, 255
 display partitioning 165
 display register (EDF) 55
 display trigger field 177
 disposition and message routing 197
 distributed report printing, report controller 300
 distributed transaction processing (DTP) 239
DL/I
 access scheduling 115
 and EDF 59
 call check 119
CALL statement 115
 data base access 116
 data base call 118
EXEC DL/I command 105
 interface block (DIB) 108
 response codes 119
 sync points 405
 work area 117
DLI option 17
DSATTS operand 211
DSIDERR condition 100
DSSTAT condition 329
DTP (distributed transaction processing) 239
 dual screen mode (EDF) 51
DUMP command 396
 dump control 396
DUMPCODE option 396
DUPKEY condition 100
DUPREC condition 100
 dynamic program 23
 dynamic storage 12

E

ECADDR option 347
EDF (see execution diagnostic facility)
EDF option 17, 71
EI option 393
EIB (see **EXEC** interface block)
EIBAID field 156, 157, 413
 examining contents 224
EIBATT field 413
EIBCALEN field 413
EIBCOMPL field 413
EIBCONF field 413
EIBCPOSN field 413
EIBDATE field 340, 414
EIBDS field 414
EIBEOC field 414

EIBERR field 414
EIBERRCD field 414
EIBFMH field 414
EIBFN field 33
EIBFREE field 415
EIBNODAT field 416
EIBRCODE field 33, 416
EIBRECV field 418
EIBREG argument 14
EIBREQID field 418
EIBRESP field 418
EIBRESP2 field 320, 419
EIBRLDBK field 419
EIBRSRCE field 420
EIBSIG field 420
EIBSYNC field 420
EIBSYNRB field 420
EIBTASKN field 420
EIBTIME field 340, 420
EIBTRMID field 420
EIBTRNID field 420
 end browse operation 97
 end of message (EOM) order 163
END-EXEC delimiter (COBOL) 5
 end-of-command delimiter 5
ENDBR command 97
ENDDATA condition 349
ENDFILE condition 101
ENDINPT condition 290
ENQ command 353
ENQBUSY condition 354
 enqueue upon resource 353
ENTER command 392
ENTER key 66, 156
ENTER option 156, 254
ENTRY option 363
 entry point address 23
 entry point, trace 389
 entry to assembler program 12
 entry-sequenced data set (ESDS) 76
ENTRYNAME option 393
ENVDEFERR condition 350
EOC condition
 basic mapping support 236
 terminal control 290
EODS condition
 basic mapping support 236
 batch data interchange 329
 terminal control 290
EOF condition 290
EOM (end of message) order 163
EPILOG option 17
EQUAL option 98
 equated symbols 6
 erase all unprotected fields 253
ERASE option 153, 231
 terminal control 284
ERASEAUP option 153, 231
ERROR condition 32
 error handling 22
ERRTERM option 231
ESCAPE option 306, 314
ESDS (entry-sequenced data set) 76
 establish a sync point 406
event
 control area, timer 341
 monitoring point 389
 waiting for 342
exceptional conditions
 abnormal termination recovery 387
 basic mapping support 157, 236
 batch data interchange 329
 description 31
 file control 100
 IHANDLE CONDITION command 34
 IGNORE CONDITION command 35
 interval control 349
 journal control 403
 list of 35
 partitions 175
 program control 364
 storage control 366
 task control 354
 temporary storage control 377
 terminal control 290
 trace control 394
 transient data control 371
exceptional conditions for the report controller 320
exclusive control
 DAM 91
 releasing (UNLOCK) 95
 VSAM 88
EXEC CICS command format 5
EXEC DLI command 59, 105
EXEC interface block (EIB)
 description 41
 fields 413
execution diagnostic facility (EDF)
 CEBR initiation 69
 CEDF transaction 51
 displays 52, 57
 dual screen mode 51
 EXEC DLI command 59, 105
 functions 49
 installing 51
 invoking 51
 program labels 58
 pseudoconversational program 58
 security rules 51
 single screen mode 51
 terminal sharing 54
exit (see abnormal termination recovery)
exit from ASM program 12
expanded area (CL interpreter) 66
expiration time
 notification when reached 341

specifying 339
EXPIRED condition 350
EXTATT operand 211
EXTDS option 44
extended attribute 162
extended color 137
EXTRACT ATTACH command 259
EXTRACT LOGONMSG command 249
EXTRACT PROCESS command 261
EXTRACT TCT command 259
extrapartition destination 368

F

FACILITY option 44
facility, alternate 239
facility, principal 239
FCI option 44
FCT option 396
FE option 17
field concepts, 3270 136
field data format (BMS) 148, 149
field definition macro (BMS) 144, 206
field edit built-in function 409
field group 144
field of EIB 41
FIELD option 409
field outlining 139
field separator operand 211
fields, EIB 413
file control
 an overview 75
 DAM data sets 89
 data set identification 78
 deleting VSAM records 95
 end browse operation 97
 exceptional conditions 100
 options 98
 read a record 93
 read next record 96
 read previous record 96
 release exclusive control 95
 reset start for browse 97
 specify start for browse 95
 update a record 94
 VSAM data sets 87
 writing new record (WRITE) 94
FIND command (CEBR) 70
flag byte, route list 200
FLAG option 17
FLDSEP operand 211
FLENGTH option 241, 314, 363, 366, 396
FLENGTH option, report controller 298
floating maps 186
FMH (see function management header)
FMH option 285, 347

FMIIPARM option 231
FOOT option 314
footing feature of the spooler 302
FOOTNUM option 315
form feed control (BMS) 159
format
 command 5
 data 131
 trace table 390
format conversion, report controller 300
format of a report 300
FORMATTIME option 340
FORMFEED option 153, 164, 231
FORMS option 315
FREE command 250, 259
free main storage 365
FREEKB option 153, 231
FREEMAIN command 365
FREEMAIN restriction 23
FROM option 315
 basic mapping support 231
 batch data interchange 327
 dump control 396
 file control 98
 interval control 347
 journal control 402
 temporary storage control 376
 terminal control 285
 trace control 393
 transient data control 370
FROM option, report controller 298
FROMLENGTH option 241
FROMLENGTH option 285
FRSET option 153, 231
full function BMS 181
full function logical unit (3790) 279
fullword length option 241
FUNCERR condition 329
function management header (FMH) 247
function that is unsupported 33

G

GCHARS option 44
GCODES option 45
GDDM coordination (BMS)
 GDDM GSFLD call 154
 GDDM PSRSRV call 143
 graphic hole 154
 restriction with partitions 175
GDS option 17
general banking terminal system (see 2980)
generic key 78
GENERIC option 98
GET command (CEBR) 70
get main storage 365

GETMAIN command 365
graphic hole 154
green parameter of COLOR operand 209
GRPNAMe operand 144, 211
GTEQ option 98

H

HANDLE ABEND command 385
HANDLE AID command 156, 178, 254
HANDLE CONDITION command 34
hardware print key 158
HEAD option 315
HEADER operand 187
 map definition macros 212
HEADER option 192, 231
heading feature of the spooler 302
HEADNUM option 315
hhmmss argument 6
highlighting 137
HIGHLIGHT operand 212
HIGHLIGHT option 45
HOLD option 315, 363
HOLD option, report controller 299
HONEY option 231
horizontal picture element 170
host command processor LU (3650/3680) 273
host conversational (3270) LU (3650) 274
host conversational (3653) LU (3650)
hpel (horizontal picture element) 170
HTAB operand 212

I

I/O work area in DL/I 117
IC attribute 154
identification
 DAM record 89
 data set 78
 destination 323
 VSAM record 87
IGNORE CONDITION command 35
IGREQCD condition
 basic mapping support 236
 batch data interchange 329
 terminal control 290
IGREQID condition 236
ILLOGIC condition 101
immediate light pen field 157
INBFMH condition 290
inbound FMH 247
index, alternate (AIX) 76
indicator lights (MSR) 176

indirect destination 368
INITIAL operand 212
initialize main storage 365
initiate a task (see start a task)
INITIMG option 366
INPARTN option 45, 229, 232
input data
 chaining of 244
 unsolicited 248
input data set 11
input operation without data 253
input operations 135
input partition 168, 173
INQUIRE command 41
inquiry logical unit (3790) 280
insert-cursor indicator 139
installing EDF 51
installing the CL interpreter 68
integrity of data 77
interactive logical units 278
interface processor DFHEAI 12
interleaving conversation with message routing 197
interpreter
 installation 68
 invoking 61
 screen layout 62
 security rules 68
 variables 65
interpreter logical unit (3650) 275
interrogate a data set 324
interval control
 cancel interval control command 345
 delay processing of task 340
 exceptional conditions 349
 expiration time 339
 format of date and time 340
 notification when specified time expires 341
 options 346
 request current time of day 340
 retrieve data stored for task 344
 specifying request identifier 340
 start a task 342
 wait for event to occur 342
INTERVAL option
 basic mapping support 232
 interval control 347
INTO option
 basic mapping support 232
 batch data interchange 327
 file control 98
 interval control 347
 temporary storage control 376
 terminal control 285
 transient data control 370
intrapartition destination 367
invalid characters 301
INVERRTERM condition 237
INVITE option 244, 285

INVLDC condition 237
INVMPSZ condition 237
invoking EDF 51
invoking the CL interpreter
 CECI transaction 61
 CECS transaction 61
INVPARTN condition 237
INVPARTNSET operand 237
INVREQ condition
 basic mapping support 237
 file control 101
 interval control 350
 journal control 403
 program control 364
 temporary storage control 377
 terminal control 290
 trace control 394
INVTSREQ condition 350
IOERR condition
 file control 101
 interval control 350
 journal control 403
 temporary storage control 377
 transient data control 371
ISCINVREQ condition
 file control 102
 temporary storage control 377
 transient data 371
ISSUE ABEND command 261
ISSUE ABORT command 326
ISSUE ADD command 325
ISSUE CONFIRMATION command 261
ISSUE COPY command 253
ISSUE DISCONNECT command 242, 248
ISSUE END command 326
ISSUE ENDFILE command 277
ISSUE ENDOUTPUT command 277
ISSUE EODS command 275
ISSUE ERASE command 326
ISSUE ERASEAUP command 253
ISSUE ERROR command 261
ISSUE LOAD command 275
ISSUE NOTE command 326
ISSUE PASS command 248
ISSUE PREPARE command 249
ISSUE PRINT command 158, 252
ISSUE QUERY command 324
ISSUE RECEIVE command 324
ISSUE REPLACE command 325
ISSUE RESET command 242
ISSUE SEND command 326
ISSUE SIGNAL command 242
ISSUE WAIT command 327
ITEM option 376
ITEMERR condition 377
IUTYPE option 285

J

JCL option 315
JFILEID option 402
JIDERR condition 403
JOURNAL command 401
journal control
 create a journal record 401
 exceptional conditions 403
 journal records 399
 options 402
 output synchronization 400
JTYPEID option 402
JUSFIRST option 232
JUSLAST option 232
JUSTIFY operand
 effect on map positioning 187
 map definition macros 213
JUSTIFY option 232

K

KATAKANA option 45
KEEP option, report controller 299
key
 alternate (secondary) 76
 generic 78
 physical 89
key-sequenced data set (KSDS) 75
KEYLENGTH option
 batch data interchange 327
 file control 98
 remote data set 86
KEYNUMBER option 328
keystroke overlapping 166
keyword fields on screen 139
keyword length 241
KSDS (key-sequenced data set) 75

L

label argument 6
LABEL option 387
LANG operand 214
LANGLVL option 17
LAST option 232
 bracket protocol 248
 terminal control 285
LDC operand 214
LDC option
 basic mapping support 232

description of 272
terminal control 285
LDCMNEM option 45, 229
LDCNUM option 45, 229
LENGERR condition 238
 batch data interchange 329
 file control 102
 interval control 350
 journal control 403
 storage control 366
 temporary storage control 377
 terminal control 291
 transient data control 371
LENGTH operand 214
LENGTH option
 basic mapping support 233
 batch data interchange 328
 built-in function 409
 default (assembler language) 7
 default (PL/I) 9
 dump control 397
 file control 99
 interval control 347
 journal control 402
 program control 363
 storage control 366
 task control 354
 temporary storage control 376
 terminal control 286
 transient data control 370
levels, application program logical 355
LIBRARIAN (VSE/SP) library management,
 COBOL 15, 28
LIBRARIAN (VSE/SP) library management, PL/I 16, 30
library management, COBOL
 LIBRARIAN (VSE/SP) 15, 28
 MAINT (DOS/VS(E)) 15, 28
library management, PL/I
 LIBRARIAN (VSE/SP) 16, 30
 MAINT (DOS/VS(E)) 16, 30
light pen
 detectable field 138
 handling in program 156
light pen detectable field 156
LIGHTPEN option 156, 254
LINE command (CEBR) 70
LINE operand 187, 214
 line width for printer 159
 line, communication 242
LINEADDR option 286
LINECOUNT option 18
 linelength feature of the spooler 302
LINELENGTH option 316
LINES option 316
 lines/page feature of the spooler 301
LINK command 355
 link to program anticipating return 355
LIST option
 basic mapping support 233
 listing data set 12
 literal constant 6
 load a map set 145
 load a program, table, or map 357
LOAD command 357
 load module size 23
 local copy key 158
 locality of reference 22
locate-mode
 SERVICE RELOAD statement 28
 to minimize the working set 22
lockout (see deadlock prevention)
LOG option 316
LOG option, report controller 299
log report 297
logical device code (LDC option) 272
logical device components
 basic support 175
 page overflow 191
logical levels, application program 355
logical messages (BMS)
 completing a logical message 183
 cumulative text processing 192
 device controls 194
 direct terminal output 183
 example of use 192
 floating maps 187
 introduction 181
 map positioning 187
 message recovery 182
 page overflow 190
 PAGING output 184
 purging a logical message 183
LOGICAL option 316
logical record presentation 246
logical recovery 311
logical unit of work (LUW) 405
logical units
 batch 278
 conversing with (CONVERSE) 242
 facilities for 244
 interactive 278
 LUTYPE6.1 258
 LUTYPE6.2 260
 pipeline 272
 reading data from 324
 terminal control 241
 writing data to 325
 terminal control 241
 3270 Information Display System 269
 3270 SCS Printer 269
 3270-Display (LUTYPE2) 270
 3600 (3601) 272
 3600 (3614) 273
 3600 pipeline 272
 3650 host conversational (3270) 274

3650 host conversational (3653) 274
 3650 interpreter 275
 3650 pipeline 272
 3650/3680 host command processor 273
 3767 interactive 278
 3770 batch 278
 3770 interactive 278
 3790 (3270-display) 281
 3790 (3270-printer) 281
 3790 full function 279
 3790 inquiry 280
 3790 SCS printer 280
 look aside query 167
 LUNAME option 286
 LUTYPE2 (3270-Display LU) 270
LUTYPE4
 batch data interchange 323
 logical record presentation 246
 logical unit 257
LUTYPE6.1 logical unit 258
LUTYPE6.2 logical unit 260
 LUW (logical unit of work) 405
 L40 option 233
 L64 option 233
 L80 option 233

M

macro global bit (&DFHEIMX) 15
 macro instruction
 command equivalent 425
 field definition, DFHMDF 143
 map definition, DFHMDI 142
 map set definition, DFHMSD 141
 partition definition (DFHPDI) 170
 partition set definition (DFHPSD) 170
 used with commands 15, 23
 magnetic slot reader (MSR)
 audible alarm 176
 DFHMSRCA copy book 176
 DFHMSRCA set of constants 176
 indicator lights 176
 MSRCONTROL option 176
MAIN option 376
 main storage
 dumping (DUMP) 396
 initialize 365
 obtain 365
 releasing (FREEMAIN) 365
 temporary data 373
MAINT (DOS/VS(E)) library management,
 COBOL 15, 28
MAINT (DOS/VS(E)) library management, PL/I 16,
 30
 map definition macro (BMS) 206
 map definition macro, BMS 142

MAP option 233
 map positioning 187
 map set definition macro 141
 map set definition macro (BMS) 205
 map set loading 145
 map set name 145
 map set sufficing 145
 map set sufficing and partitions 167
 map size 24
MAPATIS operand 215
MAPCOLUMN option 45, 229
MAPFAIL condition 157, 238
MAPHEIGHT option 45, 229
MAPLINE option 45, 229
MAPNAME option 317
MAPONLY option 151, 233, 317
 mapping input data (RECEIVE MAP) 154
 maps
 ASM sample programs 452
 assembling 145
 cataloging 145
 COBOL sample programs 494
 copying symbolic description 147
 physical 132
 PL/I sample programs 533
 symbolic 132
 temporary modification 148
MAPSET option 233, 317
MAPSFIX operand 221
MAPWIDTII option 45, 229
MARGINS option 18
MASSINSERT option 99
MAXLENGTH option 241
MAXLENGTH option 286
MCC option 317
MDT (modified data tag) 138
 message
 teletypewriter 251
 title 198
 message area of screen 140
 message length, teletypewriter 251
 message recovery (BMS) 182
 message routing 195
 message switching transaction, CMSG 201
 minimum function BMS 135
 mixed mode application program 15, 23
MMDDYY option
MODE operand 215
 modified data tag (MDT) 138
 modular program 22
MONITOR option 394
 monitoring point (ENTER command) 389
MONTHOFYEAR option 347
 move-mode 22
 MSR (magnetic slot reader) 176
 MSR option 233
MSRCONTROL option 45, 176
 multiple base registers 13

multithreading 21

N

name argument 6
naming restriction 15
NETNAME option 46, 286
neutral parameter of COLOR operand 209
NEXT option 376
NLEOM option 234
NOAUTOPAGE option 183, 234
NOCC option 317
NOCHECK option 347
NOCONV (no conversion) option, report controller 300
NOCONV option 317
NOCONV option, report controller 304
NODE option 318
NODEBUG option 17
NOEDF option 17
NOEPILOG option 17
NOHANDLE option 32
NOBUFSP condition 403
nondisplay field 138
NONUM option 18, 318
NONVAL condition 291
NOOPSEQUENCE option 18
NOOPT option 18
NOOPTIONS option 18
NOPASSBKRD condition 291
NOPASSBKWR condition 291
NOPROLOG option 18
NOQUEUE option 286
normal intensity field 137
NOSEP option 318
NOSEQ option 19
NOSEQUENCE option 19
NOSOURCE option 19
NOSPACE condition
 file control 102
 temporary storage control 377
 transient data control 371
NOSPIE option 19
NOSTART condition 291
NOSTG condition 366
NOSUSPEND option 33, 286
 ENQ command 354
 GETMAIN command 366
 JOURNAL command 402
 READQ TD command 370
 WRITEQ TS command 376
NOTALLOC condition 291
notation, syntax 4
NOTAUTH condition 33
 file control 102
 interval control 350

journal control 403
program control 364
temporary storage control 377
transient data control 371
NOTFND condition
 file control 102
 interval control 351
NOTOPEN condition
 file control 102
 journal control 403
 transient data control 372
NOTRUNC compiler option 28
NOTRUNCATE option 286
NOVBREF option 19
NOWAIT option 328
NOXREF option 19
null lines and 3270 printer 159
null map 142
NUM option 18
numeric-only field (3270 attribute character) 136
NUMITEMS option 376
NUMREC option
 batch data interchange 328
 file control 99
NUMTAB option 46

O

OBFMT operand 215
OCCURS operand 215
OPCLASS option
 ASSIGN command 46
 basic mapping support 234
operator class codes 198
operator identification card reader 156
OPERID option 156, 254
OPERKEYS option 46
OPERPURGE option 183, 234
OPID option 46
OPMARGINS option 18
OPSECURITY option 46
OPSEQUENCE option 18
OPT option 18
optimization feature (COBOL) 27
option length 241
options
 abnormal termination recovery 387
 ADDRESS command 42
 ASSIGN command 43
 basic mapping support 229
 batch data interchange 327
 dump control 396
 execution time (PL/I STAE) 29
 file control 98
 HANDLE AID command 254

HANDLE CONDITION command 34
interval control 346
journal control 402
program control 363
report controller 313
STAE (PL/I) 29
storage control 366
task control 354
temporary storage control 376
terminal control 283
trace control 393
transient data control 370
translator 16
OPTIONS option 18
OPTIONS(MAIN) specification 16, 30
outboard formatting
 basic mapping support 179
 effect of partitions 174
outbound FMH 248
OUTLINE operand 215
OUTLINE option 46
output control (2980) General Banking Terminal System 267
output data set 11
output data, chaining of 246
output operations 136
output partition 168
output to common buffer (2980) 267
OVERFLOW condition 190, 238
overlapping keystroking 166
overlays 23
overtyping EDF displays 57

P

PA (program access) key 156
PA option 156, 254
page buffer (3270) 163
page fault 22
page numbering feature of the spooler 302
page overflow (BMS) 190
page overflow example 192
page width for printer 159
PAGENUM option 46, 229
paging (see terminal operator paging) 184
PAGING option 184, 234
partition definition macro (BMS) 170, 220
PARTITION JUMP key 165
partition set 167, 168
partition set definition macro (BMS) 170, 220
partition set suffixing 145
partitions
 active partition 165
 base state 169
 character cell 174
 cumulative processing 195

cursor 165
description 165
display control 165
ENTER key 165
exceptional conditions 175
GDDM restriction 175
outboard format 195
outboard formatting 174
page overflow 191
PARTITION JUMP key 165
partition set 167
presentation space 168
terminal sharing 195
transaction partitions 167
unpartitioned state 169
viewport 168
window 168
within a map set 171
PARTN operand 215
PARTN option
 basic mapping support 235
PARTNFAIL condition 238
PARTNPAGE option 46, 229
PARTNS option 46
PARTNSET option 46, 229, 235
PASSBK option 286
passbook control (2980) 266
passing
 session 248
passing control
 anticipating return (LINK) 355
 on receipt of an AID (HANDLE AID) 254
 without return (XCTL) 356
passing data
 to new tasks 344
 to other program 358
PCB (program communication block) 115
PCB address 116
PCT option 397
pel (picture element) 174
PERFORM option 394
PF (program function) key
 BMS 156
 CL interpreter 66
 EDF 52
PF (program function) key (CECI/CECS) 66
PF option 156, 254
PFXLENG option 402
PGMIDERR condition
 abnormal termination recovery 387
 program control 364
physical key 89
physical map 132
PHYSICAL option 318
physical recovery 311
PICIN operand 215
PICOUT operand 216
picture element (pel) 174

pink parameter of COLOR operand 209
pipeline logical unit 272
PIPLENGTH option 287
PIPLIST option 287
PL/I
 argument value 8
 compilers supported 29
 LENGTH option default 9
 OPTIONS(MAIN) specification 30
 PROCEDURE statement 16
 program segments 30
 restrictions 29
 sample programs 509
 STAE option 29
 translated code 15
POINT command 259
pointer-ref argument 6
pointer-value argument 6
POP HANDLE command 35
POS operand 144, 216
POST command 341
posting timer event control area 341
PPT option 397
PREFIX option 402
pregenerated modules 134
presentation space 168
pretranslated code
 COBOL 15, 28
 PL/I 16, 30
principal facility 239
PRINSYSID option 46
print displayed information 252
PRINT option 153
 basic mapping support 235
 batch data interchange 328
printer and BMS text 164
printer control character list (DFHBMSCA) 222, 255
printer destination 307
printer spooling options
 ASA 313
 BOTHNUM 313
 CLASS 313
 COPIES 313
 DATETIME 313
 DELETE 313
 DESTINATION 314
 ESCAPE 306, 314
 FLENGTH 314
 FOOT 314
 FOOTNUM 315
 FORMS 315
 FROM 315
 HEAD 315
 HEADNUM 315
 HOLD 315
 JCL 315
 LINELENGTH 316
 LINES 316

LOG 316
LOGICAL 316
MAPNAME 317
MAPONLY 317
MAPSET 317
MCC 317
NOCC 317
NOCONV 317
NODE 318
NONUM 318
NOSEP 318
PHYSICAL 318
PRINTFAIL 318
PRIORITY 318
RELEASE 318
REPORT 318
RESUME 319
RSL 319
SCS 319
SEP 319
TITLE 319
TOKEN 319
T3270 320
USERDATA 320
USERID 320
printer spooling subsystem 294
printers
 basic mapping support (BMS) 163
 device independence 164
 format conversion, report controller 300
 FORMFEED option 164
 NLEOM option 163
 printing displayed data 158
 SCS and non-3270 printers 164
 starting a printer task 158
 3270 printer page width 158
 3270 printer using NLEOM 163
 3270 printer without NLEOM 163
 3270 printers and blank lines 158
PRINTFAIL option 318
printing contents of screen 158
PRIORITY option 318
PROCESS option 287
PROCLENGTH option 287
PROCNAME option 287
PROFILE option 287
program access (PA) key 156
program communication block (PCB) 115
program control
 CL interpreter 67
 deleting loaded program 358
 exceptional conditions 364
 linking to another program 355
 load a program, table, or map 357
 options 363
 passing data to other program 358
 program logical levels 355
 returning program control 356

transfer program control 356
program function key (see PF key)
program labels in EDF 58
PROGRAM option
 abnormal termination recovery 387
 dump control 397
 program control 363
 terminal control 288
program segments
 COBOL 28
 PL/I 30
program size 23
program specification block (PSB) 115
program testing 49
programmed symbols 138
programming techniques
 COBOL 24
 general 21
 PL/I 29
programs
 checking out pseudoconversational 58
programs, sample (ASM) 427
programs, sample (COBOL) 471
programs, sample (PL/I) 509
programs, sample (report controller) 549
PROLOG option 18
PROTECT option 347
protected field 137
PS operand 217
PS option 47
PSB (program specification block) 115
PSB release 118
PSB scheduling 116
PSEUDOBIN option 288
pseudoconversational programming 58
PURGE command (CEBR) 70
PURGE MESSAGE command 183
PUSH HANDLE command 35
PUT command (CEBR) 70

Q

QBUSY condition 372
QIDERR condition
 temporary storage control 378
 transient data control 372
QNAME option 47
quasi-reenterability 21
question mark (CL interpreter) 62
QUEUE command (CEBR) 69
QUEUE option 288
 interval control 348
 temporary storage control 376
 transient data control 370
queue, temporary storage 373
QUOTE option 19

QZERO condition 372

R

RBA (relative byte address) 75
RBA option 99
RDATT condition
 basic mapping support 238
 terminal control 291
reactivate an ABEND exit 385
read ahead queueing feature 244
read attention 264
READ command 93
reading
 batch data interchange record 324
 data from a display (RECEIVE MAP) 154
 data from temporary storage queue 375
 data from terminal or LU 241
 data from transient data queue 369
 file control record 93
 next record when browsing 96
 previous record in VSAM browse 96
READNEXT command 96
READPREV command 96
READQ TD command 369
READQ TS command 375
RECEIVE command 241
RECEIVE MAP command 154, 225
RECEIVE PARTN command 173, 225
RECFM option 288
record
 deleting VSAM 95
 identification 87, 89
 journal 399
 reading 93, 324
 requesting next number 326
 updating 94, 325
 writing new (adding) 94, 325
record descriptions
 ASM sample programs 469
 COBOL sample programs 508
 PL/I sample programs 548
recovery
 abnormal termination 385
 and debugging 379
 sequential terminal support 381
 sync point 405
recovery level of report
 logical recovery 310
 no recovery 310
 physical recovery 310
red parameter of COLOR operand 209
reenterability 21
register contents in ASM 12
relative byte address (RBA) 75
relative record data set (RRDS) 76

relative record number (RRN) 76
RELEASE command 358
RELEASE option 183, 235, 318
RELEASE option, report controller 299
releasing
 a PSB 118
 area of main storage 365
 exclusive control (UNLOCK) 95
relinquish communication line 242
RELOAD operand of DFHPPT macro 358
relocatable expression 6
remote data set, KEYLENGTH option 86
reopening a resumable report 298
report controller 294
 NOCONV option 304
 report name 295
 report title 295
 userdata 295
report controller command 311
report controller exceptional conditions 320
report format 300
report name, for report controller 295
report number 296
REPORT option 318
REPORT option, report controller 298, 299
REPORT option, SPOOL... commands 295
report recovery 310
report recovery level 310
report title, for report controller 295
report type
 log 296
 resumable 296
 standard 296
REQID option 182
 basic mapping support 235
 file control 99
 interval control 348
 journal control 402
request/response unit (RU) 244
RESET option 387
reset start for browse 97
RESETBR command 97
RESOURCE option 288, 354, 394
resource scheduling 353
RESP option 32
RESP values 418
response codes (DL/I) 119
RESP2 option 32
RESP2 values 320, 419
RESTART option 47
restrictions
 assembler language 24
 COBOL 24
 PL/I 29
resumable report 296
RESUME option 319
RESUME option, report controller 298, 299
RETAIN option 183, 235

RETPAGE condition 238
RETRIEVE command 344
retrieve data stored for task 344
RETURN command 356
return facility to CICS 250
return program control 356
returning mapped data to a program 201
REWRITE command 94
REWRITE option 377
RIDFLD option
 batch data interchange 328
 file control 99
ROLLBACK option 406
ROLLEDBACK condition 406
ROUTE command 195
route list (LIST option) 198, 199, 200
routing messages (ROUTE) 197
routing terminal 197
RPROCESS option 288
RRDS (relative record data set) 76
RRESOURCE option 289
RRN option
 batch data interchange 328
 file control 100
RSL option 319
RTEFAIL condition 238
RTERMID option 348
RTESOME condition 238
RTRANSID option 348
RU (request/response unit) 244

S

SA (set attribute) order 162
sample program
 browse (ASM) 436
 browse (COBOL) 479
 browse (PL/I) 517
 inquiry/update (ASM) 429
 inquiry/update (COBOL) 473
 inquiry/update (PL/I) 511
 low balance report (ASM) 449
 low balance report (COBOL) 491
 low balance report (PL/I) 530
 low balance report, report controller 549
operator instruction (ASM) 428
operator instruction (COBOL) 472
operator instruction (PL/I) 510
order entry (ASM) 442
order entry (COBOL) 484
order entry (PL/I) 523
order entry queue print (ASM) 446
order entry queue print (COBOL) 488
order entry queue print (PL/I) 527
SBA (set buffer address) order 163

schedule a PSB 116
 schedule access (DL/I) 115
 schedule use of resource by task 353
 Screen Definition Facility/CICS (SDF/CICS) 132
 screen layout (CL interpreter)
 command input area 62
 information area 63
 status area 62
 screen layout design
 application data area 139
 data fields 139
 input operations 135
 keyword fields 139
 message area 140
 output operations 136
 requirements 139
 stopper fields 139
 title area 139
 screen partitioning 165
 screen size 140
 SCRНHT option 47
 SCRНWD option 47
 scrolling 166
 SCS option 319
 SCS printer logical unit (3790) 280
 SDF/CICS (Screen Definition Facility/CICS) 132
 secondary key 76
 security of a report 310
 public 310
 security key 310
 security rules
 CL interpreter 68
 EDF 51
 segment search argument (SSA) 117
 segments, program
 COBOL 28
 PL/I 30
 SELNERR condition 329
 semicolon delimiter (PL/I) 5
 send asynchronous interrupt 242
 SEND command 241
 SEND CONTROL command 153, 227
 send data to output device 326
 SEND MAP command 150, 226
 SEND PAGE command 183
 SEND PARTNSET command 172
 SEND TEXT command 162, 164, 226
 SEND TEXT MAPPED command 203
 SEND TEXT NOEDIT command 203
 SEND/RECEIVE mode 244
 SEND/RECEIVE protocol 244
 SEP option 319
 SEQ option 19
 SEQUENCE option 19
 sequential retrieval (see browsing)
 sequential terminal support 381
 SERVICE RELOAD statement (COBOL) 27
 SESSBUSY condition 291
 SESSION option 289
 session-oriented information (LU6) 259
 SESSIONERR condition 291
 set attribute (SA) order 162
 set buffer address (SBA) order 163
 SET command 41
 SET option 201
 basic mapping support 235
 batch data interchange 328
 file control 100
 interval control 348
 program control 363
 storage control 366
 temporary storage control 377
 terminal control 289
 transient data control 371
 share option, VSAM 77
 SIGDATA option 47
 SIGNAL condition 248, 291
 SINGLE option 394
 single screen mode (EDF) 51
 single threading 21
 SIT option 397
 SIZE operand 217
 skip-sequential processing 80
 SO/SI creation 139
 SOSI operand 218
 SOSI option 47
 SOURCE option 19
 SPACE option 19
 spanned records 76
 SPIE option 19
 SPOOL commands
 TOKEN option 295
 SPOOL... commands
 REPORT option 295
 SPOOLCLOSE command 312
 spooler automated feature 301
 spooler recovery 310
 SPOOLOPEN command 298, 311
 TITLE option 295
 USERDATA option 295
 SPOOLOPEN ESCAPE command 312
 SPOOLOPEN MAPNAME command 311
 SPOOLOPEN RESUME command 312
 SPOOLWRITE command 312
 SPOOLWRITE MAPNAME command 313
 SQL/DS (Structured Query Language)
 application programming 331
 SQL/DS publications vi
 SSA (segment search argument) 117
 STAE option (PL/I) 29
 standard attribute/printer control character list
 (DFIIBMSCA) 222, 255
 standard CICS terminal support 257
 standard function BMS 161
 standard report 296
 start a task 343
 START command 342

STARTBR command 95
STARTCODE option 47
STARTIO option 403
STATIONID option 47
status flag byte, route list 200
status of partition 169
stopper fields on screen 139
storage (see main storage)
storage area length 43
storage control 365
STORAGE operand 218
STORAGE option 397
STRFIELD option 289
Structured Query Language (see SQL/DS)
stub (see interface processor)
SUBADDR option 328
subroutines 22
SUFFIX operand 145, 218, 221
suffixing, map/partition sets 145
supplied constants (DFHMSRCA) 176
suspend a task 248, 353
SUSPEND command 353
switched line disconnection 242
symbol set 169
symbolic cursor positioning 154
symbolic description map
 block data format 179
 copying 147
 definition 132
 field data format 148, 149
symbolic register DFHEIPLR 14
sync point 405
sync point and the spooler 311
sync points in reports 299
synchronization level 260
synchronizing
 journal output 400
 journal output (WAIT JOURNAL) 402
 terminal input/output 242
SYNCLEVEL option 289
SYNCPOINT command 406
syntax notation 4
syntax style 5
SYSBUSY condition 292
SYSID option
 ASSIGN command 47
 file control 100
 interval control 348
 temporary storage control 377
 terminal control 289
 transient data control 371
SYSIDERR condition 372
 file control 103
 interval control 351
 temporary storage control 378
 terminal control 292
system information, access to 41
SYSTEM option 394

system trace entry point 389
System/3 262
System/370 262
System/7 262

T

tab character 164
TABLES option 397
task control 353
task identification 243
task initiation (see start a task)
TASK option 397
task suspension 248
TCI option 398
TCTUA option 42
TCTUALENG option 47
techniques, programming 21
teletypewriter programming 251
TELLERID option 48
temporary storage
 auxiliary 373
 browse transaction (CEBR) 69
 exceptional conditions 377
 main 373
 options 376
 queue 373
TERM operand 218
TERMCODE option 48
TERMERR condition
 terminal control 292
TERMID option
 interval control 349
 terminal control 289
TERMIDERR condition
 interval control 351
 terminal control 292
terminal code table 202
TERMINAL command (CEBR) 69
terminal control
 an overview 239
 bracket protocol (LAST option) 248
 BTAM programmable device 250
 chaining of input data 244
 chaining of output data 246
 converse with terminal or LU 242
 definite response 247
 detecting attention condition (SIGNAL) 248
 disconnect a switched line 242
 display device operations 252
 exceptional conditions 290
 facilities for logical units 244
 facilities for terminals 242
 facilities for terminals and LUs 241
 FMH, inbound 247

FMH, outbound 248
function management header (FMH) 247
handle attention identifier 156
interactive logical units 278
logical record presentation 246
LUTYPE2 (3270-Display LU) 270
map input data (RECEIVE MAP) 154
options 283
passing a session 248
pipeline logical unit 272
print (ISSUE PRINT) 158
read attention 264
reading data from terminal or LU 241
relinquish communication line 242
standard CICS terminal support 257
sync point processing 249
synchronize terminal I/O 242
System/3 262
System/370 262
System/7 262
teletypewriter programming 250
terminate a session 248
unsolicited input 248
VTAM logon data 249
writing data to terminal or LU 241
2260 Display Station 264
2265 Display Station 264
2741 Communication Terminal 264
2770 Data Communication System 265
2780 Data Transmission Terminal 266
2980 General Banking Terminal System 266
3270 (BTAM or TCAM supported) 268
3270 field concept 136
3270 Information Display System logical unit 269
3270 SCS Printer logical unit 269
3270-Display LU (LUTYPE2) 270
3600 (3601) logical unit 272
3600 (3614) logical unit 273
3600 pipeline logical unit 272
3650 host conversational (3270) LU 274
3650 host conversational (3653) LU 274
3650 interpreter logical unit 275
3650 pipeline logical unit 272
3650/3680 host command processor LU 273
3660 275
3735 276
3740 277
3767 interactive logical unit 278
3770 batch logical unit 278
3770 interactive logical unit 278
3790 (3270-display) logical unit 281
3790 (3270-printer) logical unit 281
3790 full function logical unit 279
3790 inquiry logical unit 280
3790 SCS printer logical unit 280
4700 Finance Communication System 282
7770 Audio Response Unit 282
terminal operator paging
example of paging message 192
initiating the paging transaction 183
message recovery 182
operator paging commands 185
PAGING logical messages 184
TERMINAL option
basic mapping support 235
dump control 398
terminal sharing
CL interpreter 67
EDF 54
terminal-oriented task identification 243
terminating
map set definition 144
processing of data set 326
session 248
task abnormally (ABEND) 387
testing using sequential devices 382
text data format 131
THEN clause (COBOL) 5
time feature of the spooler 303
time field of EIB 41
time of day, requesting (ASKTIME) 340
TIME option
basic mapping support 235
interval control 349
time-initiated transaction (3735) 276
timer event control area 341
TIMESEP option 349
TIOA, mixed mode programs 23
TIOAPFX operand 219
title area of screen 139
TITLE option 198, 235, 319
TITLE option, SPOOLOPEN command 295
title, message 198
TOLENGTH option 241
TOKEN option 319
TOKEN option, report controller 298, 299
TOKEN option, SPOOL commands 295
tokens, in report creation 295
TOLENGTH option 289
TOP command (CEBR) 70
TRACE command 393
trace control 389
auxiliary trace facility 392
controlling trace facility 393
exceptional conditions 394
options 393
trace entry format 392
trace entry point 389
trace facility control 390
trace flags 390
trace table format 390
user trace entry point 392
trace entry format 392
trace entry point 389
trace facility control 390
trace table format 390
TRACEID option 394

TRAILER operand 187, 219
TRAILER option 192, 235
transaction identifier (CEDF) 49
transaction partition 167
transfer program control 356
TRANSID option
 basic mapping support 236
 interval control 349
 program control 363
TRANSIDERR condition 351
transient data control
 automatic task initiation (ATI) 368
 delete intrapartition queue 370
 exceptional conditions 371
 extrapartition destination 368
 indirect destination 368
 intrapartition destination 367
 options 370
 read data from TD queue 369
 write data to TD queue 368
translated code
 ASM 12
 COBOL 15
 PL/I 15
translation tables for 2980 421
translator data set
 input 11
 listing 12
 output 11
translator options 16
TRANSP operand 219
trigger field 177
TRIGGER option 254
TSIOERR condition 238
turquoise parameter of COLOR operand 209
TWA option 42
TWALENG option 48
TYPE operand 219
T3270 option 320

U

UIB (user interface block) 116
UNATTEND option 48
UNEXPIN condition
 basic mapping support 238
 batch data interchange 329
UNLOCK command 95
unpartitioned state 169
unprotected field (3270 attribute character) 136
unsolicited input 248
unsupported function 33
update a record
 batch data interchange 325
 file control 94
UPDATE option 100

upgrade set 76
user interface block (UIB) 116
USER option 394
user trace entry point 389, 392
USERDATA option 320
USERDATA option, SPOOLOPEN command 295
userdata, for report controller 295
USERID option
 ASSIGN command 48
 report controller 320

V

validation 138
VALIDATION option 48
validity of reference 22
VALIDN operand 178, 220
values of arguments 6
variable (CL interpreter) 65
VBREF option 19
vertical forms control 159
vertical picture element 170
viewport 168
VIEWPOS operand 221
VIEWSZE operand 221
virtual storage environment 21
VOLUME option 329
VOLUMELENG option 329
vpel (vertical picture element) 170
VSAM
 data sets 87
VSAM share option 77
VTAB operand 220
VTAM logon data, access to 249

W

WAIT CONVID command 261
WAIT EVENT command 342
WAIT JOURNAL command 402
WAIT option
 basic mapping support 236
 interval control 349
 journal control 403
 of SEND command 242
 terminal control 242, 290
WAIT SIGNAL command 248
WAIT TERMINAL command 242
waiting
 batch data interchange 327
 for event to occur 342
 terminal control operation 242
WHERE clause in EXEC DLI command

literal string 108
window 168
working set 22
WPMEDIA option 329
WRITE command 94
WRITEQ TD command 368
WRITEQ TS command 374
writing
 batch data interchange record 325
 data to temporary storage queue 374
 data to terminal or logical unit 241
 data to transient data queue 368
 file control record 94
writing to a report 298

X

XCTL command 356
XINIT operand 212
XOPTS keyword 16
XREF option 19

Y

YEAR option 349
yellow parameter of COLOR operand 209
YYDDD option 349
YYDDMM option 349
YYMMDD option 349

Z

zero length field 140
zero parameter 213

Numerics

2260 Display Station 264
2265 Display Station 264
2721 Portable Audio Terminal 282
2741 Communication Terminal 264
2770 Data Communication System 265

2780 Data Transmission Terminal 266
2980 General Banking Terminal System 266
 DFII2980 structure 267
 output control 267
 output to common buffer 267
 passbook control 266
 translation tables 421
3270 Information Display System
 (BTAM or TCAM supported) 268
 attribute character 136
 field concepts 136
 input operations 135
 logical unit 269
 screen sizes 140
3600 Finance Communication System 270
 pipeline logical unit 272
 3601 logical unit 272
 3614 logical unit 273
3630 Plant Communication System 273
3650 Store System
 host conversational (3270) LU 274
 host conversational (3653) LU 274
 interpreter logical unit 275
 pipeline logical unit 275
3650/3680 Store System
 full function logical unit 275
 host command processor LU 273
3660 Supermarket Scanning System 275
3680 Programmable Store System
 host command processor LU 273
3735 Programmable Buffered Terminal 276
3740 Data Entry System 277
3767 Communication Terminal
 interactive logical unit 278
3770 Communication System
 batch logical unit 278
 full function logical unit 279
 interactive logical unit 279
3780 Communications Terminal 279
3790 Communication System
 full function logical unit 279
 inquiry logical unit 280
 SCS printer logical unit 280
 3270-display logical unit 281
 3270-printer logical unit 281
4700 Finance Communication System 282
7770 Audio Response Unit 282
8775 Display Terminal
 field validation attribute character 136
 partition support 165
 trigger validation attribute 177

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).

Customer Information Control System
CICS/DOS/VS Version 1 Release 7
Application Programmer's Reference Manual (CL)

**READER'S
COMMENT
FORM**

Order No. SC33-0077-5

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you. Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

Number of your latest Technical Newsletter for this publication . . .

Note: *Staples can cause problems with automated mail-sorting equipment.
Please use pressure-sensitive or other gummed tape to seal this form.*

If you want an acknowledgement, give your name and address below.

Name

Job Title Company

Address

Zip

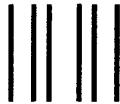
Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

Reader's Comment Form

Fold and tape

Please Do Not Staple

Fold and tape



NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

A standard POSTNET barcode representing the mailing address.

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

International Business Machines Corporation
Department 6R1H,
180 Kost Road,
Mechanicsburg, PA 17055, USA



Fold and tape

Please Do Not Staple

Fold and tape

IBM®



SC33-0077-5
Version 1.7

Program Number
5746-XX3 (CICS/DOS/VS)

Printed in U.S.A.

SC33-0077-05

