

Reference Manual



**HP 3000 Computer Systems**

# **TurboIMAGE Data Base Management System**

**Reference Manual**



19447 PRUNERIDGE AVE., CUPERTINO, CA 95014

Part No. 32215-90050  
E1285

Printed in U.S.A. 12/85

#### **NOTICE**

The information contained in this document is subject to change without notice.

**HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.**

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

This document contains proprietary information which is protected by copyright. All rights are reserved. No part of this document may be photocopied, reproduced or translated to another language without the prior written consent of Hewlett-Packard Company.

# PRINTING HISTORY

New editions are complete revisions of the manual. Update packages, which are issued between editions, contain additional and replacement pages to be merged into the manual by the customer. The dates on the title page change only when a new edition or a new update is published. No information is incorporated into a reprinting unless it appears as a prior update; the edition does not change when an update is incorporated.

The software code printed alongside the date indicates the version level of the software product at the time the manual or update was issued. Many product updates and fixes do not require manual changes and, conversely, manual corrections may be done without accompanying product changes. Therefore, do not expect a one to one correspondence between product updates and manual updates.

First Edition . . . . . Dec 1985. . . . . 32215C.00.00





# LIST OF EFFECTIVE PAGES

The List of Effective Pages gives the date of the most recent version of each page in the manual. To verify that your manual contains the most current information, check the dates printed at the bottom of each page with those listed below. The date on the bottom of each page reflects the edition or subsequent update in which that page was printed.

Effective Pages

Date

all . . . . . Dec 1985



# PREFACE

This manual describes the TurboIMAGE/3000 Data Base Management System for HP 3000 computers.\* It is the reference document for all persons involved in designing and maintaining a data base and for application programmers writing data base access programs.

This manual describes enhancements to IMAGE/3000 based on hardware and operating system improvements along with information previous IMAGE/3000 users will find familiar. The first three sections will prove helpful for new users of the IMAGE data base structure. TurboIMAGE utilities are covered in a separate section which includes syntax and examples. New recovery processes and enhanced MPE user logging are covered in a section which provides information on maintaining the data base.

Designers of TurboIMAGE data bases will find knowledge of the HP 3000 MPE operating and file systems useful in determining the amount of system resources, such as disc space and computation time, needed to maintain a specific data base. Because access to IMAGE data bases requires the use of a host programming language, application programmers need familiarity with at least one of the programming languages available on the HP 3000 computer: COBOL, FORTRAN, Pascal, SPL, BASIC, or RPG.

In addition to this manual, you may need to consult the following manuals:



BASIC Interpreter Reference Manual . . . . .	30000-90026
BASIC/3000 Compiler Reference Manual. . . . .	32103-90001
COBOL/3000 Reference Manual . . . . .	32213-90001
Console Operator's Guide . . . . .	30000-90013
DS/3000 Reference Manual. . . . .	32190-90001
EDIT/3000 Reference Manual . . . . .	30000-90012
Error Messages and Recovery Manual. . . . .	30000-90015
FORTRAN Reference Manual. . . . .	30000-90040
General Information Manual . . . . .	30000-90008
Machine Instruction Set Manual. . . . .	30000-90022
MPE Commands Reference Manual. . . . .	30000-90009
MPE Intrinsics Reference Manual. . . . .	30000-90010
NLS/3000 Reference Manual. . . . .	32414-90001
Pascal/3000 Reference Manual. . . . .	32106-90001
QUERY Reference Manual . . . . .	30000-90042
RPG/3000 Reference Manual. . . . .	32104-90001
System Manager/System Supervisor Manual . . . . .	30000-90014
Systems Programming Language Reference Manual. . . . .	30000-90025
TurboIMAGE Profiler User Guide. . . . .	36914-91001

\* This manual is for 3000 systems operating on MPE V/E or later versions.





# CONVENTIONS USED IN THIS MANUAL

NOTATION	DESCRIPTION
nonitalics	Words in syntax statements which are not in italics must be entered exactly as shown. Punctuation characters other than brackets, braces and ellipses must also be entered exactly as shown. For example:  EXIT;
<i>italics</i>	Words in syntax statements which are in italics denote a parameter which must be replaced by a user-supplied variable. For example:  CLOSE <i>filename</i>
[ ]	An element inside brackets in a syntax statement is optional. Several elements stacked inside brackets means the user may select any one or none of these elements. For example:  $\begin{bmatrix} A \\ B \end{bmatrix}$ User <i>may</i> select A or B or neither.
{ }	When several elements are stacked within braces in a syntax statement, the user must select one of those elements. For example:  $\begin{Bmatrix} A \\ B \\ C \end{Bmatrix}$ User <i>must</i> select A or B or C.
...	A horizontal ellipsis in a syntax statement indicates that a previous element may be repeated. For example:  [, <i>itemname</i> ]...;  In addition, vertical and horizontal ellipses may be used in examples to indicate that portions of the example have been omitted.
	A shaded delimiter preceding a parameter in a syntax statement indicates that the delimiter <i>must</i> be supplied whenever (a) that parameter is included or (b) that parameter is omitted and any <i>other</i> parameter which follows is included. For example:  <i>itema</i> [  <i>itemb</i> ][, <i>itemc</i> ]  means that the following are allowed:  <i>itema</i> <i>itema,itemb</i> <i>itema,itemb,itemc</i> <i>itema,,itemc</i>

# CONVENTIONS (continued)

$\Delta$  When necessary for clarity, the symbol  $\Delta$  may be used in a syntax statement to indicate a required blank or an exact number of blanks. For example:

`SET[(modifier)] $\Delta$ (variable);`

underlining When necessary for clarity in an example, user input may be underlined. For example:

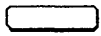
`NEW NAME? ALPHA`


In addition, brackets, braces or ellipses appearing in syntax or format statements which must be entered as shown will be underlined. For example:

`LET var[[subscript]] = value`

**shading**

Shading represents inverse video on the terminal's screen. In addition, it is used to emphasize key portions of an example.



The symbol  may be used to indicate a key on the terminal's keyboard. For example, **(RETURN)** indicates the carriage return key.

**(CONTROL)** *char*

Control characters are indicated by **(CONTROL)** followed by the character. For example, **(CONTROL)Y** means the user presses the control key and the character Y simultaneously.

## Section 1

### INTRODUCTION

TurboIMAGE . . . . .	1-1
TurboIMAGE Enhancements . . . . .	1-2
General Overview . . . . .	1-3
How to Use TurboIMAGE . . . . .	1-6
How to Use This Manual . . . . .	1-8
Data Base Personnel . . . . .	1-8

## Section 2

### DATA BASE STRUCTURE AND PROTECTION

Data Elements . . . . .	2-1
Data Items . . . . .	2-1
Compound Data Items . . . . .	2-2
Data Types . . . . .	2-2
Data Entries . . . . .	2-2
Data Sets . . . . .	2-2
Data Set Types and Relations . . . . .	2-2
Master Data Sets . . . . .	2-3
Detail Data Sets . . . . .	2-4
Paths . . . . .	2-4
Automatic and Manual Masters . . . . .	2-6
Manual vs. Automatic Data Sets . . . . .	2-7
Primary Paths . . . . .	2-7
Sort Items . . . . .	2-7
The ORDERS Data Base . . . . .	2-9
Data Base Files . . . . .	2-11
Root File . . . . .	2-11
Data Files . . . . .	2-11
Record Size . . . . .	2-11
Blocks . . . . .	2-11
Protection of the Data Base . . . . .	2-12
Privileged File Protection . . . . .	2-12
Account and Group Protection . . . . .	2-12
User Classes and Passwords . . . . .	2-12
Read and Write Class Lists . . . . .	2-13
Access Modes and Data Set Write Lists . . . . .	2-14
Granting a User Class Access . . . . .	2-14
User Classes and Locking . . . . .	2-19
Protection in Relation to Library Procedures . . . . .	2-19
Protection Provided by the TurboIMAGE Utilities . . . . .	2-19



# CONTENTS (continued)

## Section 3

### DEFINING A DATA BASE

Data Base Description Language. . . . .	3-1
Language Conventions . . . . .	3-2
Schema Structure . . . . .	3-2
PASSWORD PART . . . . .	3-3
ITEM PART . . . . .	3-4
Data Item Length. . . . .	3-5
TurboIMAGE and Program Language Data Types . . . . .	3-6
Data Items of Type P . . . . .	3-8
Complex Numbers . . . . .	3-8
QUERY and Data Types. . . . .	3-8
Data Item Identifiers . . . . .	3-9
SET PART (Master) . . . . .	3-10
SET PART (Detail). . . . .	3-12
Master and Detail Search Items. . . . .	3-13
Data Set Identifiers. . . . .	3-13
Schema Processor Operation. . . . .	3-14
Creating the Textfile . . . . .	3-15
The Data Base Creator . . . . .	3-16
Schema Processor Commands . . . . .	3-17
Continuation Records. . . . .	3-17
\$PAGE. . . . .	3-18
\$TITLE . . . . .	3-19
\$CONTROL . . . . .	3-20
Selecting the Block Size. . . . .	3-21
Schema Processor Output . . . . .	3-22
Summary Information . . . . .	3-22
Schema Errors . . . . .	3-24
Schema Processor Example . . . . .	3-24

## Section 4

### USING THE DATA BASE

Opening the Data Base. . . . .	4-1
Data Base Control Blocks. . . . .	4-1
Passwords . . . . .	4-2
Access Modes . . . . .	4-2
Concurrent Access Modes . . . . .	4-3
Data Base Operations . . . . .	4-4
Selecting an Access Mode . . . . .	4-5
Dynamic Locking . . . . .	4-6
Transaction Logging . . . . .	4-7
Entering Data in the Data Base . . . . .	4-7
Sequence for Adding Entries . . . . .	4-7

# CONTENTS (continued)

Access Mode and User Class Number . . . . .	4-8
Search Items . . . . .	4-8
Reading the Data . . . . .	4-9
Current Path . . . . .	4-9
Reading Methods . . . . .	4-9
Directed Access . . . . .	4-11
Locking . . . . .	4-11
Serial Access . . . . .	4-11
Locking . . . . .	4-12
Calculated Access . . . . .	4-12
Chained Access . . . . .	4-12
Locking . . . . .	4-13
Re-Reading the Current Record . . . . .	4-13
Updating Data . . . . .	4-14
Access Modes and User Class Number . . . . .	4-14
Updating Search and Sort Items . . . . .	4-15
Deleting Data Entries . . . . .	4-15
Access Modes and User Class Numbers . . . . .	4-16
Using the Locking Facility . . . . .	4-16
Lock Descriptors . . . . .	4-17
How Locking Works . . . . .	4-18
Conditional and Unconditional Locking . . . . .	4-18
Access Modes and Locking . . . . .	4-19
Automatic Masters . . . . .	4-19
Locking Levels . . . . .	4-19
Deciding on a Locking Strategy . . . . .	4-20
Choosing a Locking Level . . . . .	4-20
Locking at the Same Level . . . . .	4-20
Length of Transactions . . . . .	4-21
Locking During User Dialog . . . . .	4-21
Choosing an Item for Locking . . . . .	4-21
Examples of Using the Locking Facility . . . . .	4-22
Issuing Multiple Calls to DBLOCK . . . . .	4-24
Releasing Locks . . . . .	4-24
Using the Logging Facility . . . . .	4-25
What Logging Does . . . . .	4-25
How Logging Works . . . . .	4-25
Logging and Logical Transactions . . . . .	4-26
Transaction Numbers . . . . .	4-26
Logging and Process Suspension . . . . .	4-26
Obtaining Data Base Structure Information . . . . .	4-27
Special Uses of DBINFO . . . . .	4-28
Checking Subsystem Flag . . . . .	4-28
Closing the Data Base or a Data Set . . . . .	4-28
Checking the Status of a Procedure . . . . .	4-29
Interpreting Errors . . . . .	4-30
Abnormal Termination . . . . .	4-30

# CONTENTS (continued)

## Section 5

### TurboIMAGE LIBRARY PROCEDURES

Using TurboIMAGE Library Procedures. . . . .	5-1
Intrinsic Numbers. . . . .	5-3
Data Base Protection . . . . .	5-3
Unused Parameters . . . . .	5-3
The Status Array . . . . .	5-3
DBBEGIN . . . . .	5-4
DBCLOSE . . . . .	5-6
DBCONTROL . . . . .	5-9
DBDELETE . . . . .	5-11
DBEND . . . . .	5-14
DBERROR. . . . .	5-16
DBEXPLAIN. . . . .	5-25
DBFIND . . . . .	5-28
DBGET . . . . .	5-30
DBINFO . . . . .	5-34
DBLOCK. . . . .	5-40
DBMEMO . . . . .	5-48
DBOPEN. . . . .	5-50
DBPUT. . . . .	5-56
DBUNLOCK. . . . .	5-61
DBUPDATE . . . . .	5-63

## Section 6

### HOST LANGUAGE ACCESS

COBOL . . . . .	6-2
Open Data Base . . . . .	6-2
Add Entry . . . . .	6-3
Read Entry (Serially) . . . . .	6-4
Read Entry (Directly). . . . .	6-5
Read Entry (Calculated) . . . . .	6-6
Read Entry (Backward Chain) . . . . .	6-7
Update Entry . . . . .	6-9
Delete Entry . . . . .	6-10
Lock and Unlock (Data Base) . . . . .	6-11

# CONTENTS (continued)

Request Data Item Information. . . . .	6-12
Rewind Data Set . . . . .	6-12
Close Data Base. . . . .	6-13
Print Error . . . . .	6-13
Move Error to Buffer. . . . .	6-13
Sample Cobol Program . . . . .	6-14
<b>FORTTRAN. . . . .</b>	<b>6-20</b>
Open Data Base. . . . .	6-20
Add Entry . . . . .	6-21
Read Entry (Serially) . . . . .	6-22
Read Entry (Directly). . . . .	6-24
Read Entry (Calculated) . . . . .	6-25
Read Entry (Forward Chain) . . . . .	6-26
Update Entry . . . . .	6-28
Delete Entry . . . . .	6-29
Lock and Unlock (Data Base). . . . .	6-30
Lock (Data Entries). . . . .	6-31
Request Data Set Information . . . . .	6-32
Rewind Data Set . . . . .	6-33
Close Data Base. . . . .	6-33
Print Error . . . . .	6-34
Move Error to Buffer. . . . .	6-34
<b>PASCAL. . . . .</b>	<b>6-35</b>
Open Data Base. . . . .	6-38
Add Entry . . . . .	6-39
Read Entry (Serially) . . . . .	6-40
Read Entry (Directly). . . . .	6-41
Read Entry (Calculated) . . . . .	6-42
Read Entry (Backward Chain) . . . . .	6-43
Locate and Update Entry. . . . .	6-44
Delete Entry . . . . .	6-45
Lock and Unlock (Data Base). . . . .	6-45
Request Data Item Information. . . . .	6-46
Rewind Data Set . . . . .	6-46
Print Error . . . . .	6-47
Move Error to Buffer. . . . .	6-47
Close Data Base. . . . .	6-47
<b>SPL . . . . .</b>	<b>6-48</b>



# CONTENTS (continued)

BASIC . . . . .	6-61
String Variables . . . . .	6-66
Type-Integer Expressions as Parameters . . . . .	6-66
Doubleword Integer Parameters . . . . .	6-66
Readlist, Writelist, Descriplist Parameters . . . . .	6-66
The Status Parameter . . . . .	6-67
Open Data Base . . . . .	6-68
Add Entry . . . . .	6-69
Read Entry (Serially) . . . . .	6-70
Read Entry (Calculated) . . . . .	6-71
Read Entry (Backward Chain) . . . . .	6-72
Update Entry . . . . .	6-73
Delete Entry (with Locking and Unlocking) . . . . .	6-74
Request Data Set Information . . . . .	6-76
Rewind Data Set . . . . .	6-77
Close Data Base . . . . .	6-78
Print Error . . . . .	6-78
Move Error to Buffer . . . . .	6-79
RPG . . . . .	6-80
RPG Programs and TurboIMAGE . . . . .	6-80

## Section 7

### MAINTAINING THE DATA BASE

Restructuring the Data Base . . . . .	7-2
Allowed Structural Changes . . . . .	7-2
Conditional or Unsupported Structural Changes . . . . .	7-3
Making a Data Base Backup Copy . . . . .	7-4
Data Base Recovery Options . . . . .	7-5
Intrinsic Level Recovery . . . . .	7-6
Using ILR . . . . .	7-7
Special Considerations . . . . .	7-7
Logical Transactions and Locking . . . . .	7-8
Locking Requirements . . . . .	7-9
Program Abort and Recovery Considerations . . . . .	7-12
Recovery Tables . . . . .	7-15
Logging Installation . . . . .	7-17
1. Acquiring Logging Capability . . . . .	7-17
2. Acquiring Log Identifier . . . . .	7-18
3. Setting Log Identifier and Flags . . . . .	7-19
4. Building a Logfile for Logging to Disc . . . . .	7-20

# CONTENTS (continued)

Displaying Logging Status . . . . .	7-21
Maintaining Logging . . . . .	7-22
Starting the Logging Process . . . . .	7-22
Changelog Capability . . . . .	7-23
Setting Data Base Enable/Disable Flags . . . . .	7-25
Ending the Logging Maintenance Cycle . . . . .	7-25
Notes on Logging . . . . .	7-26
Roll-Back Recovery . . . . .	7-27
Intrinsic Level Recovery (ILR) Requirements. . . . .	7-28
Enabling the Roll-Back Feature . . . . .	7-28
Disabling the Roll-Back Feature . . . . .	7-29
Performing Roll-Back . . . . .	7-29
Run Command . . . . .	7-30
Other DBRECOV Commands . . . . .	7-30
Control Command . . . . .	7-31
File Command . . . . .	7-32
Print Command . . . . .	7-32
Roll-Forward Recovery . . . . .	7-33
Intrinsic Level Recovery (ILR) Requirements. . . . .	7-34
Enabling the Roll-Forward Feature . . . . .	7-34
Restoring the Backup Data Base Copy . . . . .	7-35
Recovering Data Without a Backup Copy . . . . .	7-36
Performing Roll-Forward . . . . .	7-36
Post-Recovery Procedures . . . . .	7-37
Recover Command . . . . .	7-38
Run Command . . . . .	7-38
Other DBRECOV Commands . . . . .	7-39
Control Command . . . . .	7-39
File Command . . . . .	7-40
Print Command . . . . .	7-40
The Mirror Data Base . . . . .	7-41
DBRECOV STOP-RESTART Feature . . . . .	7-42
Notes on Logging . . . . .	7-43
Transferring Log Files . . . . .	7-44
Performing DBRECOV STOP-RESTART . . . . .	7-46
Stopping DBRECOV . . . . .	7-46
Storing the Data Bases . . . . .	7-47
Restarting DBRECOV . . . . .	7-48
Aborting DBRECOV . . . . .	7-50
Purging a RESTART File . . . . .	7-53

# CONTENTS (continued)

## Section 8

### USING THE DATA BASE UTILITIES

Utility Program Operation. . . . .	8-3
Backup Files . . . . .	8-3
Error Messages . . . . .	8-3
DBLOAD . . . . .	8-4
Operation . . . . .	8-4
Console Messages. . . . .	8-6
Using Control Y . . . . .	8-6
DBRECOV. . . . .	8-8
Operation . . . . .	8-8
CONTROL . . . . .	8-11
Record Numbers and Table Overflow . . . . .	8-14
EXIT . . . . .	8-15
FILE . . . . .	8-16
PRINT . . . . .	8-19
RECOVER . . . . .	8-20
CONTROL . . . . .	8-22
RUN . . . . .	8-24
DBRESTOR . . . . .	8-26
Operation . . . . .	8-26
Operation Discussion . . . . .	8-26
Console Messages. . . . .	8-27
DBSTORE . . . . .	8-28
Operation . . . . .	8-28
Operation Discussion . . . . .	8-29
Logging . . . . .	8-29
Console Messages. . . . .	8-30
DBUNLOAD. . . . .	8-31
Operation . . . . .	8-31
Broken Chains . . . . .	8-32
Operation Discussion . . . . .	8-32
Console Messages. . . . .	8-34
Using Control Y . . . . .	8-34
Writing Errors . . . . .	8-34
DBUTIL . . . . .	8-37
Operation . . . . .	8-37
Operation Discussion . . . . .	8-37
ACTIVATE . . . . .	8-38
Unexpected Results . . . . .	8-38
CREATE . . . . .	8-40
DEACTIVATE . . . . .	8-42
DISABLE . . . . .	8-43
ENABLE . . . . .	8-45

# CONTENTS (continued)

ERASE . . . . .	8-47
EXIT . . . . .	8-48
HELP . . . . .	8-49
MOVE . . . . .	8-50
PURGE . . . . .	8-52
Unexpected Results . . . . .	8-53
RELEASE. . . . .	8-54
SECURE . . . . .	8-55
SET . . . . .	8-56
SHOW. . . . .	8-59
Format of Show Device List . . . . .	8-62
Format of Show Locks List . . . . .	8-62
VERIFY. . . . .	8-65

## Section 9

### USING A REMOTE DATA BASE

Access Through a Local Application Program . . . . .	9-2
Method 1 . . . . .	9-2
Method 2 . . . . .	9-3
Method 3 . . . . .	9-4
Filename. . . . .	9-8
User Identification. . . . .	9-8
Activating a Data-Base-Access File. . . . .	9-10
Deactivating a Data-Base-Access File. . . . .	9-11
Referencing the Data Base. . . . .	9-11
Using QUERY . . . . .	9-13

## Section 10

### INTERNAL STRUCTURES AND TECHNIQUES

Data Set Structural Elements . . . . .	10-1
Pointers. . . . .	10-1
Data Chains. . . . .	10-1
Media Records . . . . .	10-1
Media Records of Detail Data Sets . . . . .	10-2
Chain Heads . . . . .	10-2
Primary Entries. . . . .	10-2
Secondary Entries. . . . .	10-2
Synonym Chains . . . . .	10-2
Media Records of Master Data Sets. . . . .	10-3
Blocks and Bit Maps . . . . .	10-4



# CONTENTS (continued)

Run-Time TurboIMAGE Control Blocks . . . . .	10-5
Local Data Base Access . . . . .	10-5
Remote Data Base Access . . . . .	10-6
Control Block Sizes . . . . .	10-7
Internal Techniques . . . . .	10-8
Primary Address Calculation . . . . .	10-8
Migrating Secondaries . . . . .	10-9
Space Allocation for Master Data Sets . . . . .	10-9
Space Allocation for Detail Data Sets . . . . .	10-9
Buffer Management . . . . .	10-10
Locking Internals . . . . .	10-10

## Appendix A ERROR MESSAGES

Schema Processor Messages . . . . .	A-1
Library Procedure Error Messages . . . . .	A-12
Abort Conditions . . . . .	A-13
Utility Error Messages . . . . .	A-32

## Appendix B RESULTS OF MULTIPLE ACCESS

Results of Multiple Access . . . . .	B-1
--------------------------------------	-----

## Appendix C SUMMARY OF DESIGN CONSIDERATIONS

Summary of Design Considerations . . . . .	C-1
--	-----

## Appendix D MULTIPLE RIN SPECIAL CAPABILITY

Sort Sequence for Lock Descriptors . . . . .	D-2
Conditional Locks . . . . .	D-2
Remote Data Bases . . . . .	D-3

# CONTENTS (continued)

## Appendix E TurboIMAGE LOG RECORD FORMATS

TurboIMAGE Log Record Formats . . . . .	E-1
---	-----

## Appendix F MPE LOG RECORD FORMATS

MPE Log Record Formats . . . . .	F-1
----------------------------------	-----

## Appendix G RECOVERY AND LOGGING QUICK REFERENCE

Recovery Quick Reference. . . . .	G-1
Intrinsic Level Recovery . . . . .	G-1
Roll-Back Recovery . . . . .	G-2
Roll-Forward Recovery. . . . .	G-3
Recovery . . . . .	G-3
Logging Device Quick Reference . . . . .	G-4
Sample Job Streams . . . . .	G-5

## Appendix H TurboIMAGE CONVERSION (DBCONV)

Preconversion Considerations . . . . .	H-1
Converting from IMAGE/3000 to TurboIMAGE. . . . .	H-3
Converting from TurboIMAGE to IMAGE/3000 . . . . .	H-7
Converting Using Job Streams . . . . .	H-9
Error Message. . . . .	H-11



# TABLES

Table	Page
2-1. Sample Read/Write Class Lists . . . . .	2-13
2-2. Granting Capability to User Class 11 . . . . .	2-14
2-3. Enabling a User Class to Perform a Task . . . . .	2-15
2-4. Sample Read and Write Class Lists . . . . .	2-18
3-1. Additional Conventions . . . . .	3-2
3-2. Type Designators . . . . .	3-5
3-3. TurboIMAGE Type Designators and Programming Languages . . . . .	3-7
3-4. Examples of an Item Part . . . . .	3-9
3-5. Schema Processor Files . . . . .	3-14
3-6. RUN and FILE Commands, Examples. . . . .	3-15
3-7. Data Set Summary Table Information . . . . .	3-23
4-1. Access Mode Summary . . . . .	4-3
4-2. Logged Intrinsic . . . . .	4-7
4-3. Locking in Shared-Access Environments . . . . .	4-23
5-1. TurboIMAGE Procedures . . . . .	5-2
5-2. Calling a TurboIMAGE Procedure . . . . .	5-3
5-3. DBBEGIN Condition Word Values . . . . .	5-5
5-4. DBCLOSE Modes 2 and 3 . . . . .	5-7
5-5. DBCLOSE Condition Word Values . . . . .	5-8
5-6. DBCONTROL Condition Word Values . . . . .	5-10
5-7. DBDELETE Condition Word Values . . . . .	5-13
5-8. DBEND Condition Word Values . . . . .	5-15
5-9. DBERROR Messages. . . . .	5-17
5-10. DBEXPLAIN Message Format . . . . .	5-26
5-11. DBFIND Condition Word Values . . . . .	5-29
5-12. DBGET Condition Word Values . . . . .	5-33
5-13. DBINFO <i>mode</i> and <i>qualifier</i> Values and Results. . . . .	5-35
5-14. DBINFO Condition Word Values . . . . .	5-39
5-15. Locking <i>mode</i> Options . . . . .	5-42
5-16. Lock Descriptor Fields . . . . .	5-45
5-17. DBLOCK Condition Word Values . . . . .	5-47
5-18. DBMEMO Condition Word Values. . . . .	5-49
5-19. DBOPEN Condition Word Values . . . . .	5-54
5-20. Special <i>list</i> Parameter Constructs . . . . .	5-58
5-21. DBPUT Condition Word Values . . . . .	5-59
5-22. DBUNLOCK Condition Word Values . . . . .	5-62
5-23. DBUPDATE Condition Word Values . . . . .	5-65
6-1. TurboIMAGE and Pascal Data Structures. . . . .	6-36
6-2. BIMAGE Procedure Calls . . . . .	6-62
6-3. BIMAGE Procedure Parameters. . . . .	6-64
6-4. Additional BIMAGE Condition Word Values . . . . .	6-67
8-1. TurboIMAGE Utility Programs . . . . .	8-1
10-1. Formulas for Approximating Control Block Sizes . . . . .	10-7

# TABLES (continued)

Table	Page
A-1. TurboIMAGE Schema Processor File Errors . . . . .	A-2
A-2. TurboIMAGE Schema Processor Command Errors . . . . .	A-3
A-3. TurboIMAGE Schema Syntax Errors . . . . .	A-4
A-4. TurboIMAGE Library Procedure File System and Memory Management Errors. . . . .	A-14
A-5. TurboIMAGE Library Procedure Calling Errors . . . . .	A-15
A-6. TurboIMAGE Library Procedure Exceptional Conditions . . . . .	A-26
A-7. TurboIMAGE Library Procedure Abort Condition Messages . . . . .	A-31
A-8. TurboIMAGE Utility Program Conditional Messages. . . . .	A-33
A-9. TurboIMAGE Utility Program Unconditional Messages . . . . .	A-52
A-10. TurboIMAGE Extended Utility Program Unconditional Messages . . . . .	A-57
B-1. Actions Resulting from Multiple Access of Data Bases . . . . .	B-2
C-1. Selected Prime Numbers . . . . .	C-2
H-1. DBCONV Program Conditional Messages. . . . .	H-12
H-2. DBCONV Program Unconditional Messages . . . . .	H-13

Figure	Page
1-1. TurboIMAGE Flow Diagram . . . . .	1-7
2-1. CUSTOMER Data Set Sample. . . . .	2-1
2-2. Master and Detail Data Set Relations. . . . .	2-3
2-3. Master and Detail Data Sets Example. . . . .	2-5
2-4. Adding an Entry to a Sorted Chain . . . . .	2-8
2-5. ORDERS Data Sets and Paths. . . . .	2-10
2-6. Sample Entries for ORDERS Data Sets . . . . .	2-10
2-7. Security Flow-Chart . . . . .	2-17
3-1. Data Base Definition Process . . . . .	3-1
3-2. Sample Schema Creation Session . . . . .	3-16
3-3. Schema Processor Batch Job Stream. . . . .	3-17
3-4. Data Set Summary Table . . . . .	3-22
3-5. ORDERS Data Base Schema. . . . .	3-25
4-1. Sample Data Entries from ORDERS Data Base . . . . .	4-8
4-2. Read Access Methods (DBGET Procedure) . . . . .	4-10
4-3. Lock Descriptor List. . . . .	4-18
5-1. Sample DBEXPLAIN Messages . . . . .	5-27
5-2. Qualifier Array Format . . . . .	5-44
5-3. Lock Descriptor Format. . . . .	5-44
6-1. Inventory Update Program . . . . .	6-15
6-2. Sample RECEIVE Execution . . . . .	6-19
6-3. Supplier Modification Program . . . . .	6-49
6-4. Sample SUPPLMOD Execution . . . . .	6-53
6-5. Purchase Transaction Display Program . . . . .	6-54
6-6. Sample SHOWSALE Execution . . . . .	6-60
6-7. Sales Transaction Display Program . . . . .	6-82
6-8. Sample SALES1 Execution . . . . .	6-84
7-1. Transactions and Transaction Blocks . . . . .	7-8
7-2. Suppression of Transactions Due to Inadequate Locking . . . . .	7-10
7-3. Quiet Periods and Recovery Blocks . . . . .	7-14
7-4. Transferring Log Files to a Secondary System. . . . .	7-45
9-1. Using a Remote Program . . . . .	9-1
9-2. Using Method 1 . . . . .	9-2
9-3. Using Method 2 . . . . .	9-3
9-4. Using Method 3 . . . . .	9-4
9-5. Preparing a Data-Base-Access File . . . . .	9-12
9-6. Using a Data-Base-Access File . . . . .	9-13
10-1. Media Record for Detail Entry . . . . .	10-2
10-2. Media Record for Primary Entry . . . . .	10-3
10-3. Media Record for Secondary Entry . . . . .	10-3
10-4. Block with Blocking Factor of Four . . . . .	10-4
G-1. Sample Job Stream for Starting Logging Cycle . . . . .	G-5
G-2. Sample Job Stream for Roll-Forward Recovery . . . . .	G-6
G-3. Sample Job Stream for Roll-Back Recovery . . . . .	G-7



## **TurboIMAGE**

This manual is a reference document for anyone involved in designing and maintaining a data base and for application programmers writing data base access programs. This manual covers data base concepts and design implementation useful to the first time IMAGE user. Previous IMAGE users will find familiar information and instructions on how to convert current IMAGE/3000 data bases to TurboIMAGE.

TurboIMAGE provides improved data base performance, better recovery processes, and increased growth capability. New recovery processes and enhanced MPE user logging aid in providing a data base that is highly accessible and both logically and structurally consistent. To increase input/output rate and performance, TurboIMAGE files may be specified to reside on different discs. The internal structure of the control blocks has been altered to minimize application process time and improve concurrency and performance. In addition, the lock area has been increased to an 8K word limit to enable better concurrency for applications.

The structure of the root file and media records in master data sets has been changed in TurboIMAGE. In addition, limitations on the number of data sets and items has been expanded to allow you to create larger data bases. These enhancements make existing IMAGE/3000 data bases incompatible with TurboIMAGE. An easy to use migration tool is provided which allows you to convert existing data bases so you can take advantage of the new features in TurboIMAGE. This migration tool is a conversion program that must be run against all IMAGE/3000 data bases before you can access them in TurboIMAGE. There exists a temporary and potential permanent increase in disc space requirements when converting the data base to TurboIMAGE due to TurboIMAGE's expanded limit on chain entries. It is recommended that you read the "Pre-Conversion Considerations" in Appendix H for more information. It is also recommended, prior to running the conversion program, that you back up (DBSTORE) all data bases. Appendix H contains all the information necessary to run the (DBCONV) conversion program.



## TurboIMAGE Enhancements

- Ability to specify, during schema definition, the device class on which each data set will reside.
- Expanded (199) data sets per data base.
- Expanded (1023) data items per data base and (255) data items per data set.
- The limit on data entries per chain and entries per data set has been expanded to 2,147,483,647.
- The lock area has been expanded to 8192 words to increase concurrency.
- DBUTIL >>MOVE command which allows TurboIMAGE files to be moved across devices.
- A user may enable automatic deferred output for a specified data base using DBUTIL. This provides the ability to speed processing time.
- A new tracing facility in TurboIMAGE passes information on the data base to a new data base design tool, TurboIMAGE Profiler. The information passed to Profiler is used to interpret the performance of data bases and application programs. TurboIMAGE Profiler must be installed on the system in order to use this tracing facility. (Refer to the *TurboIMAGE Profiler User Guide* for information on tracing and use of Profiler.)
- Intrinsic Level Recovery provides recovery after a system crash which will redo an interrupted intrinsic. Once enabled, use of ILR is automatic and transparent to the user.
- Roll-back recovery provides rapid recovery of a data base following a "soft" system crash. Data base recovery can be done without a data base backup copy (DBSTORE).
- A new feature of DBRECOV called STOP-RESTART provides the ability to have two identical data bases on two computer systems. The primary system's data base can be constantly accessed while regular maintenance and required recovery are applied to the secondary system's data base.
- TurboIMAGE uses four different types of control blocks to improve concurrency and performance. These control blocks will provide more buffer space to reduce I/O activity.
- The DBUTIL >>SHOW and >>HELP commands have been expanded to include information on these new enhancements.

## GENERAL OVERVIEW

A data base is a collection of logically-related files containing both data and structural information. Pointers within the data base allow you to gain access to related data and to index data across files.

TurboIMAGE is a set of programs and procedures that you can use to define, create, access, and maintain a data base.

The primary benefit of the TurboIMAGE data base management system is time savings. These savings are typically provided in the following areas:

### FILE CONSOLIDATION

Most information processing systems that serve more than one application area contain duplicate data. For example, a vendor's name may appear in an Inventory File, an Accounts Payable File, and an Address Label File.

The data stored in these three files probably varies slightly from file to file, resulting not only in wasted file space but also inconsistent program output. Redundant and inconsistent information severely impedes any system's capacity to deal with large amounts of data.

File consolidation into a data base eliminates most data redundancy. Through the use of pointers, logically related items of information are chained together, even if they are physically separated. In the example of vendor names and addresses, only one set of data would be stored. Through the use of logical associations, the data could be used by any program needing it. Since there is only one record to retrieve, the work required for data maintenance is greatly reduced. Finally, all reports drawn from that item of information are consistent.

### PROGRAM FILE INDEPENDENCE

Conventional file structures tend to be rigid and inflexible. The nature of conventional file management systems require that the logic of application programs be intricately interwoven with file design. When it becomes necessary to alter the structure of a file, a program must be written to change the file, and programs that access the file must be changed to reflect the file change. Since change is the rule rather than the exception in data processing, a large percentage of total time and manpower is spent reprogramming.

TurboIMAGE allows the data structure to be independent of the application program. Data item relationships are independently defined. Changes in the data base structure need only be incorporated into those programs that manipulate the changed data. User programs need view only that portion of the data base description that pertains to each program's processing requirements. Since all references to the data base are resolved at execution time, only those programs affected by changes to the data base description need be changed.

### VERSATILITY

Conventional file organization techniques allow limited access to the data they contain. Most structures allow single key access with additional relational access available only through the implementation of extensive application level programming support. TurboIMAGE allows data to be accessed with multiple keys as well as through a variety of other access methods.

## **RAPID RETRIEVAL**

Conventional file organization frequently requires the use of multiple file extracts, sorts and report programs to produce meaningful output data across file boundaries. One-time information requests frequently require weeks to implement, during which time the usefulness of the requested data may have eroded considerably.

QUERY, the Hewlett-Packard data base inquiry facility, or user-written inquiry programs which use the TurboIMAGE procedures, allow instant interrogation of the data base by individuals with access to the system.

## **DATA SECURITY**

Conventional file management systems have extremely limited data security provisions. Access to computer readable data may only be denied to individuals with system access by providing physical protection for the media upon which the file is stored; for example, the use of a data vault for storage of sensitive data stored on magnetic tape or disc.

TurboIMAGE provides security at the account, file group, and data item level. The implementation of security at the item level allows sensitive data to be stored on-line under the control of TurboIMAGE, a data base manager or designer, and system manager, with minimal regard for additional security provisions. TurboIMAGE security provisions can limit even programmer or operator access to extremely sensitive information.

When implementing a new application system, TurboIMAGE can be expected to save time in the following ways:

## **PROGRAM DEVELOPMENT**

The data base structure can be defined and built without the use of special purpose application level programming. Since control of the linkage portion of the data base is under TurboIMAGE software control, the programmer need not be concerned with testing the structure and can concentrate on the functional programming task at hand. If available, QUERY can be used to build test data as well as to interrogate the results of program and system tests. This feature eliminates the requirement that file-related programs be completed before meaningful functional programs can be written. It is no longer necessary to hold up functional program testing until file building or file maintenance programs are completed. In this manner, more modules of a given system can be tested in parallel.

A specific benefit in the COBOL environment is in the area of program coding time. The programmer need only define File Division entries for those files which exist outside the control of TurboIMAGE. Typically, such files are concerned with original entry into the processing cycle (data entry files) and with report files. All data under the control of TurboIMAGE is implicitly defined in every program which accesses the data base. The programmer need not code the data division entries associated with anything except the detail data used by a given program. The time-savings generated in correct data definition the first time the program is coded, as well as in the correct description of the physical location of the data to be processed, will reap significant benefits in the program test cycle.

## PROGRAM MAINTENANCE

Throughout the life of a system, processing requirements evolve as the usefulness of the data is explored. As file organization concepts change with the needs of the application, some data restructuring can be done with little impact on existing programs. Changes to the structure of an existing data base affect only those programs that process the changed data; no other programs in the system need be recompiled to reflect the new data base structure.

The evolution of the data base is not limited by the need to balance the cost of changing an existing system against the benefits to be derived from the new structure. It is not necessary to do a "where-used" evaluation on a data item carried in multiple files to assess the impact of a data change on existing systems.

Finally, the accessibility of data need not be limited by design decisions made during initial system design. The structure of a data base can evolve with the needs of the application user. The application designer no longer has to attempt to anticipate the needs of the user across the full life of the system.

TurboIMAGE has some effects on existing applications. Although the external interfaces remain unchanged, some application programs developed for IMAGE/3000 may require modifications if they were hard-coded with any of the old (IMAGE) limitations. BASIC programs which read the chain count may need slight modifications. This is because TurboIMAGE uses a two-word chain count whereas existing BASIC does not support double integers. Any software that reads or modifies the root file (operates in privileged mode), or is hard-coded with IMAGE/3000 limits may not work on TurboIMAGE, if the TurboIMAGE limitations are used.

## SPECIAL INFORMATION NEEDS

The requirement for one-time information in a format that has never been requested before is no longer the bane of data processing users. The user with a special data requirement can get to any subset of information on the data base, frequently without the intervention of a programmer.

Volatile analytical data requirements can be filled in a minimal amount of time by the people who need the data. The time savings in programming overhead and report specification generation can be enormous.

Native Language Support (NLS/3000) on TurboIMAGE enables localized applications to prompt the user with symbols displayed in the native language of the end user. Programs can be designed to generate multinational applications on data bases. NLS enhancements can be accessed via four TurboIMAGE utilities: DBSCHEMA, DBUTIL, DBUNLOAD, and DBLOAD. For more information refer to the *NLS/3000 Reference Manual* or sections in this manual covering the utilities listed above.

In summary, effective use of TurboIMAGE can remove a large portion of the overhead associated with integrated system design from the shoulders of application analysts and programmers. It affords the opportunity to channel system design talents into functional rather than structurally-supportive design tasks.

## HOW TO USE TurboIMAGE

The following five steps provide a summarized procedure of how to use TurboIMAGE. Refer to Figure 1-1 for an illustration of each of the following steps:

1. **DESIGN OF THE DATA BASE.** A data base designer (system analyst) or team of designers determine what data is required by all the application projects that will share the data base. They determine which data should be protected from unauthorized access and how the data will be used. These design considerations and others described in Appendix C determine the data base content and structure.
2. **DESCRIPTIONS OF THE DATA BASE.** Once the design is complete, it is described using the TurboIMAGE data base definition language. This external definition is called a schema. The data base creator processes the schema using the TurboIMAGE Schema Processor which creates an internal definition of the data base called a root file. Section 3 contains the description language syntax and operating instructions for the Schema Processor.
3. **CREATION OF THE DATA BASE FILES.** DBUTIL, a TurboIMAGE utility program, builds the data base files according to requirements of the data base structure specified in the root file. The files contain no data initially.
4. **STORAGE AND RETRIEVAL OF THE DATA.** TurboIMAGE provides a set of library procedures which can be called from COBOL, FORTRAN, Pascal, SPL, TRANSACT/3000, or BASIC language application programs. TRANSACT/3000 will work with TurboIMAGE as long as the IMAGE/3000 limits are not exceeded. TRANSACT/3000 is being updated to accomodate the new limits of TurboIMAGE. The data base can also be used with RPG programs but the Report Program Generator issues the calls to TurboIMAGE procedures. The application project members can design and write programs in the programming language which best suits their needs and call the TurboIMAGE procedures to store, modify, retrieve, and delete data. These procedures rapidly locate the data, maintain pointer information, manage the allocated file space, and return status information about the activity requested. Each procedure is described in detail in Section 5 and examples of calling them from the different languages are given in Section 6.
5. **MAINTENANCE OF THE DATA BASE.** The TurboIMAGE utility programs may be used to maintain backup copies of the data base and perform other utility functions such as recovering or restructuring the data base. These programs are described in Sections 7 and 8. You may also use the TurboIMAGE procedures to write your own maintenance programs.

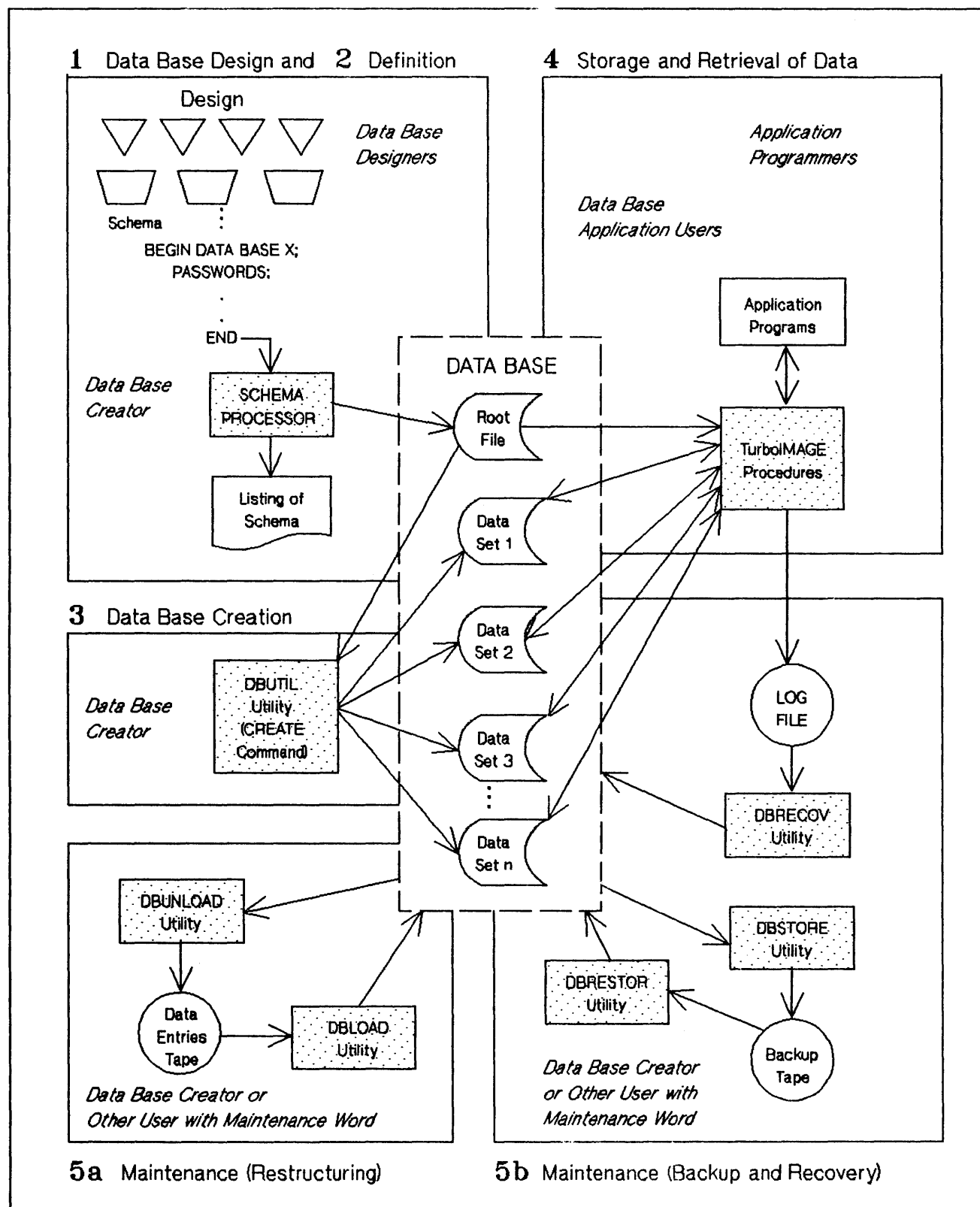


Figure 1-1. Simplified Flow Diagram, How To Use TurboIMAGE

## HOW TO USE THIS MANUAL

The information in this manual is presented in the order in which the user will begin to use the various TurboIMAGE modules. A text discussion of the overall purpose of a module and definitions of terms used to describe the module precede the reference specifications, which are identified by large headings to enable the user to locate them easily.

Each section assumes a knowledge of the material presented in preceding sections. Therefore, it is recommended that the user read the manual the first time from beginning to end, possibly skipping the discussion of topics which are already familiar.

The internal structure of TurboIMAGE elements and the methods used to perform certain functions are presented in the last section, Section 10. This section can be referenced at any time if you want to know exactly how something is accomplished by TurboIMAGE, but it is not necessary to understand the material in this section to use TurboIMAGE.

If the system has Distributed System (DS/3000) capability, refer to Section 9 for information about accessing a data base residing on another HP 3000 system.

Appendix A contains a description of the error messages issued by the various TurboIMAGE modules and Appendix B provides additional information about sharing the data base. A summary of important considerations when designing the data base is provided in Appendix C. Appendix D contains information about the special multiple RIN capability relevant to locking. Appendices E and F contain TurboIMAGE and MPE log record formats to aid in interpreting log and user recovery files. Appendix G represents a quick reference guide to recovery and logging processes. Information on converting IMAGE/3000 data bases to TurboIMAGE is given in Appendix H.

The conventions used in this manual are described on page ix.

## DATA BASE PERSONNEL

The terms data base administrator or manager, data base creator, and data base designer may refer to one or more persons. Designer refers to anyone who cooperates in the design of the data base. The creator is defined by the MPE user name, account, and group used when executing the Schema Processor to create the root file and when executing the DBUTIL program to create the data base files. The data base administrator is responsible for coordinating data base use. This person knows the passwords and can authorize others to use the data base by making a password available if it is needed for a particular application. The data base administrator is also responsible for system backup and recovery. The data base creator and administrator may be the same person. If not, the administrator will probably have access to the user name and account in which the data base resides or to the maintenance word which is defined in Section 7.

# DATA BASE STRUCTURE AND PROTECTION

SECTION

2

This section describes the various data elements and their relationships within the data base.

## DATA ELEMENTS

A data base is a named collection of related data. It is defined in terms of data items and data sets. Figure 2-1 shows a sample of one data set from a data base named **ORDERS** which will be used as an example throughout this manual. The data set is named **CUSTOMER**. The information in this data set pertains to the customers of a business. All the data about a particular customer is contained in a *data entry*. Each piece of information such as account number or last name is a *data item*.

### Data Items

A data item is the smallest accessible data element in a data base. Each data item consists of a value referenced by a data item name, typically selected to describe the data value. In general, many data item values are referenced by the same data item name, each value existing in a different data entry.

For example, in Figure 2-1, the data item **FIRST-NAME** has the value **JAMES** in one data entry and **ABIGAIL** in another data entry.

The diagram shows a table with 10 columns and 5 rows. The columns are labeled: ACCOUNT, LAST-NAME, FIRST-NAME, INITIAL, STREET-ADDRESS, CITY, CREDIT-RATING, STATE, ZIP, and CREDIT-RATING. The first row contains the values: 12345678, MILLER, JAMES, L, 1645 MARSHALL AVENUE, GLENDALE, AZ, 85301, 3.4. The second row contains: 95430301, BRIGHTON, ABIGAIL, S, 72 E. HAMPTON DRIVE, CARMEL, CA, 93921, 6.7. The third row contains: 54777833, GRAZIANO, ISABEL, M, 113 SHASTA LANE, SANTA CLARA, CA, 95050, 5.8. The fourth row is empty. The fifth row is empty. Annotations include: 'Data Item Names' with arrows pointing to the column headers; 'Data Item Value' with an arrow pointing to the value '12345678' in the first row; and 'Data Entries' with an arrow pointing to the first row.

ACCOUNT	LAST-NAME	FIRST-NAME	INITIAL	STREET-ADDRESS	CITY	CREDIT-RATING	STATE	ZIP	CREDIT-RATING
12345678	MILLER	JAMES	L	1645 MARSHALL AVENUE	GLENDALE	AZ	85301	3.4	
95430301	BRIGHTON	ABIGAIL	S	72 E. HAMPTON DRIVE	CARMEL	CA	93921	6.7	
54777833	GRAZIANO	ISABEL	M	113 SHASTA LANE	SANTA CLARA	CA	95050	5.8	

Figure 2-1. CUSTOMER Data Set Sample



## COMPOUND DATA ITEMS

A compound data item is a named group of identically defined, adjacent items within the same data entry. Each occurrence of the data item is called a sub-item and each sub-item may have a value. A compound item is similar to an array in programming languages such as FORTRAN and BASIC. A data entry might contain a compound item named MONTHLY-SALES with 12 sub-items in which the total sales for each month are recorded.

## DATA TYPES

The data base designer defines each data item as a particular type depending on what kind of information is to be stored in the item. It may be one of several types of integers, real or floating-point numbers, or ASCII character information. The data types are described in detail in the next section and summarized in Tables 3-2 and 3-3.

## Data Entries

A data entry is an ordered set of related data items. You specify the order of data items in an entry when you define the data base. Data entries may be defined with at most 255 data item names, none of which is repeated. The length of the data entry is the combined length of the data items it contains.

## Data Sets

A data base may contain up to 199 data sets. A data set is a collection of data entries where each entry contains values for the same data items. For example, the CUSTOMER data set contains entries composed of the same nine data items: ACCOUNT, LAST-NAME, INITIAL, STREET-ADDRESS, CITY, STATE, ZIP, and CREDIT-RATING. Normally, each data set is associated with some real world entity such as orders, customers, employees, and so forth.

Each data set is referenced by a unique data set name. Each data set is stored in one disc file consisting of storage locations called records. When you describe the data base with the data base definition language, you specify the *capacity*, or number of records, of each data set. Each record is identified by a record number which can be used to retrieve the entry within it.

## DATA SET TYPES AND RELATIONS

A TurboIMAGE data set is either a *master* or a *detail* data set. Figure 2-2 illustrates the relations between and types of six data sets in the ORDERS data base. Master data sets are identified by triangles and detail data sets by trapezoids. This convention is useful when diagramming the data base design.

## Master Data Sets

Master data sets have the following characteristics:

- They are used to keep information relating to a uniquely identifiable entity; for example, information describing a customer. The CUSTOMER data set in Figure 2-3 illustrates this type of information.
- They allow for rapid retrieval of a data entry since one of the data items in the entry, called the *search item*, determines the location of the data entry. A search item may not be a compound item. In Figure 2-3, the CUSTOMER data set contains a search item named ACCOUNT. The location of each entry is determined by the value of the customer's account number.
- They can be related to detail data sets containing similar search items and thus serve as indexes to the detail data set. The ACCOUNT search item in the CUSTOMER master data set is related to the ACCOUNT search item in the SALES detail data set. The entry for a customer named Abigail Brighton with account number 95430301 serves as an index to two entries in the SALES data set which contain information about purchases she made.

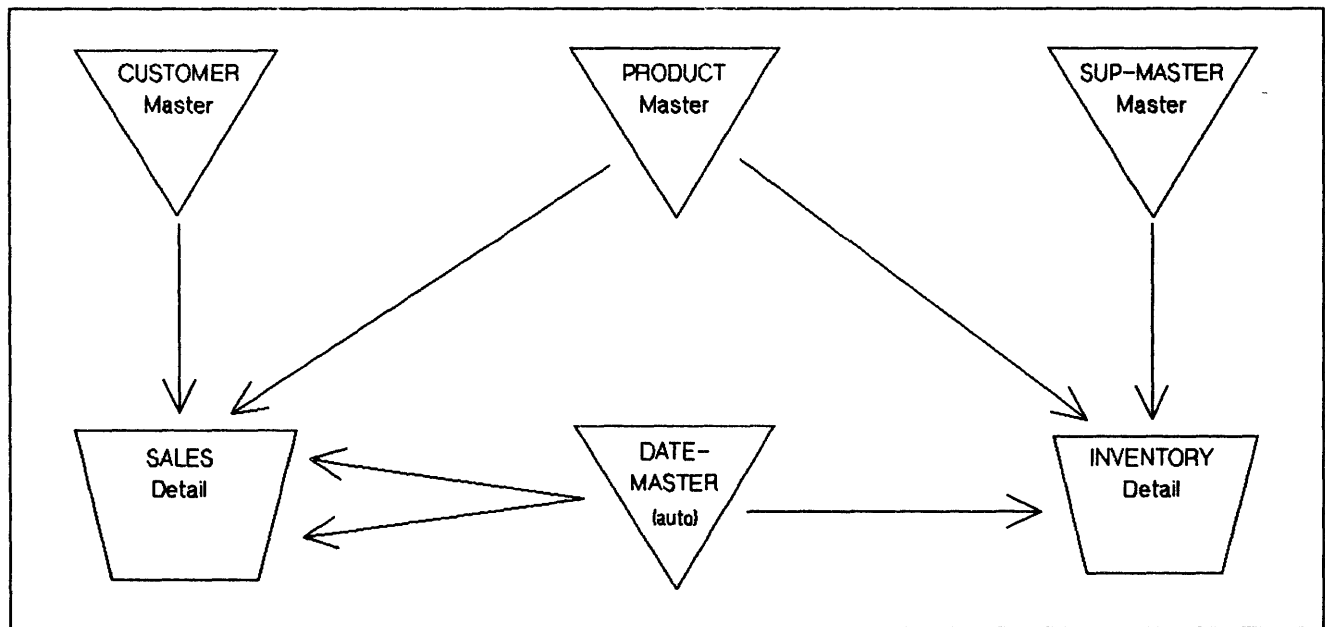


Figure 2-2. Master and Detail Data Set Relations

Although there are unused storage locations in the CUSTOMER data set, TurboIMAGE disallows any attempt to add another data entry with account number 95430301. The search item value of each entry must remain unique. The values of other data items in the master data set are not necessarily unique. This is because they are not search items and are not used to determine the location of the data entry.

## Detail Data Sets

Detail data sets have the following characteristics:

- They are used to record information about related events; for example, information about all sales to the same account.
- They allow retrieval of all entries pertaining to a uniquely identifiable entity. For example, account number 95430301 can be used to retrieve information about all sales made to Ms. Brighton.
- The storage location for a detail data set entry has no relation to its data content. When a new data entry is added to a detail data set, it is placed in the first available location.
- A detail data set may be defined with from zero to 16 search items (unlike a master data set which contains at most one search item). The values of a particular search item need not be unique. Generally, a number of entries will contain the same value for a specific search item.

The SALES data set contains four search items: ACCOUNT, STOCK#, PURCH-DATE, and DELIV-DATE. Two entries in the example in Figure 2-3 have identical values for the ACCOUNT item in the SALES data set.

TurboIMAGE stores pointer information with each detail data entry which links together all entries with the same search item value. Entries linked together in this way form a *chain*. A search item is defined for a detail data set if it is desired to retrieve together all entries with a common search item value, in other words, all entries in a chain. The SALES entries with ACCOUNT equal to 95430301 form a two-entry chain. A single chain may consist of at most 2,147,483,647 entries.

## Paths

A master data set search item can be related to a detail data set search item of the same type and size. This relationship forms a *path*. A path contains a chain for each unique search item value. In Figure 2-3, the ACCOUNT search item in CUSTOMER and the ACCOUNT search item in SALES link the CUSTOMER master to the SALES detail forming a path. One chain links all SALES entries for account number 95430301. The chain for account number 12345678 consists of one entry. Both chains belong to the same path.

Since a detail data set can contain as many as 16 search items, it can be related to at most 16 master data sets. Note that each master to detail relationship must be relative to a different detail search item. The SALES data set is related to the CUSTOMER, PRODUCT, and DATE-MASTER data sets.

A detail data set may be multiply indexed by a master data set. For example, SALES is indexed twice by DATE-MASTER. The DATE search item forms one path with the PURCH-DATE search item and one path with the DELIV-DATE search item.

Each master data set may serve as an index, singly or multiply, to one or more detail data sets. No master data set may be involved in more than 16 such relationships. For each such relationship, TurboIMAGE keeps independent chain information with each master entry. This information consists of pointers to the first and last entries of the chain whose search item value matches the master set entry's search item value and a count of the number of entries in the chain. This is called a *chain head*. The format of chain heads is given in Section 10. For example, the DATE-MASTER data entries each contain two sets of pointers, one for PURCH-DATE chains and one for DELIV-DATE chains. Chain heads are maintained automatically by TurboIMAGE.

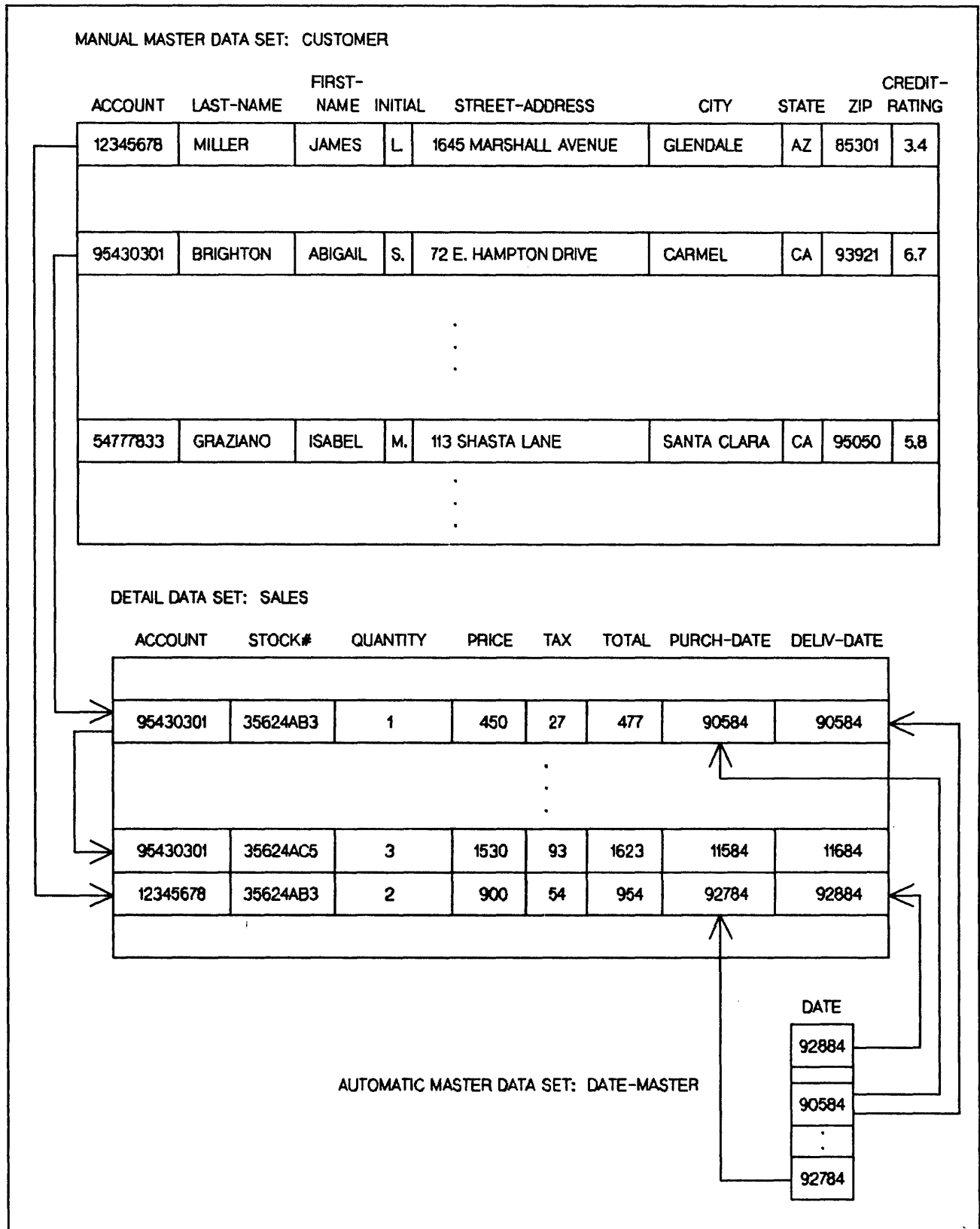


Figure 2-3. Master and Detail Data Sets Example

## Automatic and Manual Masters

A master data set may be automatic or manual. These two types of masters have the following characteristics:

MANUAL	AUTOMATIC
May be stand-alone. Need not be related to any detail data set.	Must be related to one or more detail data sets.
May contain data items in addition to the search item.	Must contain only one data item, the search item.
You must explicitly add or delete all entries. A related detail data entry cannot be added until a master entry with matching search item value has been added. When the last detail entry related to a master entry is deleted, the master entry still remains in the data set. Before a master entry can be deleted, all related detail entries must be deleted.	TurboIMAGE automatically adds or deletes entries when needed based on the addition or deletion of related detail data set entries. When a detail entry is added with a search item value different from all current search item values, a master entry with matching search item value is automatically added. Deletions of detail entries trigger an automatic deletion of the matching master entry if it is determined that all related data chains are empty.
The search item values of existing master entries serve as a table of legitimate search item values for all related detail data sets. Thus, a non stand-alone manual master can be used to prevent the entry of invalid data in the related detail data sets.	

For example, in Figure 2-3 CUSTOMER is a manual master data set and DATE-MASTER is an automatic master. Before the SALES entry for account 12345678 is added to SALES, CUSTOMER must contain an entry with the same account number. However, the DATE-MASTER entries for DATE equal to 92784 and 92884 are automatically added by TurboIMAGE when the detail entry is added to SALES, unless they are already in the DATE-MASTER data set.

Note that DATE-MASTER contains only one data item, the search item DATE, while CUSTOMER, which is a manual master, contains several data items in addition to the search item.

If the SALES entry with account number 95430301 and stock number 35624AB3 are deleted and no other SALES entry contains a PURCH-DATE or DELIV-DATE value of 90584, the DATE-MASTER entry with that value is deleted automatically by TurboIMAGE.

## Manual vs. Automatic Data Sets

Data base designers may use:

- Manual masters to ensure that valid search item values are entered for related detail entries.
- Automatic masters to save time when the search item values are unpredictable or so numerous that manual addition and deletion of master entries is undesirable.

Whenever a single data item is sufficient for a master data set, the data base designer must decide between the control of data entry available through manual masters and the time-savings offered by automatic masters. For example, since DATE-MASTER is an automatic data set, erroneous dates such as 331299 may be entered accidentally.

## Primary Paths

One of the paths of each detail data set may be designated by the data base designer as the *primary path*. The main reason for designating a path primary is to maintain the entries of each chain of the path in contiguous storage locations. You do this by occasionally using the DBUNLOAD utility program to copy the data base to tape, the DBUTIL utility program to erase the data base, and the DBLOAD program to reload the data base from the tape. When the data base is reloaded, contiguous storage locations are assigned to entries of each primary path chain. Therefore, the data base designer should designate the path most frequently accessed in chained order as the primary path. This type of access is discussed in Section 4.

A primary path also serves as the default path when accessing a detail data set if no path is specified by the calling program. This characteristic of primary paths is described with the DBGET procedure in Section 4.

## Sort Items

For any path, it is possible to designate a data item other than the search item as a *sort item*. If a sort item is specified, each of the chains of the path are maintained in ascending sorted order, based on the values of the sort item. Different paths may have different sort items, and one path's sort item may be another path's search item. Only data items of type logical or character can be designated as sort items.

For example, chains in the SALES data set composed of entries with identical ACCOUNT values are maintained in sorted order by PURCH-DATE. When information about sales to a particular customer is required, the SALES data entries for that customer's account can be retrieved in sorted order according to purchase date. (For PURCH-DATE to be a meaningful sort item, dates must be stored in a properly collatable form such as year-month-day rather than the order shown in preceding figures.)

The sorted order of entries is maintained by logical pointers rather than physical placement of entries in consecutive records. Figure 2-4 illustrates the way in which sorted paths are maintained by TurboIMAGE. When an entry is added to a detail data set it is added to or inserted in a chain. If the path does not have a sort item defined, the entry follows all existing entries in the chain. If the path has a sort item, the entry is inserted in the chain according to the value of that item.

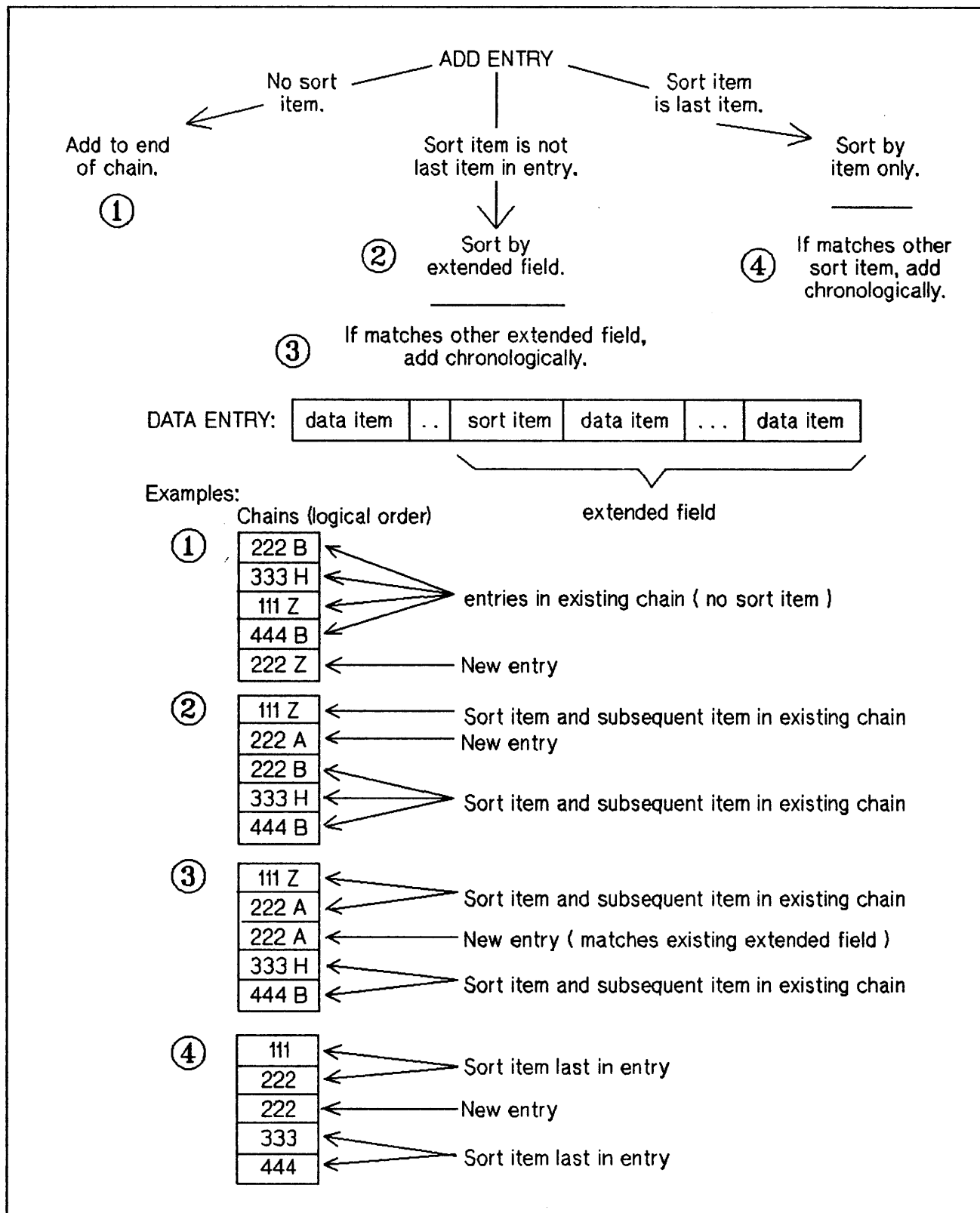


Figure 2-4. Adding an Entry to a Sorted Chain

If the entry's sort item value matches the sort item values of other entries in the chain, the position of the entry is determined by an extended sort field consisting of the sort item value and the values of all items following the sort item in the entry. If the extended sort field matches another extended sort field, the entry is inserted chronologically following the other entries with the same extended sort field value. This also occurs if the sort item is the last item in the entry and its value matches another entry's sort item value. Native Language Support does not support extended sort items. If the data base language is other than ASCII, extended sort fields are not used.

If you are using extended sort fields to sort a chain, you should not call **DBUPDATE** to modify any of the values in the extended sort fields because the chain will not be resorted automatically according to the new data values. Instead, call **DBDELETE** and **DBPUT** to re-enter the records with modified values.

If you do not want TurboIMAGE to sort chains by extended sort fields, structure the data record so that the sort item is in the last field of the record.

When the data base content is copied to magnetic tape using the TurboIMAGE utility program **DBUNLOAD**, the pointers that define an entry's position in a chain are not copied to the tape. When the data is loaded back into the data base, the chains are recreated. Therefore, entries which were previously ordered chronologically will not necessarily be in that same order. The new chronological ordering is based on the order in which the entries are read from the tape. The chains of a primary path are an exception; the order of these chains is preserved if the tape was created with **DBUNLOAD** in the chained mode. (Section 8 contains more information about **DBUNLOAD**.)

#### NOTE

It is important to limit the use of sorted chains to paths consisting of relatively short chains or chronologic sort items (for example, date) which are usually added to the end of chain. It is not intended that sorted paths be used for multiple key sorts, or for sorting entire data sets. These functions are handled more efficiently by user-written routines or the MPE subsystem, Sort/3000.

## The ORDERS Data Base

Figures 2-5 and 2-6 illustrate the complete ORDERS data base. Figure 2-5 lists the data items that define entries in each data set. The data type is in parentheses. (Data types are described in Section 3 with the item part of the schema.) Paths are indicated by arrows. **CUSTOMER**, **SUPMASTER**, **PRODUCT**, and **DATE-MASTER** are master data sets and **SALES** and **INVENTORY** are detail sets. Figure 2-6 shows a sample entry from each data set except **DATE-MASTER** for which it shows two sample entries.

Chains of the path formed by **CUSTOMER** and **SALES** are maintained in sorted order according to the value of **PURCH-DATE**. The primary path for **INVENTORY** is the one defined by **SUPMASTER** and the primary path for **SALES** is the one defined by **PRODUCT**.



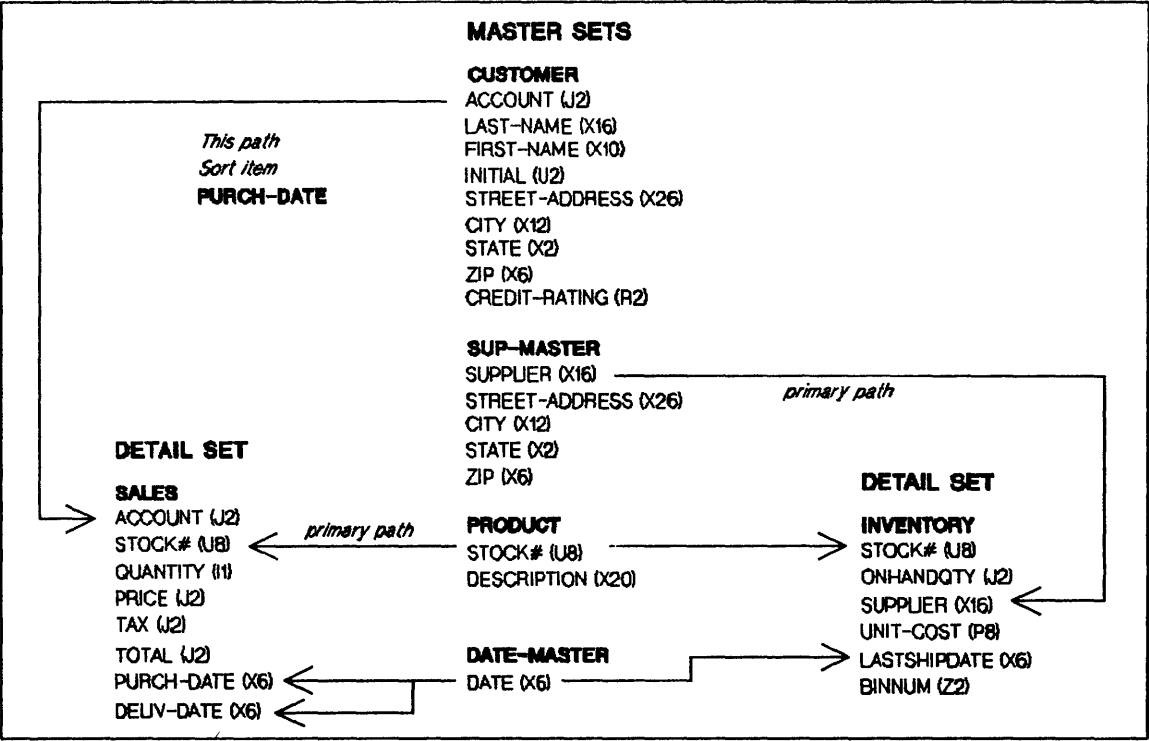


Figure 2-5. ORDERS Data Sets and Paths

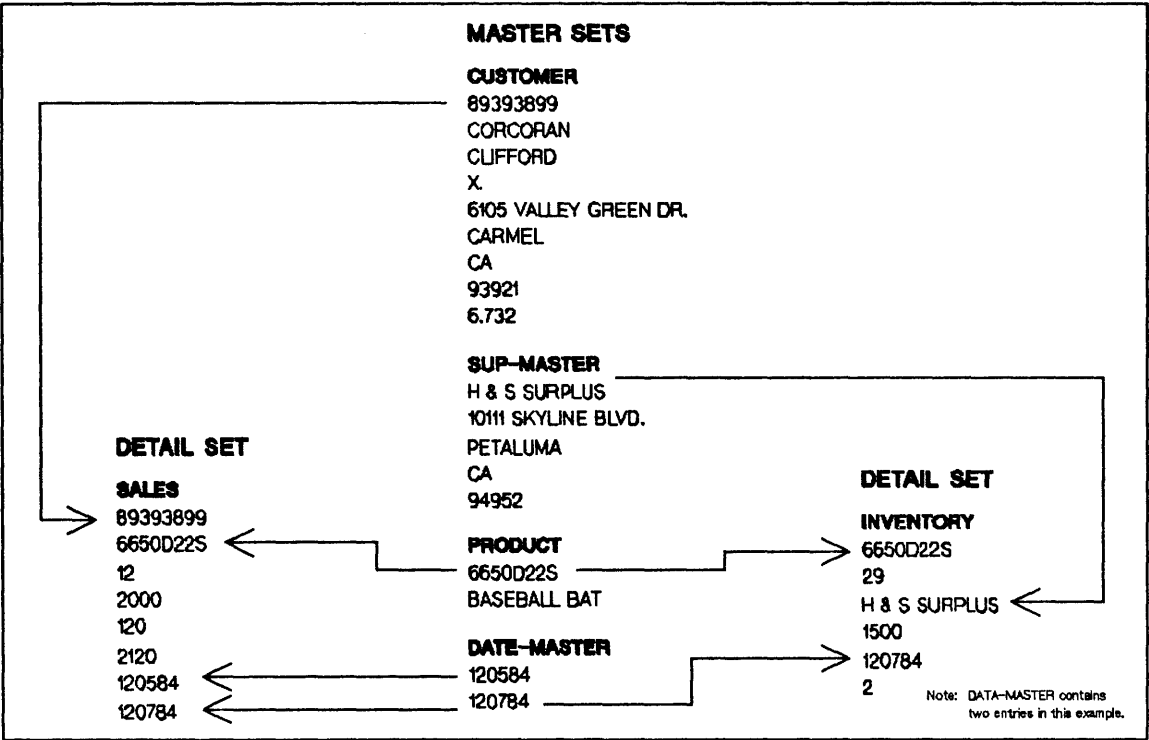


Figure 2-6. Sample Entries for ORDERS Data Sets

## DATA BASE FILES

Data base elements are stored in privileged MPE disc files. In addition to the root file which contains the data base definition, other files called *data files* contain the data sets.

### Root File

The root file is created by the data base creator when the Schema Processor is executed. It is catalogued within the creator's log-on group and account with a local file name identical to the data base name. Thus, the name of the root file for the ORDERS data base is ORDERS. Refer to the *MPE Commands Reference Manual* for more information about MPE account and log-on groups.

The root file is a single-extent MPE disc file: that is, the entire file occupies contiguous sectors on the disc. It serves as a common point of entry to, and a source of information about the data base.

### Data Files

There is one data file for each data set of a data base. The size of each record and number of records in the file are determined by the contents of the root file. The data files are created and initialized with the TurboIMAGE utility program, DBUTIL.

Each data file is catalogued within the same group and account as the root file. Local file names are created by appending two characters to the local name of the root file. These two characters are assigned to the data sets according to the order in which they are defined in the schema. For example, the ORDERS data base is defined with CUSTOMER and DATE-MASTER as the first two data sets. These data sets are in data files ORDERS01 and ORDERS02. (For more information refer to "DBUTIL >>CREATE" command in Section 8.)

Each data file is physically constructed from one to 32 extents of contiguous disc sectors, as needed to meet the capacity requirements of the file, subject to the constraints of the MPE file system. Each data file contains a user label in a disc sector maintained and used by the TurboIMAGE library procedures. The label contains structural pointers and counters needed for dynamic storage allocation and deallocation.

### RECORD SIZE

Record sizes vary between data files but are constant within each file. Each record is large enough to contain a data entry and the associated TurboIMAGE pointer information. The amount of pointer information depends on the way the data set is defined. Pointer information is described in Section 10. The maximum number of records in a data set file depends on the record size, the available disc space, and the MPE file system constraints.

### BLOCKS

The records in a data file are physically transferred to and from the disc in groups. Each group involved in a single disc transfer is called a *block*. The number of records in each block is called the *blocking factor*. The Schema Processor determines the blocking factor during creation of the root file. Section 3 contains more information about block size and blocking factors in the discussion of the set part of the schema. The format of blocks is given in Section 10.

## PROTECTION OF THE DATA BASE

TurboIMAGE prevents unauthorized persons from gaining access to the data base. It provides external protection through the MPE privileged file, account, and group structures and, in addition, provides the data base designer and data base manager with devices for further protection of the data base.

### Privileged File Protection

All TurboIMAGE data base files are privileged files. (Refer to the *MPE Intrinsic Reference Manual* for a description of the MPE privileged file capability.) Access by unprivileged processes or through most MPE file system commands is not allowed. Therefore, non-privileged users are prevented from accidentally or deliberately gaining access to the data base.

The use of MPE commands that permit copying of TurboIMAGE files to tape represent a potential breach of data base privacy, and their use should be controlled. In particular, anyone who uses the :SYSDUMP, :STORE, or :RESTORE commands should notify the data base manager. The :SYSDUMP and :STORE commands permit system supervisors, system managers and other privileged users to copy files not currently open for output to tape. The MPE RESTORE command may purge and replace a data base file with a different file if it has the same name and is encountered on tape.

### Account and Group Protection

In order to gain access to a TurboIMAGE data base, you must be able to access the files in the account and group in which the data base resides. The system manager and account manager administer the security levels for accounts and groups. The system manager is responsible for creating accounts and the account manager for creating new groups and users. (*The System Manager/System Supervisor Reference Manual* contains detailed information about the maintenance of MPE accounts and groups.)

The system and account managers can prevent members of other accounts from accessing the data base by specifying access as user type AC (Account Member) for the account and group containing the data base. They can prevent users who are members of the account, but not of the group, containing the data base from accessing it by specifying GU (Group User) for the group. On the other hand, they can allow access from other accounts by specifying user type ANY at both the account and group levels.

These MPE security provisions provide an account and group level of security controlled by the system manager and account manager.

### User Classes and Passwords

TurboIMAGE allows the data base designer to control access to specific data sets and data items by defining up to 63 *user classes* and then associating the user classes with data sets and data items in *read* or *write class lists*. This association determines which user classes may access which data elements and the type of access that is granted.

Each user class is identified by an integer from 1 to 63 and is associated with a *password* defined by the data base designer. For example, the ORDERS data base is defined with these user classes and passwords:

User Class	Password
11	CREDIT
12	BUYER
13	SHIP-REC
14	CLERK
18	DO-ALL

The magnitude of the user class number has no relation to the capability it grants. When you initiate access to the data base, you must supply a password to establish your user class. If the password is null or does not match any password defined for the data base, the user class assigned is zero. This does not apply if you are the data base creator and supply a semicolon in which case you have full access to all data sets in the data base. TurboIMAGE uses the number 64 to identify the data base creator.

### READ AND WRITE CLASS LISTS

When the data items and data sets are defined in the schema, a read class list and a write class list can be specified for each item or set. Table 2-1 contains sample lists for the CUSTOMER data set and CREDIT-RATING data item in the ORDERS data base.

Table 2-1. Sample Read/Write Class Lists

	READ CLASS LIST	WRITE CLASS LIST
CUSTOMER	11, 14	11, 18
CREDIT-RATING	14	14

User class numbers included in the write class list are, by implication, included in the read class list. Since a write class list of 14 implies that user class 14 is in the read class list, the CREDIT-RATING read class list is redundant. However, it may be included as a reminder in the schema of the total capability granted to user class 14.

A distinction must be made between the absence of a read and write class list and a null list. When you specify the lists in the schema, they are enclosed in parentheses and separated by a slash, for example, (11,14/15). A null list may be one of the following:

- (/) Both read and write class lists are null.
- (11,14/) The write class list is null.

Since the existence of a write class list implies a read class list, there is no situation where only the read class list is null.

The absence of both a read and write class list, and the parentheses and slash, yields the same result as a read class list containing all user classes and write class list which is null. For example:

(0,1,2,3,...63/)

The effect of null and absent lists is illustrated later in this section.


## Access Modes and Data Set Write Lists

Before you can gain access to a data base, you must open it specifying a password that establishes your user class number and an *access mode* that defines the type of data base tasks you want to perform. Access modes are described in Section 4 with the instructions for opening a data base. At this time it is necessary only to note that some of the eight available access modes nullify the data set write list. If the data base is opened in access mode 2, 5, 6, 7, or 8, all data set write class lists are effectively null. This effect should be considered when you are designing the security scheme for the data base.

## Granting a User Class Access

Tables 2-2 and 2-3 illustrate the use of read and write class lists from two different perspectives. Table 2-2 shows what capability user class 11 has if it appears in the lists as shown. The same rules apply to any user class. The access mode must be as indicated.

Table 2-2. Granting Capability to User Class 11

	LIST	CAPABILITY	LIST	CAPABILITY	LIST	CAPABILITY
<b>Control at Data Set Level</b>	( /11 ) or ( 11/11 )	Total access to set if access mode 1, 3, or 4	( / )	No access to set	( 11 ) or absent list	Controlled at item level
						
<b>Control at Data Item Level</b>	( /11 ) or ( 11/11 )	Update and read item	( / )	No access to item	( 11 ) or absent list	Read item

A null read and write class list can be used by the data base creator at the data set level to deny access to the data set by all user classes; that is, only the data base creator will be able to use the data set.

Table 2-3 presents the same rules organized by the task which the user class is to perform. It lists the required access modes and the security rules at both the data set and data item level. For simplicity assume there are always read and write class lists even if they are the default lists (0, 1, 2,...63 /) resulting when the lists are not actually specified in the schema (absent lists).

**Table 2-3. Enabling a User Class to Perform a Task**

<b>TASK</b>	<b>READ DATA ITEM</b>	<b>UPDATE DATA ITEM</b>	<b>ADD OR DELETE DATA ENTRIES</b>
<b>Access Modes</b>	1 - 8	1 - 4	1, 3, 4
<b>Data Set Security Rules</b>	<p>If access mode 1, 3, 4: User class in write list</p> <p>OR</p> <p>User class in read list and pass data item security.</p> <p>If access mode 2, 5-8: User class in read or write list and pass data item security.</p>	<p>If access mode 1, 3, 4: User class in write list</p> <p>OR</p> <p>User class in read list and pass data item security.</p> <p>If access mode 2: User class in read or write list and pass data item security.</p>	User class in data set write list.
<b>Data Item Security Rules</b>	User class in read or write list.	User class in write list.	

In summary, the data base designer can grant access to a data set in the following ways:

- **SPECIFY THE USER CLASS NUMBER IN THE DATA SET WRITE CLASS LIST.** If the data base is opened in access mode 1, 3, or 4, this grants the user class complete access to the data set. Users in this class can add and delete entries, update the value of any data item that is not a search or sort item, and read any item, regardless of the data item read and write class lists. A user class number must be in the data set write list in order to add and delete entries.

If the data base is opened in access mode 2, 5, 6, 7, or 8, this is the same as specifying the user class number in the data set read class list only and the next rule applies.

- **SPECIFY THE USER CLASS NUMBER IN THE DATA SET READ CLASS LIST** (or omit both lists entirely). This grants the user class a type access to the data set that is controlled at the data item level as described below. If both read and write class lists are omitted, the user class is granted this type of access since the lists are (0,1,2,...63 /) by default.
- **OMIT THE USER CLASS NUMBER FROM BOTH THE SPECIFIED READ AND WRITE CLASS LISTS.** This denies the user class any type of access to the data set.

Assuming the data base designer has established control at the data set level as summarized above, control at the data item level is established in the following ways:

- **SPECIFY THE USER CLASS NUMBER IN THE DATA ITEM READ CLASS LIST** (or omit both lists entirely). This grants the user class read access to the data item.
- **SPECIFY THE USER CLASS NUMBER IN THE DATA ITEM WRITE CLASS LIST.** This grants the user class the ability to update or change the data item value, if it is not a search or sort item. Since the user class is implied to be in the read class list, the user class can also read the item. A user class number must be in the data item write list in order to change the value.
- **OMIT THE USER CLASS NUMBER FROM BOTH THE READ AND WRITE CLASS LIST.** This denies the user class any type of access to the data item.

The protection of data set and data item values is designed so that the data base designer must explicitly specify the user class number to allow that class to make any type of change to the data base. Read access may be granted by default in some situations, for example, by omitting the lists entirely. To deny read access to a data set or data item, the data base designer must specify a list, possibly a null one, and deliberately omit the user class number.

Figure 2-7 provides a security flowchart. DBOPEN in modify access (modes 1, 3, and 4) has been passed.

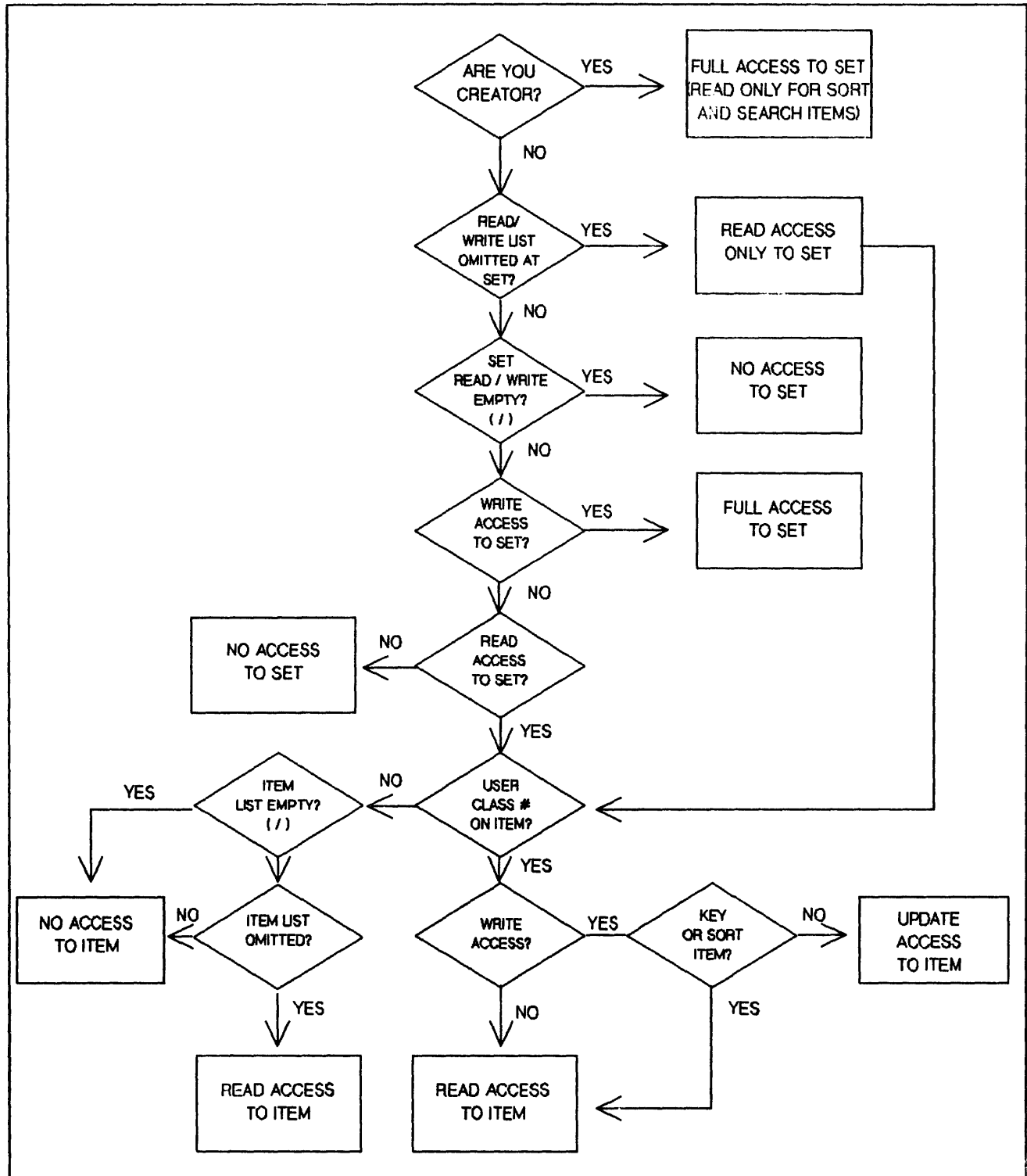


Figure 2-7. Security Flow-Chart



## Data Base Structure and Protection

For example, in the **ORDERS** data base only user classes 11 and 18 can add and delete **CUSTOMER** data entries since these are the only user class numbers in the data set write list as shown in Table 2-1. To do so, they must open the data base in access mode 1, 3, or 4.

User class 14 can update the **CREDIT-RATING** data item in the **CUSTOMER** data set because it is in the data item write list and the data set read list.

Table 2-4 contains more illustrations of the effects of read and write class lists. The data base creator and user class 9 (in access mode 1, 3, or 4) have complete access to data set 1 but only the creator has complete access to data set 2. Complete access includes the ability to read and update all items and add and delete entries.

**Table 2-4. Sample Read and Write Class Lists**

<b>Data Set 1</b>	<b>(0,18,13/9)</b>	<b>Item Read Access</b>	<b>Item Update Access</b>
Data Item A		0, 13, 18, 9	9*
Data Item B	(/13)	13, 9*	13, 9*
Data Item C	(/)	9*	9*
Data Item D	(/9)	9	9
Data Item E	(18/13)	13, 18, 9*	13, 9*
Data Item F	(/13, 18)	13, 18, 9*	13, 18, 9*
Data Item G	(12/0)	0, 9*	0, 9*
Data Item H	(13/)	13, 9*	9*
<b>Data Set 2</b>			
Data Item A		0, 1, ..., 63	
Data Item I	(13/9)	13, 9	9
*Only if DBOPEN access mode is 1, 3, or 4. None of these items are search or sort items.			

## User Classes and Locking

TurboIMAGE does not consider user classes when locking a data base entity. Any data set or any data item can be referenced in a lock request by any user of a data base regardless of his or her user class.

## Protection in Relation to Library Procedures

There is one Data Base System Control Block (DBS) for an entire system. All access to a data base is achieved through a Data Base Global Control Block (DBG), the Data Base Buffer Area Control Block (DBB), and one or more Data Base User Local Control Blocks (DBU) which reside in privileged extra data segments not directly accessible to data base users. Since no user process can read or modify these control blocks, TurboIMAGE guarantees protection of the data base from unauthorized programmatic access. Refer to the description of the DBG, DBU, DBB and DBS in Section 10. For more information about data segments and privileged mode, refer to the *MPE Intrinsic Reference Manual*.

All TurboIMAGE library procedures that structurally modify the data base execute in critical mode. This defers any requested process termination while modifications are in progress. If any file system failures occur during such data base modification, TurboIMAGE causes process termination since the data base integrity is suspect.

The DBB contains buffers which are used to transfer data. All buffers whose content has been changed to reflect a modification of the data base are always written to disc before the library procedure exits to the calling program. This guarantees data base integrity despite any program termination that might occur between successive procedure calls. However, output deferred mode allows the user to override this scheme. When output deferred or AUTODEFER mode is enabled, buffers are retained within the buffer area and are flushed only when required by space constraints or when DBCLOSE is called. For information on output deferred mode (AUTODEFER) refer to Section 8, DBUTIL >>ENABLE command.

## Protection Provided by the TurboIMAGE Utilities

The TurboIMAGE utilities perform various checks to ensure data base integrity.

- They acquire exclusive or semi-exclusive access to the data base being processed. (Section 4 contains more information about types of access in the discussion of opening a data base.)
- Only the data base creator or a user supplying the correct *maintenance word* can execute the utilities. The data base creator defines the maintenance word when the data base is created with the DBUTIL utility program. (Refer to Section 8.) In addition, anyone running the utility programs other than DBRECOV must be logged on to the group in which the data base resides.
- Unrecoverable disc or tape problems are treated as functional failures rather than limited successes and result in program termination.

# DEFINING A DATA BASE

SECTION

3

Once the data base has been designed, it must be described with the data base description language and processed by the Schema Processor to create the root file. Figure 3-1 illustrates the steps in defining the data base.

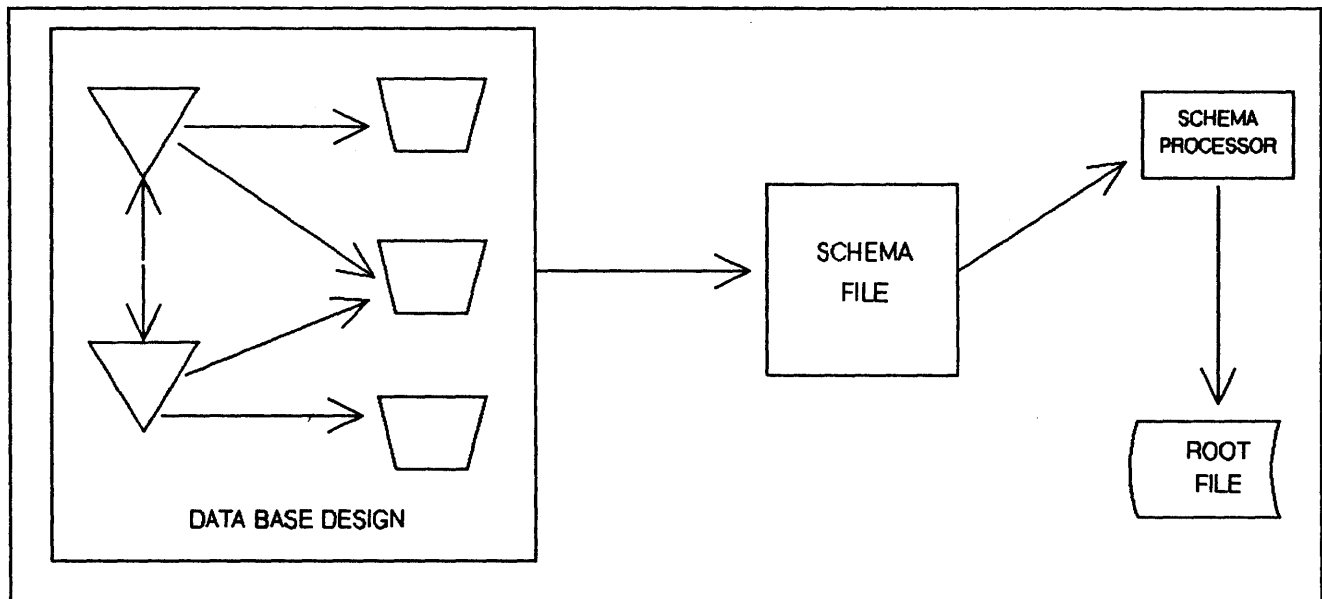


Figure 3-1. Data Base Definition Process

## DATA BASE DESCRIPTION LANGUAGE

The data base description, called a schema, may exist in the MPE system as an ASCII file on cards, magnetic tape, or as a catalogued disc file. Regardless of the actual physical record size of the file, the Schema Processor reads, prints, and processes only the first 72 characters of each record. Any remaining character positions in the record are available for your convenience, to be used for comments or collating information. The data base description language is a free-format language; you can insert blanks anywhere in the schema to improve its appearance except within symbolic names and reserved words.

## Language Conventions

The conventions used in describing the data base language are the same as those described on the conventions sheet at the beginning of this manual. In addition, the conventions in Table 3-1 apply.

**Table 3-1. Additional Conventions**

<b>PUNCTUATION</b>	All punctuation appearing in format statements must appear exactly as shown.
<b>COMMENTS</b>	Comments take the form: <<comment>>  They may contain any characters and may appear anywhere in the schema except embedded in another comment. They are included in the schema listing but are otherwise ignored by the Schema Processor program.
<b>DATA NAMES</b>	Data names may consist of from 1 to 16 alphanumeric characters, the first of which must be alphabetic. Characters after the first must be chosen from the set:  letters A - Z, digits 0 - 9, or + - * / ? ' # % & @
<b>UPSHIFTING</b>	All alphabetic input to the Schema Processor is upshifted (converted to upper case, with the exception of passwords which may contain lowercase characters).

## Schema Structure

The overall schema structure is:

```
BEGIN DATA BASE data base name [, LANGUAGE=language];
PASSWORDS: password part
ITEMS: item part
SETS: set part
END.
```

The data base name is an alphanumeric string from 1 to 6 characters. The first character must be alphabetic.

The *language* is the native language definition name or number for the data base. Refer to the *Native Language Support Reference Manual* for further information. The default language is the US ASCII character set.

The password part, item part, and set part are described on the following pages. Figure 3-5 contains a complete schema for the ORDERS data base that is used in the examples in this manual.

# PASSWORD PART

The password part defines user classes and passwords. Section 2 contains a description of user classes and how they are used to protect data elements from unauthorized access.

## Syntax

```
user class number [password];  
.  
.  
.  
user class number [password];
```

## Parameters

<i>user class number</i>	an integer between 1 and 63 inclusive. User class numbers must be unique within the password part.
<i>password</i>	from 1 to 8 ASCII characters including lower case and excluding carriage return, slash, semicolon, and blank. Blanks are removed by the Schema Processor.

## Example

```
14 CLERK  
12 BUYER  
11 CREDIT
```

## Discussion

If the same password is assigned to multiple user class numbers, the highest numbered class is used. It is not an error to omit the password, but the Schema Processor ignores lines containing only a user class number.

# ITEM PART

The item part defines data items including the data item name, length, and the user classes that have access to the item. The data set(s) in which the data item appears is defined in the set part definition.

## Syntax

```
item name, [sub-item count] type designator [sub-item length]  
[(read class list/write class list)];
```

## Parameters

<i>item name</i>	the data item name. It must be a valid TurboIMAGE data name as described in Table 3-1. It must be unique within the item part.
<i>sub-item count</i>	an integer from 1 to 255 that denotes the number of sub-items within an item. If omitted, by default it equals one. A data item whose sub-item count is 1, is a simple item. If the sub-item count is greater than one, it is a compound item.
<i>type designator</i>	defines the form in which a sub-item value is represented in the computer. The type designators I, J, K, R, U, X, Z, P are described in Table 3-2.
<i>sub-item length</i>	an integer from 1 to 255. It is the number of words, characters, or nibbles (depending on the type designator) in a sub-item. If omitted, it is equal to 1 by default.
<i>read class list</i>	a group of user class numbers between 0 and 63, inclusive, separated by commas. User class numbers are described in Section 2.
<i>write class list</i>	a group of user class numbers between 0 and 63, inclusive, separated by commas.

## Example

```
FIRST-NAME, X10 (12,14/11);
```

## Discussion

There can be no more than 1023 data items in a data base. A data item name can appear in more than one data set definition. For example, a data item named ACCOUNT appears in both the CUSTOMER and SALES data sets of the ORDERS data base.

## Data Item Length

Each data item value is allotted a storage location whose length is equal to the product of the item's sub-item length and its sub-item count. The unit of measure for the length depends upon the type designator and may be a word, byte, or nibble. A word is a 16-bit computer word, a byte is eight bits or a half-word, and a nibble is four bits or a half-byte. Table 3-2 defines the various type designators and specifies the unit of measure used for each.

**Table 3-2. Type Designators**

<b>WORD DESIGNATORS:</b>	I	A signed binary integer in 2's complement form.
	J	Same as I but QUERY allows only numbers conforming to specifications for COBOL COMPUTATIONAL data to be entered.
	K	An absolute binary quantity.
	R	A real (floating point) number.
<b>CHARACTER DESIGNATORS:</b>	U	An ASCII character string containing no lowercase alphabetic characters.
	X	An unrestricted ASCII character string.
	Z	A zoned decimal format number.
<b>NIBBLE DESIGNATOR:</b>	P	A packed decimal number.

A data item must be an integral number of words in length regardless of the type designator and its unit of measure. In other words, data items of type U, X, or Z which are measured in bytes must have a sub-item length and sub-item count such that their product is an even number. If a data item is defined as U3, it cannot be a simple (item) and must have an even numbered sub-item count so that the data item length is an integral number of words. Data items of type P which are measured in nibbles must have a sub-item length and sub-item count such that their product is evenly divisible by 4, since 4 nibbles equal 1 word.

A data item cannot exceed 2047 words in length. The entire item, whether simple or compound, is always handled as a unit by TurboIMAGE.

## ITEM PART

### TurboIMAGE and Program Language Data Types

The type designator, sub-item count, and sub-item length you specify for a data item defines its length. TurboIMAGE does not perform any conversions of data or examine the item to check its validity as it is being added to the data base. The only data item values that TurboIMAGE checks are those specified as part of a lock descriptor in calls to the DBLOCK procedure. (Refer to the discussions on locking in Section 4.) There are no rules that a specific type of data defined by a programming language must be stored in a specific type of TurboIMAGE data item.

Table 3-3 relates TurboIMAGE type designators and sub-item lengths to the data types typically used to process them in the available programming languages. Some BASIC language restrictions are noted.

Note that the UNIT-COST item in the INVENTORY data set is easier to process with COBOL or RPG programs than with the other languages since packed data is a standard data type in COBOL and RPG. However, the CREDIT-RATING data item in the CUSTOMER data set is easier to process with FORTRAN, SPL, or BASIC programs since real numbers can be arithmetically manipulated in these languages. As actual data base may be designed so that some data sets are processed by programs coded in one language and others by programs coded in another language. Another data set may be conveniently processed by programs written in any of the languages.

In order to specify a doubleword integer in BASIC, define a two-word array in which the first word contains the high-order digits of values greater than 32767, or zero, and the second word contains the low-order digits of values greater than 32767 or the entire value if it is less than 32767.



Table 3-3. TurboIMAGE Type Designators and Programming Languages

	COBOL	FORTRAN	RPG	SPL	BASIC
I	COMPUTATIONAL S9 to S9(4)	INTEGER	Binary	INTEGER	INTEGER **
I2	COMPUTATIONAL S9(5) to S9(9)	INTEGER * 4	Binary	DOUBLE INTEGER	
I4	COMPUTATIONAL S9(10) to S9(18)		Binary		
J	COMPUTATIONAL S9 to S9(4)	INTEGER	Binary	INTEGER	INTEGER **
J2	COMPUTATIONAL S9(5) to S9(9)	INTEGER * 4	Binary	DOUBLE INTEGER	
J4	COMPUTATIONAL S9(10) to S9(18)		Binary		
K1		LOGICAL		LOGICAL	***
R2 *		REAL		REAL	REAL ****
R4		DOUBLE PRECISION		LONG	LONG ****
U	DISPLAY PICTURE A	CHARACTER	Character	BYTE	String
X	DISPLAY PICTURE X	CHARACTER	Character	BYTE	String
Z	DISPLAY PICTURE 9		Character		
P	COMPUTATIONAL-3		Numeric		
<p>* Real numbers must have a length of 2 or more words; R and R1 (explicitly) are not supported by TurboIMAGE.</p> <p>** BASIC integers cannot have the value -32768</p> <p>*** Type LOGICAL items &gt; 32767 which are accessed as type INTEGER in BASIC programs are treated as negative integers.</p> <p>**** BASIC REAL and LONG data cannot have the value 10<sup>-78</sup></p>					

Refer to Section 6 for Pascal and TurboIMAGE type designators.

## ITEM PART

### Data Items of Type P

The bits used to represent the sign of a packed decimal value can vary depending on whether the value is entered using QUERY, a COBOL program, or an RPG program. Here is a summary of what happens in each case:

- For Values Entered Using QUERY:

NO Sign Specified:	Sign is 1111 <sub>2</sub>
PLUS Sign Specified:	Sign is 1100 <sub>2</sub>
MINUS Sign Specified:	Sign is 1101 <sub>2</sub>

- For Values Entered Using COBOL:

PICTURE Clause Specifies NO Sign:	Sign is 1111 <sub>2</sub>
PICTURE Clause Specifies PLUS Sign:	Sign is 1100 <sub>2</sub>
PICTURE Clause Specifies MINUS Sign:	Sign is 1101 <sub>2</sub>

- For Values Entered Using RPG:

NO Sign or PLUS Specified:	Sign is 1100 <sub>2</sub>
MINUS Sign Specified:	Sign is 1101 <sub>2</sub>

When using TurboIMAGE to locate all packed data items with a particular value (as described in a later section), you must be aware that TurboIMAGE differentiates between unsigned, positive, and negative data items with the same absolute value. For example, if you search for all data items with the value +2, TurboIMAGE will not retrieve any items with the unsigned value 2.

In general, TurboIMAGE treats any two values with different binary representations as unequal regardless of their type.

### COMPLEX NUMBERS

Applications programmed in BASIC or FORTRAN can define and manipulate complex numbers by using data type R2 with a sub-item count of 2, storing the real part in the first sub-item and the imaginary part in the second sub-item.

### QUERY AND DATA TYPES

QUERY supports only a subset of the available data item types. If you intend to use QUERY you should consult the *QUERY Reference Manual* for specific information about the way QUERY handles the various TurboIMAGE data types, including compound data items.

Table 3-4. Examples of an Item Part

ITEMS:	
A,I2;	<<32 BIT SIGNED INTEGER>>
MELVIN,3I(1,20/44);	<<COMPOUND ITEM. THREE SINGLE WORD SIGNED INTEGERS. READ CLASSES ARE 1 AND 20; WRITE CLASS IS 44.*>>
BLEVET,J;	<<SINGLE-WORD SIGNED INTEGER BETWEEN -9999 AND 9999.>>
COSTS,2X10;	<<COMPOUND ITEM. TWO 10-CHARACTER ASCII STRINGS.>>
DATE,X6;	<<SIX-CHARACTER ASCII STRING.>>
VALUES,20R2(1/8);	<<COMPOUND ITEM. 20 2-WORD REAL (FLOATING-POINT) NUMBERS. READ CLASS IS 1;WRITE CLASS IS 8.*>>
PURCHASE-MONTH,U8;	<<EIGHT-CHARACTER ASCII STRING WITH NO LOWER CASE ALPHABETICS.>>
MASK,K2;	<<32 BIT ABSOLUTE BINARY QUANTITY.>>
TEMPERATURE,17R4;	<<COMPOUND ITEM. 17 FOUR WORD REAL (FLOATING-POINT) NUMBERS.>>
SNOW*#@,Z4;	<<FOUR-DIGIT ZONED DECIMAL (NUMERIC DISPLAY) NUMBER.>>
POPULATION,P12;	<<11 DECIMAL DIGITS PLUS A SIGN IN THE LOW ORDER NIBBLE. OCCUPIES THREE WORDS.>>
	*WRITE CLASSES CAN ALSO READ.

## Data Item Identifiers

When using the TurboIMAGE procedures described in the next section, you can reference a data item by name or number. The data item number is determined by the item's position in the item part of the schema. The first item defined is item 1, the second is item 2, and so forth.

It is more flexible to use data item names since a change in the order of the item definitions or the deletion of an item definition from the schema might require changes to all application programs referencing the data items by number. Thus, to maintain program file independence, it is recommended that you use data item names if possible.

# SET PART (Master)

The set part of the schema defines data sets. It indicates which data items listed in the item part belong to which sets and links the master data sets to the detail data sets by specifying them as search items.

## Syntax

```
{NAME:} set name, {M[ANUAL]
{N:} {A[UTOMATIC]} [(read class list/write class list)]
                        [,device class];

{ENTRY:} item name [(path count)],
{E:}      .
          .
          .
          item name [(path count)];

{CAPACITY:}
{C:} maximum entry count;
```

## Parameters

<i>set name</i>	the data set name; must be a valid TurboIMAGE data name as described in Table 3-1.
MANUAL (or M)	denotes a manual master data set. Each entry within a manual master must be created manually and may contain one or more data items.
AUTOMATIC (or A)	denotes an automatic master data set. Each data entry within an automatic master is created automatically by TurboIMAGE and contains only one data item.
<i>read class list</i>	a group of user class numbers between 0 and 63, inclusive, separated by commas. User class numbers are described in Section 2.
<i>write class list</i>	a group of user class numbers between 0 and 63, inclusive, separated by commas.
<i>device class</i>	is an MPE device class name on which the data set file resides.
<i>item name</i>	the name of a data item defined in the item part.
<i>path count</i>	an integer between 0 and 16, inclusive, which is used with the search item only. It indicates the number of paths which will be established to various detail data sets. (Refer to Section 2 for more information about paths.) A path count must be specified for one, and only one, item in the master set. A zero path count may be used with a manual master data item to indicate the search item. A manual master defined in this way is not linked to any detail data set. An automatic master has one item that must have a path count greater than zero.

## SET PART (Master)

*maximum entry count* the maximum number of entries the data set can contain, the data set's capacity. It must be less than  $2^{31} - 1$  (2,147,483,647).

### Example

```
NAME:      SUP-MASTER,MANUAL(13/12,18),DISC1;
ENTRY:     SUPPLIER(1),
           STREET-ADD,
           CITY,
           STATE,
           ZIP;
CAPACITY:  200;
```

### Discussion

The example above shows the data set SUP-MASTER which will reside on Disc1. Assigning the device class where a data set will reside can provide greater performance for the TurboIMAGE data base and may aid in better utilizing system resources. An understanding of how to spread the data set files over multiple disc devices may be obtained from your system manager. Your system manager will be able to give you a listing of logical devices and their corresponding device class names (each logical device may have up to eight names).

To retrieve information on where each data set resides after specifying device classes in the schema you may run the MPE LISTDIR5 utility (after the data base is created). This utility lists the device type, logical device number, and the device class name for each data set file in the data base. DBUTIL >>SHOW may also be used to display the devices on which data set files reside.

# SET PART (Detail)

The following provides the detail set part syntax and parameters.

## Syntax

```
{NAME:} set name, D[ETAIL] [(read class list/write class list)]
{N:}                                     [,device class];

{ENTRY:} item name [(!!] master set name [(sort item name)]],
{E:}
.
.
.
item name [(!!] master set name [(sort item name)]];

{CAPACITY:}
{C:} maximum entry count;
```

## Parameters

<i>set name</i>	the data set name. It must be a valid TurboIMAGE data name as defined in Table 3-1.
DETAIL (or D)	denotes a detail data set.
<i>read class list</i>	a group of user class numbers between 0 and 63, inclusive, separated by commas. User class numbers are described in Section 2.
<i>write class list</i>	a group of user class numbers between 0 and 63, inclusive, separated by commas.
<i>device class</i>	is an MPE device class name on which the data set file resides.
<i>item name</i>	the name of a data item defined in the item part. Each item defined as a search item must be a simple item. Up to 16 items may be search items. (Refer to <i>master set name</i> for more information about search items.)
! (exclamation point)	denotes a primary path. Only one path in each detail data set can be designated as a primary path. If no path is designated as primary, the first unsorted path is the primary path by default. If all of the paths are sorted, the default primary path is the first sorted path.
<i>master set name</i>	the name of a previously defined master data set. When a master set name follows an item name, it indicates that the data item is a search item linking the detail set to the named master. Up to 16 search items can be defined for a detail data set. If no data items have a master name following them, the detail is not related to any master. In this case, the combined length of all data items in the data set must equal or exceed two words.

## SET PART (Detail)

*sort item name* the name of a detail data item of type U, K, or X which is a part of the data set being defined. A sort item defines a sorted path. Each entry added to a chain of a sorted path will be linked logically in ascending order of the sort item values. If sort item is omitted, the path order is chronological, that is, new entries are linked to the end of chains. For performance reasons, sorted chains should be kept short. (Refer to "Sort Items" in Section 2.)

*maximum entry counts* the maximum number of entries allowed in a data set (data capacity); must be less than  $2^{31} - 1$  (2,147,483,647).

### Example

```
NAME:    SALES,DETAIL(11/14,18),DISC1;
ENTRY:    ACCOUNT(CUSTOMER(PURCH-DATE)),
          STOCK#(!PRODUCT),
          QUANTITY,
          PRICE,
          TAX,
          TOTAL,
          PURCH-DATE (DATE-MASTER),
          DELIV-DATE (DATE-MASTER);
CAPACITY: 500;
```

### Master and Detail Search Items

The master and detail search items that define a path between two data sets must have identical *type designators* and *sub-item lengths* when they are defined in the item part. Since the same data item name may appear in more than one data set, you may use the same data item name and definition for both the master and detail search items. For example, the data item ACCOUNT is used as the search item in both the CUSTOMER master and SALES detail data sets.

If you want to make a distinction between the search items, however, they may be defined separately. An example of this technique is found in the ORDERS data base. The search item DATE links the DATE-MASTER data set to the SALES data set through two paths, and two search items, PURCH-DATE and DELIV-DATE. These three data items look like this in the item part:

```
DATE,      X6;
DELIV-DATE, X6 (/14);
PURCH-DATE, X6 (11/14);
```

Each data item has type designator X and sub-item length 6. The item names, read class lists, and write class lists differ however.

Figure 3-5 at the end of this section contains the listing printed by the Schema Processor when the ORDERS data base schema is processed. Refer to this figure for examples of the schema parts.

### Data Set Identifiers

Like data items, data sets may be referenced by name or number. The data set number is determined by the set's position in the set part of the schema. It is more flexible to use data set names, however, in order to maintain program file independence.

## SCHEMA PROCESSOR OPERATION

The Schema Processor is a program which accepts a *textfile* containing the schema as input, scans the schema and if no errors are detected, optionally produces a root file. The Schema Processor prints a heading, an optional list of the schema, and summary information on a *listfile*.

The Schema Processor executes in either MPE job or session mode. For further information about sessions and jobs, refer to the *MPE Commands Reference Manual*. In either case, you must use the following MPE command to initiate execution of the Schema Processor:

```
:RUN DBSCHEMA.PUB.SYS
```

Table 3-5 lists the formal file designators and default actual file designators which the Schema Processor uses for textfile and listfile. The input/output devices to which \$STDINX and \$STDLIST refer depend upon the way the system is generated. However, \$STDINX is the standard job or session input device and \$STDLIST is the standard job or session output device.

**Table 3-5. Schema Processor Files**

FILE	USE	FORMAL FILE DESIGNATOR	DEFAULT ACTUAL FILE DESIGNATOR
<b>textfile</b>	Schema and Schema Processor Commands	DBSTEXT	\$STDINX
<b>listfile</b>	output listing	DBSLIST	\$STDLIST

If you want to equate these files to some other actual file designator, you can use the MPE :FILE command. If a :FILE command is included in the job stream, you must inform the Schema Processor of this in the :RUN command in the following way:

```
:RUN DBSCHEMA.PUB.SYS;PARM=n
```

where

- n* = 1      if an actual file designator has been equated to DBSTEXT.
- n* = 2      if an actual file designator has been equated to DBSLIST.
- n* = 3      if actual file designators have been equated to both DBSTEXT and DBSLIST.



Table 3-6 shows sample combinations of :RUN and :FILE commands which can be used to initiate DBSCHEMA execution.

**Table 3-6. RUN and FILE Commands, Examples**

:RUN DBSCHEMA.PUB.SYS	Uses all default files. Prompts for lines of schema in session mode.
:FILE DBSTEXT=ORDERSSC :RUN DBSCHEMA.PUB.SYS; PARM=1	Processes schema from a user disc textfile named ORDERSSC.
:FILE DBSLIST; DEV=LP :RUN DBSCHEMA.PUB.SYS; PARM=2	Outputs the listing to a line printer.
:FILE DBSTEXT=ORDERSSC :FILE DBSLIST; DEV=LP :RUN DBSCHEMA.PUB.SYS; PARM=3	Processes schema from user textfile named ORDERSSC; outputs the listing to a line printer.

Only the first 72 characters of each textfile record are processed.

If you request a root file, and the schema is error-free, it is created, given the same name as the one specified for the data base in the schema, initialized, and saved as a catalogued disc file.

## Creating the Textfile

A convenient method for creating the input file is to use the text editor, EDIT/3000, to enter the commands and schema in a disc file. Figure 3-2 illustrates this process in a sample session which also executes the Schema Processor. (Refer to *EDIT/3000 Reference Manual* for information about the Editor.)

The steps followed in the sample in Figure 3-2 are:

- 1 Initiate an MPE session by logging on with the appropriate user name and account.
- 2 Initiate text editor execution. Enter an Editor ADD command in response to the first prompt.
- 3 Enter Schema Processor commands and the schema itself into records of the Editor work file.
- 4 Save the work file in a disc file named ORDERSSC. Then terminate the Editor.
- 5 Use the :FILE command to equate the formal file designator DBSLIST to the line printer and DBSTEXT to the disc file ORDERSSC.
- 6 Initiate execution of DBSCHEMA and indicate that the textfile and listfile have been defined in :FILE commands. When the Schema Processor has finished processing the schema it prints the number of error messages and verifies that the root file has been created.

Figure 3-3 illustrates the order of commands and other input required when executing the Schema Processor in batch mode. The job can also be stored in a disc file and executed from a terminal.

**(RETURN)**

```
1  :HELLO USER.ACCOUNT
    HP3000 / MPE V/E G.00.00.    FRI, DEC 7, 1984,  2:07 PM

2  :EDITOR

    HP32201A.7.15  EDIT/3000  FRI, DEC 7, 1984,  2:07 PM
    (C) HEWLETT-PACKARD CO.  1983
3  /ADD
    1  $PAGE "SCHEMA OF DATA BASE ORDERS"
    2  $CONTROL ERRORS=5, BLOCKMAX=256
    3  BEGIN DATA BASE ORDERS;
        .
        .
        .
    59 END.
    60 //

    ...
4  /KEEP ORDERSSC
    /EXIT

5  :FILE DBSLIST;DEV=LP
    :FILE DBSTEXT=ORDERSSC
6  :RUN DBSCHEMA.PUB.SYS;PARM=3

    HP32215C.00.00
    NUMBER OF ERROR MESSAGES: 0
    ROOT FILE ORDERS CREATED

    END OF PROGRAM
    :BYE
```

**Figure 3-2. Sample Schema Creation Session**

## **The Data Base Creator**

The person who creates the root file is identified as the data base creator and can subsequently create and initialize the data base. To do so, the data base creator must log on with the same account, user name, and group that he or she used to create the root file and execute the TurboIMAGE utility program DBUTIL. This program is described in Section 8.

## SCHEMA PROCESSOR COMMANDS

TurboIMAGE provides several commands which you may use anywhere in the schema to specify options available while processing the schema. The commands are: \$PAGE, \$TITLE, and \$CONTROL. The \$ must always be the first character of the record, immediately followed by the command name, which must be completely spelled out.

If a parameter list is included with the command, it must be separated from the command name by at least one blank. Parameters are separated from each other by commas. Blanks may be freely inserted between items in the parameter list.

Command records may not contain comments.

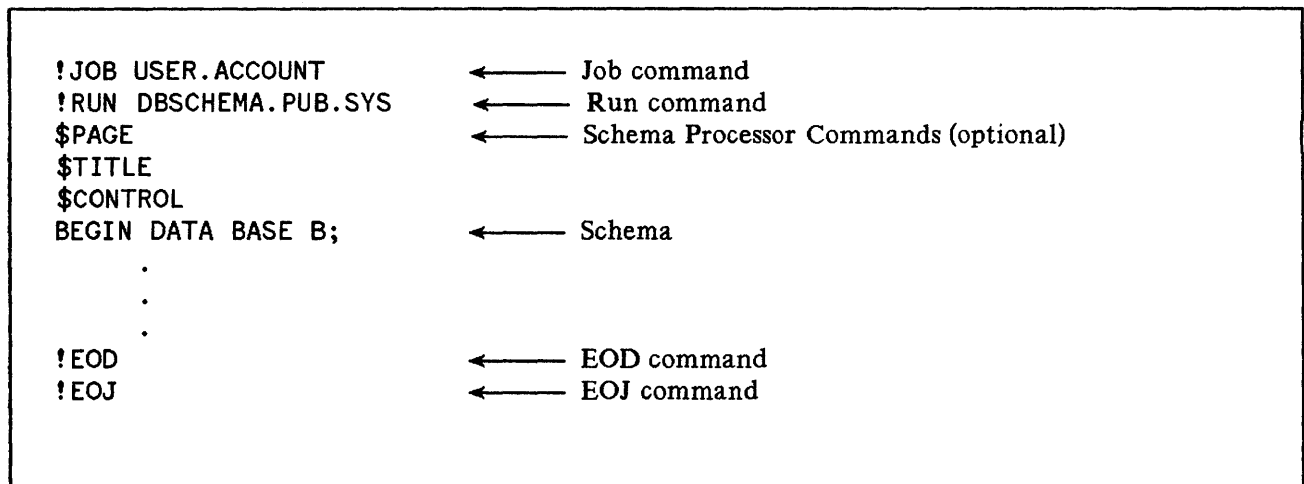


Figure 3-3. Schema Processor Batch Job Stream

## Continuation Records

To continue a command to the next record, use an ampersand (&) as the last non-blank character in the current record. The following record must begin with a \$. The records are combined and the \$ and & are deleted and replaced by one blank character. A command name or parameter cannot be broken by &. Characters beyond the 72nd character of each record are ignored.

# \$PAGE

The \$PAGE command causes the listfile to eject to the top of the next page, print character-strings which you may optionally specify, and skip two more lines before continuing the listing.

## Syntax

```
$PAGE [{"character-string"},...]
```

## Parameters

*character-string*            a list of characters enclosed in quotes. When the command is executed, the quotes are stripped and the character-strings are concatenated. A quote mark within a character-string is specified by a pair of quotes.

## Example

```
$PAGE "ORDERS DATA BASE SCHEMA", "VERSION 3"
```

```
$PAGE "MASTER DATA SETS"&  
$, "ACCOUNTING APPLICATION"
```

```
$PAGE
```

## Discussion

The \$PAGE command is effective only if the LIST option of the \$CONTROL command is on. The LIST option is on by default until a \$CONTROL command sets NOLIST. The \$PAGE command itself is not listed.

The contents of the character-strings replace those specified by a previous \$PAGE or \$TITLE command. If no character-strings are specified, the character-strings specified in the preceding \$PAGE or \$TITLE command, if any, are printed at the top of the next page.

# \$TITLE

The \$TITLE command specifies a list of characters to be printed each time a heading is printed on a new page. It does not cause a page eject.

## Syntax

```
$TITLE [{"character-string"},...]
```

## Parameters

*character-string* a list of characters enclosed in quotes. When the command is executed, the quotes are stripped and the character-strings are concatenated. A quote mark within a character-string is specified by a pair of quotes.

## Example

```
$TITLE ""PRELIM""ORDERS DATA BASE"
```

```
$TITLE "ORDERS DATA BASE SCHEMA JUNE, 1984"
```

## Discussion

The \$TITLE command may be overridden by a subsequent \$TITLE or \$PAGE command. If no *character-string* is specified, no title is printed after the command is encountered until another \$TITLE or \$PAGE command specifies one.

# \$CONTROL

The \$CONTROL command allows you to specify options in relation to processing the schema.

## Syntax

```
$CONTROL [LIST  
          NOLIST] [ERRORS=nnn] [,LINES=nnnnn] [,ROOT  
          NOROOT]  
          [,BLOCKMAX=nnnn] [,TABLE  
          NOTABLE]
```

## Parameters

LIST	causes each source record of the schema to be printed on the listfile.
NOLIST	specifies that only source records with errors be printed on the listfile. An error message is printed after these records.
ERRORS=nnn	sets the maximum number of errors to <i>nnn</i> . If more than <i>nnn</i> errors are detected, the Schema Processor terminates. <i>nnn</i> may have a value between 0 and 999, inclusive. The default value is 100.
LINES=nnnnn	sets the number of lines per page on the listfile to <i>nnnnn</i> which can be between 4 and 32767, inclusive. The default value is 60 if listfile is a line printer and 32767 if it is not.
ROOT	causes the Schema Processor to create a root file if no errors are detected in the schema.
NOROOT	prevents the Schema Processor from creating a root file.
BLOCKMAX=nnnn	sets the maximum physical block length (in words) for any data set in the data base. <i>nnnn</i> may have a value between 128 and 2048, inclusive. The default value is 512. This is an important parameter and is discussed in greater detail below.
TABLE	causes the Schema Processor to write a table of summary information about the data sets to the listfile device if no errors are detected.
NOTABLE	suppresses the TABLE option.

## Discussion

The default parameters are LIST, ROOT and TABLE. If no \$CONTROL command is used, the results are the same as if the following \$CONTROL command is used:

```
$CONTROL LIST, ERRORS=100,LINES=60,ROOT,BLOCKMAX=512,TABLE
```

The parameters may be placed in any order but must be separated by commas.

## Selecting the Block Size

The data set records are transferred from the disc to memory in blocks. (The block format is described in Section 10.) When you specify a maximum block size with the \$CONTROL command you should consider:

- Efficient disc space utilization.
- Minimum disc access.
- Program execution time which can be affected by the size of a privileged data segment in which TurboIMAGE maintains a Data Base Buffer Area Control Block. (Refer to Section 4 for a definition of the DBB.) Buffers in the DBB must be as large as the largest block of the data base, therefore, the larger the block, the larger each buffer must be.

The Schema Processor determines the number of data records which fit in a block. Larger blocks minimize disc access by enabling the transfer of more records at one time. In selecting a block size, the following considerations may apply:

- If the applications using the data base will be run as batch jobs at times when few other users are competing for system resources, particularly memory space, you may choose to use large blocks. This will reduce the frequency of disc access if an application is accessing data sets serially, or along chains whose members are physically contiguous or close.
- If the application programs are large and will be run while many users are operating in session mode, large blocks and the resulting large DBG and DBB data segments may cause the program to execute more slowly since a larger area of memory is required to execute the programs. In this case, you may want to decrease the block size. If the application programs are small, this may not be necessary.

Note that DBSCHEMA chooses a blocksize (less than or equal to the maximum blocksize) which makes the best use of disc space, and which may be substantially less than the maximum blocksize (as specified by \$CONTROL BLOCKMAX, or the default of 512 words). If the record size is greater than 512 words, BLOCKMAX must be set greater than or equal to the record size.

Other factors may depend on the application requirements and a certain amount of tuning is sometimes necessary to determine the best block size. In general, the default block size of 512 words yields reasonable performance and should be changed only with good reason.

## SCHEMA PROCESSOR OUTPUT

The Schema Processor prints the following heading on the first page of the listing:

PAGE 1 HEWLETT-PACKARD 32215C.00.00 TurboIMAGE/3000 MON, DEC 10, 1984, 4:32 PM

If your standard output device (\$STDLIST) is different from listfile, an abbreviated product identification is also printed on \$STDLIST. Subsequent pages of listfile are headed by a page number, the data base name if it has been encountered, and the title most recently specified by a \$TITLE or \$PAGE command.

If the LIST option is active, a copy of each record of the schema is sent to the listfile. However, if the textfile and listfile are the same, as for example they are when you enter the schema source from your terminal in session mode, the records are not listed. If you are entering the schema in this way, the Schema Processor prompts for each line of input with a >.

### Summary Information

After the entire schema has been scanned, several types of summary information may be printed on the listfile.

- If not all of the items defined in the item part are referenced in the set part, and if no errors are encountered, the message UNREFERENCED ITEMS: *list of items* is printed to the listfile. The list includes all items defined but not referenced in a data set. Although they are not considered errors, these extraneous items should be removed to reduce the size of the tables in the root file and the size of the extra data segment used by the library procedures.
- If no errors are detected in the schema, the Schema Processor prints a table of summary information about the data sets. Figure 3-4 contains a sample printout of this information. Table 3-7 describes the information contained in the summary. The NOTABLE parameter of the \$CONTROL command suppresses printing of this table.

DATA SET NAME	TYPE	FLD CNT	PT CT	ENTR LGTH	MED REC	CAPACITY	BLK FAC	BLK LGTH	DISC SPACE
EMPLOYEE	M	4	1	7	17	500	30	512	72
PROJECT-MASTER	M	2	1	10	20	75	19	382	15
LABOR	D	4	2	10	18	10024	28	506	1436
TOTAL DISC SECTORS INCLUDING ROOT:									1532

Figure 3-4. Data Set Summary Table



**Table 3-7. Data Set Summary Table Information**

<b>DATA SET NAME</b>	The name of the data set.	<b>CAPACITY</b>	The maximum number of entries allowed in the data set. For detail data sets, this number may differ from the number of entries specified in the schema itself, because the capacity of each detail is adjusted to represent an even multiple of the blocking factor (see below).
<b>TYPE</b>	A for automatic, M for manual, or D for detail.		
<b>FLD CNT</b>	The number of data items in each entry of the data set.	<b>BLK FAC</b>	The number of media records which are blocked together for transfer to and from the disc.
<b>PT CT</b>	Path count. For a master data set, this is the number of paths specified for the data set search item. For a detail data set, it is the number of search items defined for each entry of the data set.	<b>BLK LGTH</b>	The total length in words of the physical block as defined in BLK FAC. This includes the media records and a bit map. Bit maps are discussed in Section 10.
<b>ENTR LGTH</b>	The length in words of the data portion of the data entry (not including any of the TurboIMAGE pointers or structure information associated with a data entry).	<b>DISC SPACE</b>	The amount of disc space (in 128-word sectors) occupied by the MPE file containing the data set.
<b>MED REC</b>	The total length in words of a media record of the data set. This length includes the entry length plus any of the TurboIMAGE pointers associated with the data entry. Media records are discussed in Section 10.	<b>TOTAL DISC SECTORS INCLUDING ROOT: <i>nnnn</i></b>	The total number of 128-word disc sectors which will be occupied by the data base, when created using the DBUTIL program.

- Two lines of summary totals are printed on the listfile. For example:

```

NUMBER OF ERROR MESSAGES:  0
ITEM NAME COUNT:   22      DATA SET COUNT:   6

```

The error count includes both errors in the schema and in the Schema Processor commands. The error count is also sent to \$STDLIST, if it is different from the listfile.

- If no schema syntax or logical errors are encountered, a third line is printed. The form of this line is:

ROOT LENGTH:  $r$     BUFFER LENGTH:  $b$     TRAILER LENGTH:  $t$

ROOT LENGTH is the length in words of the body of the root file. BUFFER LENGTH is the length in words of each of the data buffers which TurboIMAGE allocates in an extra data segment (the DBB) for use in transferring data set blocks to and from disc. TRAILER LENGTH is the length in words of an area in the extra data segment used by TurboIMAGE to transfer information to and from a calling program's stack.

- If no errors are detected and the ROOT option is active, the following message is sent to the listfile:

ROOT FILE *data base name* CREATED

*data base name* is the name given in the BEGIN DATA BASE statement in the schema.

## Schema Errors

When the Schema Processor detects an error it prints a message to the listfile. If the LIST option is active, it is printed immediately after the offending statement. If NOLIST is active, the current line of the schema is printed and then the error message.

Schema Processor error messages are explained in Appendix A. The root file is not created if any of the listed errors are detected. However, the Schema Processor attempts to continue checking the schema for logical and syntactical correctness.

One error may obscure detection of subsequent errors, particularly if it occurs early in a data set. It may be necessary to process the schema again after the error is corrected to find subsequent errors. Conversely, some errors early in the schema can generate subsequent apparent errors which will disappear after the original error has been corrected.

If schema errors prohibit creation of the root file, the following message is sent to the listfile, and to \$STDLIST if it is not the same as the listfile:

PRECEDING ERRORS -- NO ROOT FILE CREATED

A few conditions, including the number of errors exceeding the total number allowed, cause immediate termination of the Schema Processor without the normal summary lines. In this case, the following message is printed:

SCHEMA PROCESSING TERMINATED

## Schema Processor Example

Figure 3-5 contains the listfile output printed when the schema of the sample ORDERS data base is processed. The data base has 5 passwords and contains 23 data item definitions and 6 data set definitions. The Schema Processor summary information is printed following the schema.

```

PAGE 1
HEWLETT-PACKARD 32215C.00.00 IMAGE/3000: DBSCHEMA FRI,DEC 7
$CONTROL LIST,LINES=46
$PAGE "SCHEMA FOR DATA BASE ORDERS"
BEGIN DATA BASE ORDERS;

PASSWORDS:
  14 CLERK;      << SALES CLERK >>
  12 BUYER;      << BUYER - RESPONSIBLE FOR PARTS INVENTORY >>
  11 CREDIT;     << CUSTOMER CREDIT OFFICE >>
  13 SHIP-REC;   << WAREHOUSE - SHIPPING AND RECEIVING >>
  18 DO-ALL;     << FOR USE BY MGMT >>

ITEMS:          << IN ALPHABETICAL ORDER FOR CONVENIENCE >>
ACCOUNT,        J2 ;          << CUSTOMER ACCOUNT NUMBER>>
BINNUM,         Z2 (/13);     << STORAGE LOCATION OF PROD>>
CITY,           X12 (12,13,14/11); << CITY>>
CREDIT-RATING,  R2 (/14);     << CUSTOMER CREDIT RATING>>
DATE,           X6 ;          << DATE (YYMMDD)>>
DELIV-DATE,     X6 (/14);     << DELIVERY DATE (YYMMDD)>>
DESCRIPTION,    X20 ;         << PRODUCT DESCRIPTION>>
FIRST-NAME,     X10 (14/11);   << CUSTOMER GIVEN NAME>>
INITIAL,        U2 (14/11);   << CUSTOMER MIDDLE INITIAL>>
LAST-NAME,      X16 (14/11);   << CUSTOMER SURNAME>>
LASTSHIPDATE,   X6 (12/ );    << DATE LAST REC D(YYMMDD)>>
ONHANDQTY,      J2 (14/12);   << TOTAL PRODUCT INVENTORY>>
PRICE,          J2 (14/);     << SELLING PRICE (PENNIES)>>
PURCH-DATE,     X6 (11/14);   << PURCHASE DATE (YYMMDD)>>
QUANTITY,       I (/14);     << SALES PURCHASE QUANTITY>>
STATE,          X2 (12,13,14/11); << STATE -- 2 LETTER ABB>>
STOCK#,         U8 ;          << PRODUCT STOCK NUMBER>>
STREET-ADD,     X26 (12,13,14/11); << NUMBER AND STREET ADD>>
SUPPLIER,       X16 (12,13/); << SUPPLYING COMPANY NAME>>
TAX,            J2 (14/);     << SALES TAX>>
TOTAL,          J2 (11,14/);  << TOTAL AMOUNT OF SALE>>
UNIT-COST,      P8 (/12);     << UNIT COST OF PRODUCT>>
ZIP,            X6 (12,13,14/11); << ZIP CODE>>

SETS:
NAME:           CUSTOMER,MANUAL(14/11,18),DISC; <<CUSTOMER MASTER>>
ENTRY:          ACCOUNT(1),
                LAST-NAME,
                FIRST-NAME,
                INITIAL,
                STREET-ADD,
                CITY,
                STATE,
                ZIP,
                CREDIT-RATING;

CAPACITY:       200;

```

Figure 3-5. ORDERS Data Base Schema

PAGE 2 SCHEMA FOR DATA BASE ORDERS

```

NAME:      DATE-MASTER,AUTOMATIC,DISC1;      <<DATE INDEX>>
ENTRY:     DATE(3);
CAPACITY:  211;
NAME:      PRODUCT,MANUAL(14,13/12,18),DISC2;<<PRODUCT INDEX>>
ENTRY:     STOCK#(2),
           DESCRIPTION;
CAPACITY:  300;
NAME:      SALES,DETAIL(11/14,18),DISC1; <<CREDIT PURCHASE>>
ENTRY:     ACCOUNT(CUSTOMER(PURCH-DATE)),
           STOCK#(PRODUCT),
           QUANTITY,
           PRICE,
           TAX,
           TOTAL,
           PURCH-DATE(DATE-MASTER),
           DELIV-DATE(DATE-MASTER);
CAPACITY:  500;
NAME:      SUP-MASTER,MANUAL(13/12,18),DISC1; <<SUPP MASTER>>
ENTRY:     SUPPLIER(1),
           STREET-ADD,
           CITY,
           STATE,
           ZIP;
CAPACITY:  200;
NAME:      INVENTORY,DETAIL(12,14/13,18),DISC1; <<PROD SUPPLY>>
ENTRY:     STOCK#(PRODUCT),
           ONHANDQTY,
           SUPPLIER(!SUP-MASTER),      <<PRIMARY PATH>>
           UNIT-COST,
           LASTSHIPDATE(DATE-MASTER),
           BINNUM;
CAPACITY:  450;

```

END.

DATA SET NAME	TYPE	FLD CNT	PT CT	ENTR LGTH	MED REC	CAPACITY	BLK FAC	BLK LGTH	DISC SPACE
CUSTOMER	M	9	1	41	52	200	7	365	90
DATE-MASTER	A	1	3	3	26	211	19	496	52
PRODUCT	M	2	2	14	31	300	16	497	80
SALES	D	8	4	19	35	504	14	491	148
SUP-MASTER	M	5	1	31	42	200	12	505	72
INVENTORY	D	6	3	20	32	450	15	481	124

TOTAL DISC SECTORS INCLUDING ROOT: 583

NUMBER OF ERROR MESSAGES: 0

ITEM NAME COUNT: 23 DATA SET COUNT: 6

ROOT LENGTH: 1176 BUFFER LENGTH: 505 TRAILER LENGTH: 256

ROOT FILE ORDERS CREATED.

Figure 3-5. ORDERS Data Base Schema (Continued)

After the data base is designed, the root file created, and the files built, application programs can be written that will be run to enter and use the data. Programs written in COBOL, FORTRAN, Pascal, SPL, or BASIC gain access to the data base through calls to TurboIMAGE procedures. RPG programs contain specifications used by the Report Program Generator to make calls to the TurboIMAGE procedures for you. This section contains a text discussion of the procedures used to open the data base, enter, read, update, and delete data, as well as information on locking, transaction logging, checking procedure status and interpreting errors. Use this section together with Section 5 which gives details about each procedure call, its parameters, and status information.

## NOTE

Before application programs can be executed, the data base must be created using DBUTIL TurboIMAGE utility program described in Section 8.

## OPENING THE DATA BASE

Before you can gain access to the data, the process you are running must open the data base with a call to the DBOPEN procedure. (A process is a unique execution of a particular program by a particular user at a particular time, as described in the *MPE Intrinsic Reference Manual*.) In opening a data base, DBOPEN establishes an access path between the data base and your program by:

- verifying your right to use the data base under the security provisions provided by the MPE file system and the TurboIMAGE user class/password scheme.
- determining that the access mode you have requested in opening the data base is compatible with the access modes of other users currently using the data base.
- opening the root file and constructing the control blocks to be used by all other TurboIMAGE procedures when they are executed. The root file remains open until the data base is closed.

Note that DBOPEN does not open the individual data sets that compose a data base.

DBOPEN also determines if the operating system supports the native language as defined in the root file. An error message "Language is not supported" will be returned if the language attribute of the data base is not supported by the current system configuration. Refer to Appendix A for more information.

## Data Base Control Blocks

TurboIMAGE executes using data stored in four different types of dynamically-constructed control blocks resident in privileged extra data segments. The Data Base System Control Block (DBS), the Data Base Globals Control Block (DBG), the Data Base Buffer Area Control Block (DBB), and the Data Base User Local Control Block (DBU). The Data Base System Control Block is created when the first user opens any data base on the system. The DBS is used as a system wide table to locate the current DBG and DBB for any opened data base. There is only one DBS per system.

TurboIMAGE creates the Data Base Globals Control Block (DBG) and the Data Base Buffer Area Control Block (DBB) for a particular data base when the first user's process calls the DBOPEN procedure to open the data base. Both will remain allocated until the last user closes the data base (DBCLOSE).

The DBG contains global information required by TurboIMAGE intrinsics during run-time. There is exactly one DBG for each open data base regardless of the number of concurrent access paths to the data base. All TurboIMAGE procedures on a particular data base (except DBERROR and DBEXPLAIN) reference the DBG. In addition, the DBG contains the lock table which holds user level locking information.

The DBB contains a set of buffers which may contain data from any of the data sets. Global information regarding logging and recovery is also contained within the DBB. The DBB is used to retrieve, log and update data located in the data set files on disc.

There is one Data Base User Local Control Block (DBU) for each user who accesses a data base. In other words, a unique DBU is created each time DBOPEN is successfully called. A DBU contains information pertaining to the user's own individual access to the data base. The privileged extra data segment containing the DBU is associated with the user's process.

All TurboIMAGE intrinsics process on the DBU except accesses for global and buffer area information found in the two global blocks (DBG and DBB). The DBU is released when the user's process calls DBCLOSE to close the data base.

## Passwords

When you open the data base you must provide a valid password to establish your *user class number*. If you do not provide one, you will be granted user class number 0. If you are the data base creator and supply a semicolon as a password, the number 64 is used to grant you unlimited data base access privileges. Passwords and user classes are discussed in Section 2.

## Access Modes

There are eight different access modes available and each mode determines the type of operation that you can perform on the data base as well as the types of operations other users can perform concurrently. To simplify the definition of the various access modes, the following terminology is used:

- read access allows the user to locate and read data entries.
- update access allows read access and, in addition, allows the user to replace values in all data items except search and sort items.
- modify access allows update and, in addition, allows the user to add and delete entries.

The procedures that can be used with each type of access are:

- read DBFIND and DBGET.
- update DBFIND, DBGET, and DBUPDATE.
- modify DBFIND, DBGET, DBUPDATE, DBPUT, and DBDELETE.

Table 4-1 summarizes the type of access granted in each access mode, provided the MPE security provisions and your password permit it. Access modes 3 and 7 provide exclusive access to the data base; all other modes allow shared access.

**Table 4-1. Access Mode Summary**

ACCESS MODE	TYPE OF ACCESS GRANTED	CONCURRENT ACCESS ALLOWED	SPECIAL REQUIREMENTS
1	modify	modify (with locking)	Locking must be used for update or modify.
2	update	update	
3	modify	none	
4	modify	read	
5	read	modify (with locking)	TurboIMAGE does not require locking but it should be used to coordinate access with users who are modifying.
6	read	modify	
7	read	none	
8	read	read	

### CONCURRENT ACCESS MODES

A data base can only be shared in certain well-defined environments. The access mode specified when a process opens a data base must be acceptable for the environment established by others who are already using the data base. Here is a summary of the acceptable environments:

- multiple mode 1 and mode 5 users.
- multiple mode 6 and mode 2 users.
- multiple mode 6 users and one mode 4 user.
- multiple mode 6 and mode 8 users.
- one mode 3 user.
- one mode 7 user.

Subsets of these environments are also allowed. For example, there may be all mode 6 users or all mode 8 users. There may be one mode 1 user or all mode 5 users and so forth.

If a mode 3 or mode 7 user is currently accessing the data base, it cannot be opened until that user closes the data base. This is true any time an attempt is made to open a data base in a mode which is not compatible with the modes of others using the data base.

## DATA BASE OPERATIONS

The descriptions below explain in detail exactly what occurs when a data base is opened in a particular mode. Locking is available in all modes. In the discussion that follows, brief suggestions are given as to when locking may be used. Refer to the discussion of the locking facility for more information.

- **ACCESS MODE 1.** The data base is opened for shared modify access. Opening in mode 1 succeeds only if all other current users of the data base have access modes 1 or 5.

All TurboIMAGE procedures are available in this mode. However, a program must obtain temporary exclusive control of the data entries before calling any procedure that changes them, such as DBUPDATE, DBPUT, or DBDELETE. In this way, changes to the data base are synchronized and carried out properly. This exclusive control must subsequently be relinquished to permit other access mode 1 or mode 5 users to access these entries. Acquiring and relinquishing is referred to as locking and unlocking, respectively. These functions are supplied by the TurboIMAGE library procedures, DBLOCK and DBUNLOCK. The locking requirements may be met by locking the affected entries, the sets containing the entries, or the whole data base.

A mode 1 (and mode 5) user who has all or part of the data base locked is assured that no concurrent user is modifying that part of the data base.

It is possible to read entries in the data base using calls to DBFIND and DBGET without locking but the calling program must provide for the possibility that another process may be simultaneously modifying the data base. This can result in an entry being deleted from a chain which the calling program is reading.

- **ACCESS MODE 2.** The data base is opened for shared update access. The opening succeeds only if all current users of the data base have access modes 2 and 6. All TurboIMAGE procedures are available to the mode 2 user except DBPUT and DBDELETE which are permanently disabled in this mode. Therefore, the mode 2 user is able to read and update data entries but is not permitted to add or delete data entries in any data set.

The programmer must be aware of the possibility that other mode 2 users are simultaneously updating data entries. In many applications, it may be possible to arrange for each user process to update unique data entries or data items so that the data base will correctly reflect all changes, even data items in the same entry updated by different processes. On the other hand, if two or more processes update the same data items of the same entry, the data base will reflect only the latest values. Locking may be used, if desired, to coordinate update sequences to an entry or to coordinate with mode 6 readers.

- **ACCESS MODE 3.** The data base is opened for exclusive modify access. If any other users are accessing the data base, it cannot be opened in this mode. All TurboIMAGE procedures are available to the mode 3 user. No other concurrent process is permitted to gain any type of access to the data base.



- **ACCESS MODE 4.** The data base is opened for semi-exclusive modify access. Only one mode 4 user can access the data base and all other current users must be in mode 6 (read only). The mode 4 user is permitted to call any TurboIMAGE procedure and has complete control over data base content. This mode differs from mode 3 only in that other read-only users are permitted concurrent access to the data base. Locking may be used to coordinate with mode 6 readers.
- **ACCESS MODE 5.** The data base is opened for shared read access. All other concurrent users must be in mode 1 or mode 5. Mode 5 operates in exactly the same way as mode 1 except the procedures that alter the data base, DBUPDATE, DBPUT, and DBDELETE, are disabled for the mode 5 user. Locking can be used, if desired, to ensure that data is not being modified while you are reading it.
- **ACCESS MODE 6.** The data base is opened for shared read access. Concurrent users must be in mode 2, 4, 6, or 8. This mode can also be used while the data base is being stored with the TurboIMAGE utility program, DBSTORE. Some of these modes are incompatible with each other as shown in the discussion of concurrent access modes above. All TurboIMAGE procedures that alter the data base are disabled. Locking can be used to synchronize with users who are concurrently updating.

Mode 5 and 6 are appropriate for inquiry-type applications if they can tolerate the possibility of data base modifications taking place simultaneously, since mode 1, 2, and 4 users can make such changes.

- **ACCESS MODE 7.** The data base is opened for exclusive read access. No other users may access the data base concurrently. Mode 7 operates in exactly the same way as mode 3 except the procedures that alter the data base are disabled for the mode 7 user.
- **ACCESS MODE 8.** The data base is opened for shared read access. Concurrent users must be in mode 6 or 8 or using the TurboIMAGE utility DBSTORE. TurboIMAGE procedures that alter the data base are not permitted. Since mode 8 allows only concurrent readers, a user program with this access mode can be assured that the data base values it reads are unchanging.

## SELECTING AN ACCESS MODE

When deciding which access mode to use, two important considerations are:

- Use the least capability that will accomplish the task. For example, select a read only access mode (5,6,7, or 8) if the program does not alter the data base in any way.
- Allow concurrent users as much capability as is consistent with successful completion of the task. If the task is merely browsing through the data base, producing a quick report, or accessing an unchanging portion of the data base, choose a mode which allows concurrent users to make data base modifications to other parts of the data base. Allowing concurrent read-only access (modes 2, 4, and 8) may be appropriate in many situations. For programs that must be assured there will be no concurrent structural changes but can tolerate simultaneous updates to entries, mode 2 may be particularly suitable. Locking may be used to control simultaneous updates to a data entry. If it is absolutely necessary to make structural changes to a data base from concurrent multiple processes, modes 1 and 5 must be used. Fully exclusive operation (modes 3 and 7) are available if needed.

## Using the Data Base

The following mode selection guidelines are organized according to the task to be performed. For some tasks, one of several modes may be selected depending on the concurrent activity allowed with each mode.

- Programs that perform all data base operations, including adding and deleting entries, should open with mode 1, 3, or 4. Choose:
  - mode 1      if it is necessary to allow other processes to add and delete entries simultaneously. In this case, the affected parts of the data base must be locked while performing updates, additions, or deletions.
  - mode 4      if exclusive ability to change the data base is required but it is possible to allow mode 6 processes to read the data base while changes are being made.
  - mode 3      if the program must have exclusive access.
- Programs that locate, read and replace data in existing entries but do not need to add or delete any entries, and do not want any other processes to do so, should open the data base in mode 2. Locking can be used to coordinate updates.
- Programs that only locate and read or report on information in the data base should open with one of the read only modes. In this case, the mode selected depends upon either the type of the process running concurrently or the need for an unchanging data base while the program is running. Choose:
  - mode 5      if concurrent processes will operate in modes 1 or 5. Parts or all of the data base may optionally be locked to prevent concurrent changes during one or more read operations.
  - mode 6      if it is not important what other processes are doing to the data base. In this case, mode 2 processes can replace entries, one mode 4 user can replace, add or delete entries, or mode 6 or mode 8 users can read entries while the program is using the data base.

## DYNAMIC LOCKING

Refer to the discussion of locking and unlocking later in this section for some special considerations.

## Transaction Logging

Users accessing the data base in access modes 1 through 4 are affected by the transaction logging facility if the data base administrator has enabled the data base for logging (a procedure described in Section 7). In this case, calls to the TurboIMAGE intrinsics listed in Table 4-2 are automatically logged to a logfile. Note that nothing is logged for programs opening the data base with read only access (modes 5-8), regardless of the data base having been enabled for logging. (The logging facility is described more fully later in this section and in Section 7.)

**Table 4-2. Logged Intrinsics**

DBBEGIN	DBCLOSE	DBDELETE	DBEND
DBMEMO	DBOPEN	DBPUT	DBUPDATE

## ENTERING DATA IN THE DATA BASE

Data is added to the data base, one entry at a time, using the DBPUT procedure. You may add data entries to manual master and detail data sets. Entries are automatically added to automatic master data sets when you add entries to the associated detail data sets.

To add an entry, you specify the data set name or number, a list of data items in the set, and the name of a buffer containing values for these items. Values must be supplied for search and sort items but are optional for other data items in the entry. If no value is supplied, the data item value is set to binary zeroes.

## Sequence for Adding Entries

Before you can add an entry to a detail data set indexed by a manual master data set, the manual master must contain an entry with a search item value equal to the one you intend to put in the detail. If more than one manual master is used to index the detail, entries which have a search item value identical to the detail search item value for the same path must exist in each master. To illustrate, consider the ORDERS data base again. Figure 4-1 contains sample data entries in four of the ORDERS data sets.

Before the SALES data entry can be added to the data set, the CUSTOMER manual master data set must contain an entry with ACCOUNT equal to 12345678 since ACCOUNT is the search item used to index the SALES detail. Similarly, the SALES data set is indexed by the PRODUCT manual master through the STOCK# search item, so the entry with STOCK# equal to 34624AB3 must be added to PRODUCT before a sales transaction for that STOCK# can be entered in SALES.

Once the entry for customer account 12345678 has been entered, the next sales transaction can be entered in the SALES detail set without changing the CUSTOMER master. This entry will be chained to the previous entry for the account. If a different customer buys a bicycle tire pump, the PRODUCT data set will not require any additional entries, but if the customer's account is not yet in the CUSTOMER data set it must be added before entering the sales transaction in SALES.

When the entry for account 12345678 and stock number 35624AB3 is added to SALES, TurboIMAGE automatically adds entries to the DATE-MASTER with a DATE item value of 92775 and 92875 if such entries do not already exist. If the entries do exist, each chain head is modified to include the entry added to the chain.

## Access Mode and User Class Number

An entry cannot be added to a data set unless the user class number established when the data base is opened grants this capability. The user class number must be in the data set write class list.

The data base must also be opened with an access mode allowing entries to be added. These access modes are 1, 3, and 4. If it is opened with access mode 1, the DBLOCK procedure must be used to establish a lock covering the entry to be inserted. For detail data sets, this may be a data entry, data set, or data base lock.

Note that the locking mechanism will accept a request to lock a data entry that does not yet exist, therefore, you may lock a data entry before you add it.

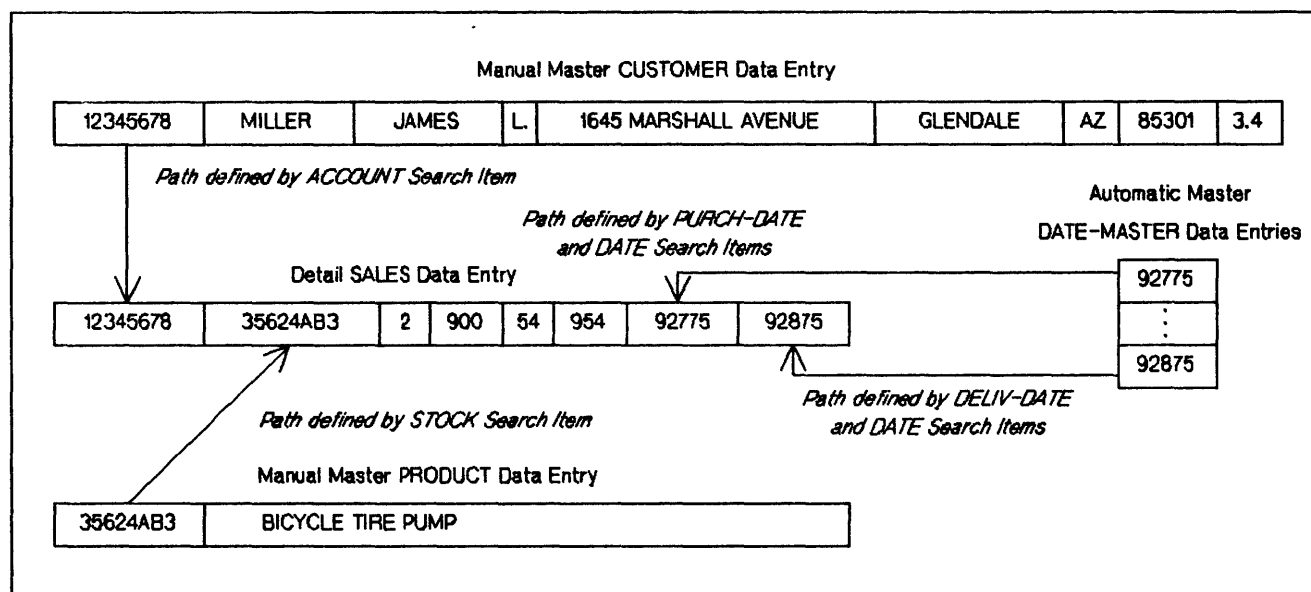


Figure 4-1. Sample Data Entries from ORDERS Data Base

## Search Items

TurboIMAGE performs checks on the values of search items before adding an entry to a data set. If the data set is a manual master, TurboIMAGE verifies that the search item value is unique for the set, that no entry currently contains a search item with the same value. If the data set is a detail, TurboIMAGE verifies that the value of each search item forming a path with a manual master has a matching value in that master. It also checks that there is room to add an entry to any automatic master data sets linked to the detail if a matching search item value does not exist.

## READING THE DATA

When you read data from the data base you specify which data set and which entry in that data set contains the information you want. If the user class number with which you opened the data base grants you read access, you may read the entire entry or specific data items from the entry. You specify the items to be read and the array where the values should be stored. You can read items or entries in any access mode if your user class grants read access to the data element.

To understand the various ways in which you can select the data entry to be read, it is important to know a little about the data set structure. Each data set consists of one disc file and each data entry is a logical record in that file. Each entry is identified by the relative record number in which it is stored. The first record in the data set is record number 1 and the last is record number  $n$ , where  $n$  is the capacity of the data set.

At any given time, a record may or may not contain an entry. TurboIMAGE maintains internal information indicating which records of a data set contain entries and which do not.

### Current Path

TurboIMAGE maintains a *current path* for each detail data set and for each accessor (access path). The current path is established by the DBFIND procedure, or if no call has been made to this procedure, it is the primary path for the data set. Each time an entry is read, no matter what read method is used, TurboIMAGE saves the entry's backward and forward chain pointers for the current path. For more information about how the current path is used, refer to the discussion of chained access later in this section.

If an entry is read from a master data set, the chain pointers are synonym chain pointers and have no relationship to a path.

### Reading Methods

The methods for requesting a data entry are categorized as

- directed access
- serial access
- calculated access
- chained access

All of these methods are available through the TurboIMAGE library procedure DBGET. The chained access method also requires the use of the DBFIND procedure. Figure 4-2 illustrates the access methods using two data sets from the ORDERS data base.

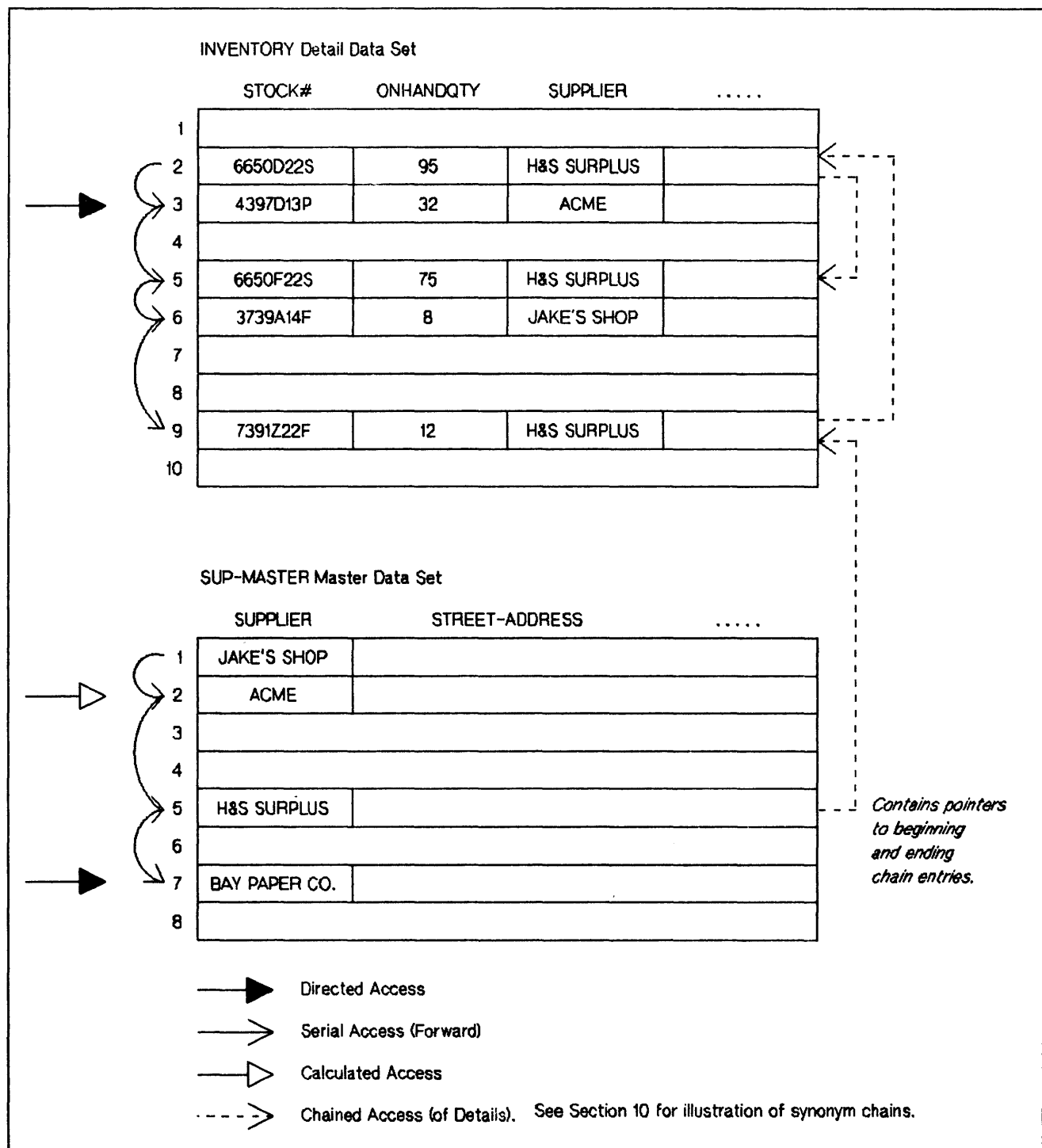


Figure 4-2. Reading Access Methods (DBGET Procedure)

## Directed Access

One method of selecting the data entry to be read is to specify its record number. This method is called directed access. If any entry exists at the record address specified by the calling program, TurboIMAGE returns the values for the data items requested in the calling program's buffer. If no such entry exists, the program is notified by an exceptional condition return such as end-of-file or beginning-of-file.

This access method can be used with any type of data set and is useful in situations where the calling program has already determined the record number of the entry to be read. For example, if a program surveys several entries using another access method to determine which one it wants to use in a report, it can save each record number and use the record number of the entry it selects to read the entry again using the directed access method.

If a program performs a directed read of record 3 of the INVENTORY data set, the entry marked with a solid black arrow in Figure 4-2 is read. If a directed read of the SUP-MASTER data set record 7 is performed, the entry in that set marked with the same type of arrow is read.

## LOCKING

If concurrent users are allowed to add to or delete from this data set, locking should be used during the search and report sequence to ensure the record numbers do not change before they are used. In this type of application, a data set lock is usually the most appropriate.

<b>NOTE</b>
-------------

When using this type of access with master data sets, you should be aware of migrating secondaries. These are described in Section 10.

## Serial Access

In this mode of retrieval, TurboIMAGE starts at the most recently accessed storage location for the data set, called the *current record*, and sequentially examines adjacent records until the next entry is located. Data items from this entry are returned to the calling program, and its location becomes the current record.

You may use both forward and backward serial access. Forward serial access consists of retrieving the next greater-numbered entry and backward serial access consists of retrieving the previous lower-numbered entry. If no entry is located, TurboIMAGE returns an end-of-file if requested access is forward, or a beginning-of-file if it is backwards.

Since there is no current record the first time a program requests an entry from a data set, a request for forward serial access causes TurboIMAGE to search from record 1. Similarly, a backward serial retrieval begins at the highest numbered record.

The entries connected by a solid line in Figure 4-2 are read by a program using the serial access method. If a forward serial read is performed on the INVENTORY data set before any other type of read, the entry in record number 2 is read. If another forward serial read is performed on the same data set, the entry in record 3 is read. On the other hand, if a serial read is performed and the current record is 6, the entry in record 9 is read. The next forward serial read returns an end-of-file.

The serial access method can be used with any type of data set and is very useful if most or all of the data in the data set is to be retrieved, for example, to be used in a report. It is efficient to retrieve all the data serially, copy it to a file, and sort it with routines external to TurboIMAGE before printing the report. The availability of serial access effectively allows you to use a data set in the same way you would use an MPE file. Thus, you have the advantages of TurboIMAGE data base organization and the efficiency of serial access.

### LOCKING

If concurrent users are allowed to modify the data set (access mode 1), you may wish to lock the data set or data base before you begin the serial access sequence. Locking will prevent entries from being added, modified, moved or removed by the other processes.

### Calculated Access

The calculated access method allows you to retrieve an entry from a master data set by specifying a particular search item value. For example, the SUP-MASTER data entry for the supplier Acme shown in Figure 4-2, can be retrieved with this method since SUPPLIER is a search item in the SUP-MASTER data set. TurboIMAGE locates the entry in the data set whose search item value matches the requested value. The exact technique used to perform calculated access is described in Section 10.

Calculated access can be used only with master data sets. It is very useful for retrieving a single entry for some special purpose. For example, a program used infrequently to get information about a particular customer or supplier could use calculated access to quickly locate the information in the ORDERS data base.

### Chained Access

The chained access method is used to retrieve the next entry in the current chain. To perform chained access of detail data set entries, you must first locate the beginning of the chain you want to retrieve, and thus establish the current chain, by calling the DBFIND procedure. The calling program specifies the name of the detail search item that defines the path to which the chain belongs and a value for the item. TurboIMAGE determines which master set forms a path with the specified search item and locates the entry in that master data set whose search item value matches the specified value. The entry it locates contains pointers to the first and last entries in the desired chain and a count of the number of entries in the chain. This information is maintained internally and defines the attributes of the current path.



If a program uses chained access to read the INVENTORY data set entries pertaining to the supplier H&S SURPLUS shown in Figure 4-2, it must first call the DBFIND procedure to locate the chain head in the SUP-MASTER data set. The program specifies the INVENTORY data set, the SUPPLIER search item in the INVENTORY data set and the value H&S SURPLUS for that item. TurboIMAGE uses a calculated read to locate the SUP-MASTER entry with a search item value of H&S SURPLUS. If the program then requests a forward chained read using the DBGET procedure, the entry in record 9 of INVENTORY, which is set at the beginning of the chain, is read. If a backward chained read is requested, the entry in record 5 is read.

If the last call to DBGET used chained access to read the entry in record 9, the next forward chained read reads the entry in record 2 of the INVENTORY data set.

Once a current path, and chain, has been established for a detail data set, the calling program can use the chained access method of retrieving data. You may use both forward and backward chained access. In either case, if there are no more entries in the chain when you request the next one, DBGET returns an exceptional condition, beginning-of-chain or end-of-chain for backward and forward access, respectively.

Chained access to master data sets retrieves the next entry in the current *synonym* chain. The use of synonym chains applies to only a limited number of special situations. They are discussed in Section 10.

Chained access to detail data sets is particularly useful when you want to retrieve information about related events such as all inventory records for the H&S Surplus supplier in the ORDERS data base.

## LOCKING

If concurrent users are allowed to modify data entries in the chain you are currently accessing, you may use locking to ensure data consistency. For example, suppose a chain consists of several data entries, each containing a line item from a particular order. If user A is performing a series of chained reads while user B is cancelling the order by deleting data entries one by one, user A may retrieve an incomplete order. To prevent this from happening, a lock may be established covering the group of data entries to be retrieved (the chain, in this case). This can usually be done with a single DBLOCK call. (Refer to the discussion of the locking facility later in this section.)

## Re-Reading the Current Record

The DBGET library procedure allows you to read the entry from the most recently accessed record again. You may want to do this in a program that has unlocked the data entry and locked it again and needs to check if the contents of the current entry have been changed.

Note that if a DBFIND procedure call has been made, the current record is zero and a request to re-read the entry causes DBGET to return an exceptional condition indicating that the current record contains no entry. Refer to the DBGET procedure Table 5-12 for more information.

## UPDATING DATA

TurboIMAGE allows you to change the values of data items that are not search or sort items if the user class number with which you opened the data base grants this capability to you. Before you call the DBUPDATE library procedure to change the item values, you must call DBGET to locate the entry you intend to update. This sets the current record address for the data set. The DBUPDATE library procedure uses the current record address to locate the data items whose values are to be changed.

A lock may be established before the call to DBGET to guard against accidental modification of the record by another user. This is recommended in any shared access mode (as discussed below).

When the program calls DBUPDATE it specifies the data set name, a list of data items to be changed, and the name of a buffer containing values for the items. For example, if a program changes the street address of a customer in the CUSTOMER data set of the ORDERS data base, the program can first locate the entry to be changed by calling DBGET in calculated access mode with the customer's account number and then calling the DBUPDATE procedure to change the value of the STREET-ADDRESS data item in that entry.

## Access Modes and User Class Number

To update data items, the data base must be opened in access mode 1, 2, 3, or 4. If it is open in access mode 1, the data entry, data set, or data base must be locked while the update is happening.

TurboIMAGE guarantees that all updates to a data entry will be carried out even if they are requested by different users concurrently and locking is not used. To ensure this, TurboIMAGE always completes the processing of one DBUPDATE request before it begins processing under another. However, data consistency problems may still occur if an update is based on data values that are not current. For example, while withdrawing 10 items from the stock, two users may read the same data entry from the INVENTORY data set. If the current value of ONHANDQTY is 30 and they each subtract 10 from it and then update the entry, both updates will operate successfully but the new value will be 20 rather than 10. To prevent errors such as this, a lock covering the data entry can be put in effect before it is read and released after it is updated.

TurboIMAGE attempts to enforce this locking technique for users in mode 1 by checking to see if an appropriate lock is in effect before executing an update. However, to have its proper effect, the lock should be made before the call to DBGET.

The password you use to open the data base must grant update capability to the data items you intend to change. The user class number associated with the password must either be in the write class list of the data set containing the items to be updated or in the read class list of the data set and in the write class list of the data item.

## Updating Search and Sort Items

You cannot use the DBUPDATE library procedure to update a search or sort item. To change such items, you must first delete the selected entry with DBDELETE (see "DELETING DATA ENTRIES" below), and then write a new entry to the data base with DBPUT.

The new entry must be complete. That is, you cannot delete an entry and then add a new entry with only the item you want changed. If you do this, the rest of the entry will be set to binary zeros by DBPUT. Furthermore, make sure the current list is truly current when using an asterisk (\*) to reference the list; otherwise, if items have been added or deleted, you may cause DBPUT to write binary zeros over existing data. Note that using the commercial "at sign" (@) to write all the items in a data entry avoids this problem.

## DELETING DATA ENTRIES

To delete an entry from a data set, you must first locate the entry to be deleted by reading it with the DBGET library procedure, or the DBFIND and DBGET procedures if it is advantageous to use chained access to locate the entry. You then call the DBDELETE procedure specifying the data set name. TurboIMAGE verifies that your password and associated user class number allow you to delete the current entry of the specified data set.

If the detail data entry deleted is the only member of a detail chain linked to an automatic master, and all other chains linked to the same automatic master entry are empty, TurboIMAGE automatically deletes the master entry.

If the data entry is in a manual master data set, TurboIMAGE verifies that the detail chains associated with the entry's search item, if any, are empty. If not, it returns an error condition to the calling program. For example, if a program attempts to delete the SUP-MASTER entry in Figure 4-2 that contains a SUPPLIER value of H&S SURPLUS, an error condition is returned since a three-entry chain still exists in the INVENTORY detail data set.

To delete the CUSTOMER data set entry with ACCOUNT equal to 75757575, the program can call DBGET in calculated access mode specifying the CUSTOMER data set and the search item value 75757575. If the procedure executes successfully, the program then can call DBDELETE specifying the CUSTOMER data set to delete the current entry provided no chains in the related SALES detail data set contain search item values of 75757575.

## Access Modes and User Class Numbers

To update data items, the data base must be opened with access mode 1, 3, or 4. If it is opened with access mode 1, the DBLOCK procedure must be used to lock the detail data entry, data set, or data base before an entry can be deleted and DBUNLOCK should be called after one or all desired entries have been deleted. As a general rule, the lock should be established before the whole delete sequence, in other words, before the call to DBGET that establishes which record is to be deleted. This will ensure that another user does not delete the data entry between the call to DBGET and the call to DBDELETE.

An entry cannot be deleted from a data set unless the user class number established when the data base is opened is in the data set write class list.

## USING THE LOCKING FACILITY

The DBLOCK procedure applies a logical lock to a data base or one or more data sets or data entries. The DBUNLOCK procedure releases these locks.

Locking can be viewed as a means of communication and control to be used by mutually cooperating users. The locking facility provides a method for protecting the logical integrity of the data shared in a data base. With the DBLOCK procedure, application programs may isolate temporarily a subsection of the data base in order to perform a transaction against the isolated data. Locking is not required to protect the structure of the data base. TurboIMAGE has internal mechanisms that do this.

If a program opens the data base in access mode 1 and locks a part of the data base, it can perform the transaction with the certain knowledge that no other user will modify the data until the application program issues a DBUNLOCK call. This is because TurboIMAGE does not allow changes in access mode 1 unless a lock covers the data to be changed. If one process has the data base opened in access mode 1, TurboIMAGE requires that all other processes that modify the data base must also operate in access mode 1.

The DBLOCK procedure operates in one of six modes. Modes 1 and 2 may be used for locking the data base and modes 3 and 4 for locking a data set. In modes 5 and 6, you describe the data base entity or entities to be locked using *lock descriptors*.

At the data entry level, locking is performed on the basis of data item values. For example, suppose a customer requests a change in an order he has placed. The data entries for his account that are in the SALES data set may be locked while his order is changed and other data base activity may continue concurrently.

## Lock Descriptors

A lock descriptor is used to specify a group of data entries that are to be locked. It consists of a data set name or number, a relational operator, and an associated value. For purposes of this discussion, the notation *dset : ditem relop value* is used. For example, the lock descriptor SALES:ACCOUNT = 89393899 requests locking of all the data entries in the SALES data set with an ACCOUNT data item equal to 89393899. Note that the result of specifying a single lock descriptor may be that none, one or many entries are locked depending on how many entries qualify.

The following relational operators may be used:

- less than or equal (<=).
- greater than or equal (>=).
- equal (= # or # =). # indicates a space character.

The value must be specified exactly as it is stored in the data base. A lock will succeed even if no data item with the specified value exists in the data set; no check is made to determine the existence of a particular data item value. This allows you to use techniques such as issuing a lock to cover a data entry before you actually add it to a data set.

With the exception of compound items, any data item may be used in a lock descriptor; it need not be a search item.

TurboIMAGE does not require that you have read or write access to a data set or data item in order to specify it in a lock request.

A process may specify any number of lock descriptors with a single DBLOCK call. For example, the following lock descriptors may be specified in one DBLOCK call:

```
CUSTOMER: ACCOUNT = 89393899
SALES: ACCOUNT = 89393899
SUP-MASTER: STATE = AZ
INVENTORY: ONHANDQTY < = 100
INVENTORY: ONHANDQTY > = 1500
```

<b>NOTE</b>
-------------

Multiple calls to DBLOCK without intervening calls to DBUNLOCK are not allowed unless the program has Multiple RIN (MR) capability. (Refer to "Issuing Multiple Calls to DBLOCK" later in this section.)

## How Locking Works

The internal implementation of locking does not involve reading or writing to the data base element to be locked. TurboIMAGE keeps a table of everything that is locked by all processes that have the data base opened. One table is associated with each data base. This table serves as a global list of lock descriptors. In locking mode 5 or 6, a data base lock is specified with the descriptor @@ and a data set lock with *dset:@*. If you call DBLOCK in locking mode 1, 2, 3, or 4, TurboIMAGE sets up the appropriate lock descriptor and puts it in the lock descriptor table. Figure 4-3 illustrates the contents of this list in a situation where one process has locked all SALES data entries with ACCOUNT equal to 12121212 or equal to 33334444. Another process has locked all INVENTORY data entries with STOCK# equal to 6650D22S. A third process has locked the whole SUP-MASTER data set. The figure illustrates what the table represents, not the actual internal format.

When a lock request is made, TurboIMAGE compares the newly specified lock descriptors with those that are currently in the list. If a conflict exists, TurboIMAGE notifies the calling process that the entity cannot be locked or, if the process has requested unconditional locking, it is placed in a waiting state until the entity can be locked. If there are no conflicts, TurboIMAGE adds the new lock descriptors to the list.

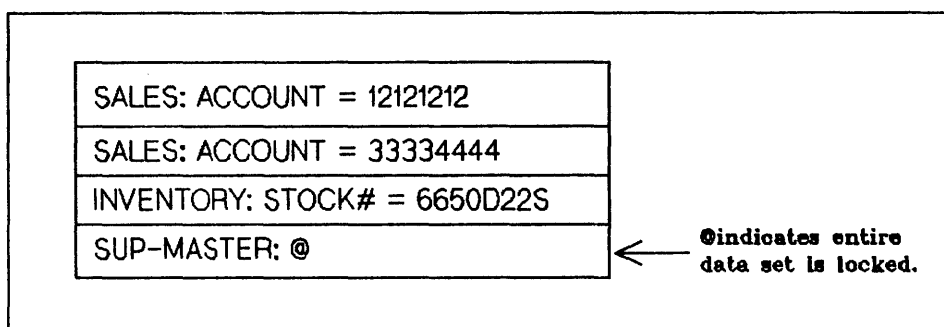


Figure 4-3. Lock Descriptor List

## Conditional and Unconditional Locking

You may request conditional or unconditional locking. If you request unconditional locking, TurboIMAGE returns control to your calling program only after the specified entity has been locked. If you request conditional locking, TurboIMAGE returns immediately. In this case, the condition code must be examined to determine whether or not the requested locks have been applied. If multiple lock descriptors are specified, the status area indicates the numbers that have been applied. The calling program should call DBUNLOCK only if a subset of the requested locks succeeded.

## Access Modes and Locking

It is anticipated that access mode 1 will typically be used by applications implementing a locking scheme. In this mode, TurboIMAGE enforces the following rules:

- To modify (DBPUT, DBDELETE, or DBUPDATE) a data entry, you must first issue a successful lock covering the affected data entry. It may be a data entry, data set, or data base lock.
- To add to or delete from (DBPUT or DBDELETE) a master data set, you must first successfully lock the data set or data base. To update (DBUPDATE) a master data set, data entry level locks are sufficient.

If your application opens the data base in access mode 2, it is recommended that you use locking to coordinate updates with other users.

TurboIMAGE does not prevent any process from reading data even though another process holds a lock on it. If you want to ensure that no modifications are in progress while you are reading from the data base, you should place an appropriate lock on the data before starting. Therefore, you may want to use locking in access modes 2, 4, 5, and 6 to coordinate the reading and modifying sequences and ensure that they do not occur concurrently.

Since access mode 3 and 7 users have exclusive control of the data base and access mode 8 users allow concurrent reading only, locking need not be used in these modes.

## Automatic Masters

When adding or deleting entries from a detail data set, you need not have locks covering the implicit additions or deletions that occur in any associated automatic masters.

## Locking Levels

Locking can be viewed as operating on three levels: the whole data base, whole data sets, or data entries. TurboIMAGE allows mixed levels of locking. For example, one user may be locking data entries and another locking the data set. In this situation, a request to lock the data set cannot succeed until all the currently locked data entries have been released. Subsequent requests to lock data entries, those that are made while the data set lock is pending, are placed in a queue behind the data set lock.

This principle is followed for data base locks also. If data set or data entry locks are in effect at the time a data base lock is requested, the data base lock must wait until they are released and all subsequent locking requests must wait behind the pending data base lock.

In either case, if the request is for a conditional lock, an exceptional condition is generated. (Refer to Table 5-15.)

## Deciding on a Locking Strategy

It is important, especially for on-line interactive applications, to establish a locking strategy at system design time. In general, locking is related to the *transaction*, the basic unit of work performed against a data base. Typically a transaction consists of several calls to TurboIMAGE intrinsics to locate and modify data. For example, a transaction to add a new order with three line items may require several reads to locate customer information and several DBPUT calls to add the order detail records.

One characteristic of a transaction is that the data in the data base is consistent both before and after the transaction, but not while it is in progress. For example, a user reading the detail data set being modified by the above order transaction may only see some of the line items and may get no indication that the transaction is incomplete. This type of problem is referred to as *logical inconsistency* of data and can be prevented by using the locking facilities.

The general principle that should be applied for *any* transaction in a shared-access environment is: *At the start of any transaction, establish locks that cover all data entries that you intend to modify (DBPUT, DBDELETE, or DBUPDATE) and/or all data entries which must not change during the transaction.*

## Choosing a Locking Level

Because TurboIMAGE needs more information to lock data entries than to lock the whole data base, program complexity tends to increase the lower the level of locks employed. Locking the whole data base or a single data set is the simplest operation, followed in increasing order of complexity by locking multiple data sets and locking data entries. At system design time, a compromise must be made between the benefits of low-level locking and the extra programming effort required.

Data entry locking should give the best performance; however, there are situations in which the extra programming effort for data entry locking is not worthwhile. Performance is least optimum at the higher level of the lock. Performance and programming effort should be considered, some other considerations that may effect your choice of locking level are discussed below.

### LOCKING AT THE SAME LEVEL

All programs concurrently accessing a data base should lock at the same level most of the time. For example, one process locking a data set will hold up all other processes that are attempting to lock entries in that set. Therefore, the attempt by the process locking at the data entry level to allow other processes to share the data base is nullified by the process locking at the data set level and the effect is as if all processes were locking at the data set level. The rule of locking at the same level may be violated for infrequent operations such as exception handling or rare transactions.



## LENGTH OF TRANSACTIONS

Generally, the longer the lock is to be held, the lower the level it should be. In other words, if you are performing lengthy transactions (more than about 8 TurboIMAGE calls), you should probably lock at the entry level. For transactions shorter than this, data base or data set locks will give approximately the same results.

An extreme case of a long transaction is one in which user dialog takes place while a lock is held. For example, a program may read some data entries, interact with a terminal operator, and modify some or all of the entries. A lock to cover this transaction may last several minutes which is an unacceptable amount of time to stop all data base or data set activity. In this situation, data entry level locking should be used.

Since the length of different transactions varies, the longest transaction (that is also frequently used) should guide the choice of locking level.

## LOCKING DURING USER DIALOG

In the situation described above, where a lock is held during interactive dialog with a terminal operator, the terminal timeout feature of MPE may be used to avoid having the locked entity inaccessible when the terminal operator is interrupted in the middle of the dialog. The timeout feature may be used to cause the terminal read to terminate automatically if no response is received within a certain time period. Refer to the discussion of "FCONTROL" in the *MPE Intrinsic Manuals*.

## Choosing an Item for Locking

An important convention to follow in designing a locking scheme is that all programs sharing the data base concurrently use the same data item to lock data entries in a particular data set. At any one time, TurboIMAGE allows no more than one data item per data set to be used for locking purposes. However, several values of the data item may be locked at the same time. For example, if one process has successfully locked SALES:ACCOUNT=54321000, another process may lock SALES:ACCOUNT=11111111. If a request is made to unconditionally lock SALES:STOCK#=8888X22R, the requesting process will be made to wait until all entries locked by ACCOUNT number are unlocked. Furthermore, any new requests for locking other SALES:ACCOUNT values will wait until SALES:STOCK#=8888X22R is successfully locked and unlocked again.

With this in mind, it is apparent that it is more efficient if all processes locking data entries in the SALES data set use the same data item since it is much less likely that one process will have to wait until another process finishes using the data. Therefore, at system design time, decide which item will be used in each data set for lock specification purposes. (It may be useful to add comments in the schema indicating which item is the locking item for each set.)

## Examples of Using the Locking Facility

The following examples list the order in which TurboIMAGE intrinsics may be called when using the locking facility while performing various transactions. The examples refer to the ORDERS data base described in Figures 2-5 and 2-6.

- Add a New Customer

1. DBLOCK the Customer data set or the whole data base.
2. DBPUT new data entry in CUSTOMER data set.
3. DBUNLOCK.

TurboIMAGE requires a data set or data base lock to cover addition of an entry to a master data set.

- Update an INVENTORY data entry to increase UNIT-COST for part 6650322S by 12 percent

1. DBLOCK INVENTORY: STOCK#=6650D22S. (Alternatively, the INVENTORY set or the whole data base can be locked.)
2. DBFIND and DBGET the data entry that is locked in step 1.
3. Compute new  $\text{UNIT-COST} = \text{UNIT-COST} + .12 * \text{UNIT-COST}$ .
4. DBUPDATE the data entry that is locked.
5. DBUNLOCK.

- Insert a new product with a new supplier

1. DBLOCK the PRODUCT master data set.
2. DBPUT new product data entry in PRODUCT master data set. (For example: 4444A33B CALIPER).
3. DBUNLOCK.
4. DBLOCK the SUP-MASTER data set.
5. DBPUT new supplier data entry in SUP-MASTER data set.
6. DBUNLOCK.
7. DBLOCK INVENTORY: STOCK# = 4444A33B.
8. DBPUT new data entry in INVENTORY data set for STOCK# = 4444A33B.
9. DBUNLOCK.

This has been done in three transactions. If the user did not want other users to see the data base with the supplier record present but with no inventory shown, one transaction with the data sets locked in two calls to DBLOCK with modes 3 or 4 can be performed.

- Interactively modify an order for customer account 89393899

1. DBLOCK SALES: ACCOUNT = 89393899.
2. DBFIND the CUSTOMER master data set entry with ACCOUNT = 89393899 in order to prepare to read the chain of SALES data entries with the same ACCOUNT value.
3. DBGET each entry in the chain and display it to user until the correct order is located.
4. Modify the contents of the data entry according to the user's request.
5. DBUNLOCK.

All data entries for ACCOUNT 89393899 in the SALES data set are locked. Note that these locks are held while a dialog takes place with the terminal operator, therefore, the lock may be held for several minutes. For this type of transaction, it may be best to first perform a conditional lock to determine if the records are accessible. For example:

1. DBLOCK SALES: ACCOUNT = 89393899 with mode 6.
2. If the lock does not succeed, the following message is displayed:  
RECORDS BEING MODIFIED. WANT TO WAIT?  
If the response is NO then go to other processing. If YES, call DBLOCK again with mode 5.

Table 4-3 contains guidelines that may be helpful in designing locking schemes for shared-access environments which include users who might modify the data base. Although data entry level locks are recommended in this table and illustrated in the examples above, data set or data base locks may be more appropriate for similar tasks depending upon other application requirements.

**Table 4-3. Locking in Shared-Access Environments**

ACTION	RECOMMENDED LOCKS
Chained DBGET calls	Lock all data entries in the chain. This usually requires one lock descriptor.
Serial DBGET calls	Lock the data set.
Update a data entry (DBUPDATE)	Lock the data entry before calling DBGET to read the data entry. Unlock after the update.
Directed reads (DBGET calls)	These are not recommended in a shared environment. Lock the data set before determining which data entry is needed.
Add a data entry to a detail data set (DBPUT)	Any lock which covers this data entry, but preferably uses the data item that was decided on as the "lock item" for the data set.
Add to or delete from a master data set (DBPUT and DBDELETE)	Lock the data set or data base. This is mandatory if the data base is open in access mode 1.

## Issuing Multiple Calls to DBLOCK

In order to guarantee that two processes cannot deadlock, once a call to DBLOCK is made by any process in a session or job, TurboIMAGE does not allow a second call to be made unless the locks are cancelled with a call to DBUNLOCK first. There are two exceptions to this rule:

- A redundant call may be made to lock the whole data base with DBLOCK mode 1 or 2 provided the call relates to the same access path. The redundant call will have no effect. (This is allowed in order to maintain compatibility with earlier versions of IMAGE.)
- More than one DBLOCK call may be made if the program from which multiple DBLOCK calls are issued has Multiple RIN (MR) capability. (A user cannot prepare such a program unless they also have this capability. Refer to the *System Manager/System Supervisor Reference Manual* for more information.)

The DBLOCK procedure is similar to MPE global RIN locks (no RINs are actually involved) in that it may put a process into a waiting state and thus, may cause a deadlock to occur. For example, a deadlock may occur if process A is waiting for a global RIN to be freed by process B, and process B is waiting for a data base entity to be unlocked by process A. Therefore, issuing a DBLOCK in conjunction with a lock applied by MPE intrinsics such as LOCKGLORIN or FLOCK or by the COBOLLOCK procedure requires MR capability. (The use of MR capability is not recommended unless absolutely necessary.)

Users whose programs have MR capability and issue multiple DBLOCK calls are responsible for deadlock prevention. This type of locking must be done very carefully. Recovery from a deadlock requires a restart of the operating system.

No matter how many descriptors are listed in a single DBLOCK call, TurboIMAGE guarantees that deadlocks will never occur provided that no executing program that accesses the data base has MR capability. Programs that execute successfully using TurboIMAGE locks in a single process environment will not execute in a process-handling environment without MR (Multiple RIN) capability. (Refer to Appendix D for more information on the MR capability.)

## Releasing Locks

The locks held by a process for a particular access path of a data base are relinquished when the process calls DBUNLOCK, or automatically when the process closes the data base, terminates, aborts, or is aborted by an operator.

Failure of a program to release locks will result in other programs waiting indefinitely for any conflicting locks. These programs, while in a waiting state, cannot be aborted by the operating system. An attempt to abort such a waiting process will result in the abort taking effect as soon as the process obtains the lock for which it was waiting.

### NOTE

Any program that executes a DBGET in modes 5 or 6 should lock the data base. This will prevent the execution of any DBPUTs or DBDELETEDs in the detail data set and will help prevent broken chains. The program should also lock the chain in a detail set to prevent the next or previous chain in a detail set from being DBDELETED.

## USING THE LOGGING FACILITY

TurboIMAGE has the capability of recovering a data base from a transaction-oriented logfile in the event of a system failure. Transaction logging and recovery is fully discussed in Section 7; however, some considerations relevant to applications are discussed here.

### What Logging Does

The TurboIMAGE logging and recovery facility enables all data base modifications to be logged automatically to a tape or disc logfile. In the event of a system failure the logfile is read to re-execute transactions or identify incomplete transactions, depending on what type of recovery process is being used. In addition, the transaction logging system can be a useful tool for auditing. The logfile is actually a record of all modifications to the data base. The intrinsic DBMEMO, capable of logging user text, facilitates interpretation of the logfiles for future reference.

The data base administrator is responsible for enabling or disabling the logging and recovery processes and generating backup data base copies, thus making logging a global function controlled at the data base level rather than at the individual user level.

A process is said to be logging if all of the following are true:

- The data base has been enabled for logging by the data base administrator.
- A logging process has been initiated from the system console.
- The user is accessing the data base in one of modes 1 through 4.

### How Logging Works

The following TurboIMAGE intrinsics are automatically logged when the data base is enabled for logging and a user opens the data base in a mode which permits modifications: DBOPEN, DBCLOSE, DBPUT, DBUPDATE, DBDELETE, DBBEGIN, DBEND, and DBMEMO.

TurboIMAGE calls the MPE logging intrinsics OPENLOG, WRITELOG, and CLOSELOG in order to log information to the logfile. When a data base is opened, DBOPEN calls the OPENLOG intrinsic using the log identifier and password stored in the data base root file. If this call succeeds, DBOPEN calls WRITELOG to log a DBOPEN log record containing information about the data base and the new user.

The WRITELOG intrinsic is also used to log information when the TurboIMAGE intrinsics DBPUT, DBDELETE, and DBUPDATE are called. WRITELOG is called after all error checks are made, but before actually modifying the working data base. Consequently, a log record is not written until the TurboIMAGE procedure has committed itself to succeed. WRITELOG is also used by the TurboIMAGE intrinsics DBBEGIN, DBEND, and DBMEMO.

DBCLOSE (mode 1) calls WRITELOG to log out a DBCLOSE log record, and then calls CLOSELOG to terminate access to the logfile. If a transaction initiated with DBBEGIN fails to call DBEND, DBCLOSE causes a special DBEND log record to terminate access to the logfile. DBCLOSE also causes a special DBEND log record to be written if the program is aborting with a transaction unfinished.

## Logging and Logical Transactions

A transaction can be considered as the basic work unit performed against a data base. A transaction could consist of a single modification, but more typically might consist of several calls to TurboIMAGE intrinsics which lock, read, modify, and unlock information. Logical transactions transfer the data base from one consistent state to another, but in the midst of a multiple-step transaction, the data base could be temporarily inconsistent with itself. (For an example, see Section 7.)

In the event of a system failure and subsequent recovery, only complete logical transactions are re-executed, returning the data base to a consistent state. Therefore, it is essential that an application program mark the beginning, and end of a sequence of calls which constitute a single logical transaction with the intrinsics DBBEGIN and DBEND.

For reasons explained more fully under "Logical Transactions and Locking" in Section 7, the following sequence of operations should be followed as closely as possible when performing modifications:

1. Call DBLOCK to lock all data which must not change during the transaction. This includes data to be read and data to be modified.
2. Read data using DBFIND and DBGET to determine the necessary modifications.
3. Call DBBEGIN to declare the beginning of modifications.
4. Make modifications using DBPUT, DBDELETE, or DBUPDATE.
5. Call DBEND to declare the end of the modifications.
6. Call DBUNLOCK to release all of the locks.

## Transaction Numbers

TurboIMAGE maintains a double word transaction number for each user's access to the data base. Transaction numbers enable the DBRECOV recovery program to associate log records with a particular transaction. This number is initialized by DBOPEN and incremented each time DBBEGIN is called, or for each single call to DBPUT, DBUPDATE, or DBDELETE if it is not included in a transaction delimited by DBBEGIN and DBEND. Transaction numbers are included in all DBBEGIN, DBPUT, DBUPDATE, DBDELETE, DBMEMO, and DBEND log records. The transaction number is always incremented as described, regardless of whether the user process is actually logging. A user process may determine its transaction count (and whether the data base and user is logging) by calling DBINFO using mode 401.

## Logging and Process Suspension

The MPE logging intrinsics will suspend a calling process if the logging buffers become full. Consequently, a user process which calls TurboIMAGE may become suspended, for example, if a tape logfile reaches the end of a reel and logging buffers become full before a new tape can be mounted.

## OBTAINING DATA BASE STRUCTURE INFORMATION

The DBINFO library procedure allows you to acquire information programmatically about the data base. It provides information about data items, data sets, or data paths. The information returned is restricted by the user class number and access mode established when the data base is opened.

Any data items, data sets, or paths of the data base inaccessible to that user class or in that access mode are considered to be non-existent. For example, if the access mode grants only read access, this procedure will indicate that no data sets may have entries added. The information that can be obtained through separate calls to DBINFO is summarized below.

In relation to data items, DBINFO can be used:

- To determine whether the user class number established when the data base is opened allows a specified data item value to be changed in at least one data set, or allows a data entry containing the item to be added or deleted.
- To get a description of a data item including the data item name, type, sub-item length, and sub-item count. This information corresponds to that which is specified in the item part of the schema.
- To determine the number of items in the data base available to the current user and to get a list of numbers identifying those items. The numbers indicate the position of each data item in the item part of the schema. The type of access, for example read-only, can also be determined.
- To determine the number of items in a particular data set available to the current user and get a list of those item numbers and the type of access available for each one.

In relation to data sets, DBINFO can be used:

- To determine whether the current user can add or delete entries to a particular data set.
- To get a data set description including the data set name, type, length in words and blocking factor for data entries in the set, number of entries in the set, and the capacity.
- To determine the number of data sets the current user can access and get a list of the data set numbers indicating the position of the data set definition in the set part of the schema. The type of access to each set is also indicated.
- To determine in which data sets a particular data item is available to the current user. The number of data sets, a list of data set numbers, and the type of access available for each set is returned.

In relation to paths, DBINFO can be used:

- To get information about the paths associated with a particular data set including the number of paths. If the data set is a master set, the information includes the data set number, search item number, and sort item number for each related detail. If the data set is a detail set, the information includes the master data set number of the related master data set, the detail search item number and sort item number for each path.
- To determine the search item number of a master data set or the search item number for the primary path of the detail and the data set number of the related master. In either case, if the search item is inaccessible to the current user, no information is returned.

## Special Uses of DBINFO

If the application program uses data items and data set numbers when calling the other TurboIMAGE procedures, it is good practice to determine these numbers by calling DBINFO at the beginning of the program to set up the numbers. It is not practical to code the numbers into the program since a change to the data base structure might require extensive changes to the application programs. Likewise, it is inefficient and time consuming to call DBINFO throughout the program to determine these numbers. Many application programmers prefer the convenience and flexibility of using the data item and data set names in procedure calls.

DBINFO is useful when writing general inquiry applications similar to the QUERY data base inquiry facility. DBINFO may also be used to obtain information regarding the logging facility. In relation to Native Language Support (NLS), DBINFO can be used to get the MPE number code that defines the native language supported by the data base. (Refer to "Data Base Description Language" and "Schema Structure", in Section 3.)

## Checking Subsystem Flag

A subsystem flag can be set by the DBUTIL program's >>SET command. This flag indicates whether subsystems, including user programs, can access the TurboIMAGE data base and, if access is allowed, whether it is read only or both read and write. Because the flag does not actually allow or prevent access, the subsystem or user program must include a call to DBINFO to test this flag.

## CLOSING THE DATA BASE OR A DATA SET

After you have completed all the tasks you want to perform with the data base, you use the DBCLOSE library procedure to terminate access to it. When DBCLOSE is used for this purpose, all data set files and the root file are closed and the data segment containing the DBU is released to the MPE system. If there are no other concurrent users of the data base, the extra data segments containing the DBB and DBG are also released. All locks that you still have on the data base through the closed access path are automatically released.

The DBCLOSE procedure can also be used to rewind or close, access to a data set. Rewinding consists of resetting the dynamic status information kept by TurboIMAGE to its initial state. If a detail data set is closed or rewound, the current path does not change when the status information is initialized.

The purpose of closing a data set completely is to return the resources required by that data set to the MPE system without terminating access to the data base. A typical reason for rewinding a data set is to start at the first, or last, entry again when doing a forward or backward serial read.

It is important to close the data base before terminating programs operating under control of the BASIC Interpreter since termination of your BASIC program does not coincide with termination of the BASIC Interpreter process.



## CHECKING THE STATUS OF A PROCEDURE

Each time a procedure is called, TurboIMAGE returns status information in a buffer specified by the calling program and sets the condition code maintained by MPE in the status register. The condition code, or the condition word (described later), should be checked immediately after TurboIMAGE returns from the procedure to the calling program.

A condition code is always one of the following and has the general meaning shown:

Condition Code	General Meaning
CCE	The procedure performed successfully. No exceptional condition was encountered.
CCG	An exceptional condition, other than an error, was encountered.
CCL	The procedure failed due to an invalid parameter or a system error.

The first word of the status information returned in the calling program's buffer is a *condition word* whose value corresponds to the condition code as follows:

Condition Code	Condition Word Value
CCE	0
CCG	>0
CCL	<0

The calling program must check either the condition code or the condition word to determine the success or failure of the procedure. The condition word is also used to indicate various exceptional conditions and errors. These are summarized in Appendix A.

The other words of status information vary with the outcome of the call and from one procedure to another. The content of these words is described in detail with each procedure definition later in this section and in Appendix A, which describes error conditions.

## **INTERPRETING ERRORS**

TurboIMAGE provides two library procedures, DBEXPLAIN and DBERROR, which can be used to interpret status information programmatically. DBEXPLAIN prints on the \$STDLIST device an English language error message which includes the name of the data base and the name of the procedure that returned the status information. DBERROR performs a similar function but returns the information in a buffer specified by the calling program.

These procedures are intended primarily for use in debugging application programs rather than in interpreting errors in the production environment where more specific application messages are necessary.

## **ABNORMAL TERMINATION**

Under certain conditions, the calling process may be terminated by TurboIMAGE. Conditions giving rise to process termination and a description of the accompanying error messages are presented in Appendix A.

## USING TurboIMAGE LIBRARY PROCEDURES

This section contains the reference specifications for the TurboIMAGE procedures, arranged alphabetically. Table 5-1 gives a summary of the procedures with a brief description of their function in logical order.

On the following pages, the calling parameters for each procedure are defined in the order in which they appear in the call statement. Each parameter must be included when a call is made since a parameter's meaning is determined by its position.

Table 5-1. TurboIMAGE Procedures

PROCEDURE	FUNCTION
DBOPEN	Initiates access to a data base. Sets up user's access mode and user class number for the duration of the process.
DBLOCK	Locks one or more data entries, a data set, or an entire data base (or a combination of these) temporarily to allow the process calling the procedure to have exclusive access to the locked entities.
DBFIND	Locates the first and last entries of a data chain in preparation for access to entries in the chain.
DBGET	Reads the data items of a specified entry.
DBBEGIN	When logging, designates the beginning of a transaction and optionally writes user information to the logfile.
DBMEMO	When logging, writes user information to the logfile.
DBPUT	Adds new entries to a data set.
DBUPDATE	Updates or modifies the values of data items that are not search or sort items.
DBDELETE	Deletes existing entries from a data set.
DBEND	When logging, designates the end of a transaction and optionally writes user information to the logfile.
DBUNLOCK	Releases those locks obtained with previous calls to DBLOCK.
DBCLOSE	Terminates access to a data base or a data set, or resets the pointers of a data set to their original state.
DBINFO	Provides information about the data base being accessed, such as the name and description of a data item.
DBEXPLAIN	Examines status information returned by an TurboIMAGE procedure that has been called and prints a multi-line message on the \$STDLIST device.
DBERROR	Supplies an English language message that interprets the status information set by any callable TurboIMAGE procedure. The message is returned to the calling program in a buffer.
DBCONTROL	Allows process operating in exclusive mode to enable or disable the "deferred update" option.

Table 5-2 illustrates the forms of the call statements for the four languages that can be used to call the procedures. Section 6 contains examples of using the procedures and specifications for declaration of parameters for each language. It also provides a sample RPG program.

**Table 5-2. Calling a TurboIMAGE Procedure**

COBOL	CALL "name" USING <i>parameter,parameter,...,parameter</i>
FORTRAN	CALL <i>name (parameter,parameter,...,parameter)</i>
SPL	<i>name (parameter,parameter,...,parameter)</i>
BASIC	<i>linenumber</i> CALL <i>name (parameter,parameter,...,parameter)</i>
PASCAL	<i>name (parameter,parameter,...,parameter)</i>

All procedures may be called directly from programs in any of the five host languages. However, when using BASIC it is recommended to use the BIMAGE interface procedures. (Refer to Section 6 for more information.) Since they are not TYPE procedures, they do not use the SPL OPTION VARIABLE capability, and all parameters are call-by-reference word pointers.

## Intrinsic Numbers

An intrinsic number is provided for each procedure except DBEXPLAIN and DBERROR. This number, which uniquely identifies the procedure within TurboIMAGE and the MPE operating system, is returned with other status information when an error occurs. You can use it to identify the procedure that caused the error.

## Data Base Protection

When each procedure is called, TurboIMAGE verifies that the requested operation is compatible with the user class number and access mode established when the data base is opened.

## Unused Parameters

When calling some procedures for a specific purpose, one of the parameters may be ignored, however, it must be listed in the call statement. An application program may find it useful to set up a variable named DUMMY to be listed as the unused parameter in these situations, as a reminder that the value of the parameter does not effect the procedure call.

## The Status Array

The status array is a communication area. If the procedure executes successfully, the contents reflect this as described in this section. If the procedure fails, standard error information is returned as described in Appendix A.

# DBBEGIN

## INTRINSIC NUMBER 412

Designates the beginning of a sequence of TurboIMAGE procedure calls which are to be regarded as a single logical transaction for the purposes of logging and recovery. Text area may be used to log user information to the logfile.

### Syntax

DBBEGIN,*base*,*text*,*mode*,*status*,*textlen*

### Parameters

- base*is the name of the array used as the base parameter when opening the data base. The first word of the array must contain the *base id* returned by DBOPEN. (Refer to DBOPEN for more information about the *base id*.)
- text*is an array up to 256 words long which contains user ASCII or binary data to be written to the logfile as part of the DBBEGIN log record. The *text* argument is used to assign each particular transaction a distinct name. (Refer to "Discussion" below for more information.)
- mode*must be an integer equal to 1.
- status*is the name of a ten-word array in which TurboIMAGE returns status information. If the procedure executes successfully, the status array contents are:

Word	Contents
1	Condition word is 0.
2-4	Unchanged from previous procedure call using this array.
5-10	Procedure call information. Refer to Appendix A for a description of this information.

Table 5-3 lists the contents of Word 1 when the procedure does not succeed.

*textlen*is an integer equal to the number of words to be logged in the *text* parameter, or if negative, equal to the number of bytes. Length may also be zero.

## Discussion

DBBEGIN is called to designate the beginning of a sequence of TurboIMAGE procedure calls which are jointly considered as a single logical transaction. (The end of such a sequence is designated by a matching call to DBEND.) If the calling process is logging, DBBEGIN causes a log record to be written to the logfile which includes such information as the time, date, and user text buffer. DBBEGIN log records are used by the data base recovery program DBRECOV to identify the beginning of logical transactions.

If TurboIMAGE Profiler is used to gain information on the effectiveness of program calls, the text argument is used to identify the name of each logical transaction. If text is left blank, Profiler assigns the program name to the logical transaction. To gain the greatest use of Profiler define each logical transaction with a name in the text argument of DBBEGIN. (Refer to the *TurboIMAGE Profiler User Guide* for further information.)

DBBEGIN will return an error condition if it is called twice without an intervening call to DBEND, whether the process is actually logging or not.

**Table 5-3. DBBEGIN Condition Word Values**

<b>CALLING ERRORS:</b>	-11	Bad data base reference.
	-31	Bad mode.
	-151	Text length too large.
	-152	Transaction already in progress.
<b>COMMUNICATIONS FAILURES:</b>	-102	DSWRITE failure.
	-106	Remote data inconsistent.
	-107	DS procedure call error.
<b>EXCEPTIONAL CONDITIONS:</b>	0	Logging not enabled for this user.
	62	DBCBC cannot expand.
	63	Bad DBG.
<b>LOG SYSTEM FAILURES:</b>	-111	WRITELOG intrinsic failure.
		Consult Appendix A for more information about these condition codes.

# DBCLOSE

## INTRINSIC NUMBER 403

Terminates access to a data base or terminates, temporarily or permanently, access to a data set, or rewinds a data set.

## Syntax

DBCLOSE, *base*, *dset*, *mode*, *status*

## Parameters

*base* is the name of an array used as the *base* parameter when opening the data base. The first word of the array must contain the *base id* returned by DBOPEN. (Refer to DBOPEN for more information about the *base id*.)

*dset* is the name of an array containing the left-justified name of the data set to be closed or is an integer referencing the data set by number if *mode* equals 2 or 3. If *mode* equals 1, this parameter is ignored.

The data set name may be 16 characters long or, if shorter, terminated by a semicolon or blank.

*mode* is an integer indicating the type of termination desired. If *mode* equals 1, access to the data base is terminated. Any locks through the closed access path are released.

If *mode* equals 2, the data set referenced by the *dset* array is closed, but locks held in the data set are not released.

If *mode* equals 3, the data set referenced by the *dset* array is reinitialized but not closed.

*status* is the name of a ten-word array in which TurboIMAGE returns status information about the procedure. If the procedure executes successfully, the status array contents are:

Word	Contents
1	Condition word is 0.
2-4	Unchanged from previous procedure call using this array.
5-10	Procedure call information. Refer to Appendix A for a description of this information.

Table 5-5 lists the contents of Word 1 when the procedure does not succeed.



## Discussion

You must call DBCLOSE mode 1 to terminate access to the data base when you have completed all the tasks you want to perform. If a process has issued multiple calls to DBOPEN for the same data base, only the access path specified in the DBCLOSE base parameter is affected by the call to DBCLOSE.

The capability to reset and close a data set is provided to perform functions such as reinitializing dynamic status information for a process accessing a particular data set and returning system resources. In both modes 2 and 3, status information is reinitialized, but system resources are returned in mode 2 only. Table 5-4 summarizes the functions performed in each mode.

**Table 5-4. DBLCLOSE Modes 2 and 3**

FUNCTION	MODE 2	MODE 3
Reinitialize dynamic status information for the data set:		
such as the chain count, forward and backward pointers, current record number and last condition word	YES	YES
quiesce the data set in addition	YES	NO
Close the data set and return system resources	YES	NO
Release locks held within the data set	NO	NO

Since mode 3 does not close and re-open a data set, it is more efficient than mode 2 if the data set is to be accessed again before the data base is closed.

If the process is logging, a mode 1 DBCLOSE will cause a DBCLOSE log record to be written to the logfile. DBCLOSE log records contain such information as the time, date, and user log identification number. A DBCLOSE log record is also written if the process aborts or terminates without closing the data base. If the process aborts before completing an active transaction, a special DBEND log record is written prior to the DBCLOSE.

DBCLOSE will return an error condition if the process has not completed an active transaction, in other words, has called DBBEGIN without a matching call to DBEND. Transactions which abort in this manner are not automatically suppressed by DBRECOV during recovery in order to salvage as many subsequent transactions that may depend on the aborted transaction as possible.

## DBCLOSE

Table 5-5. DBCLOSE Condition Word Values

FILE SYSTEM AND MEMORY MANAGEMENT FAILURES:	-2	FCLOSE failure.
CALLING ERRORS:	-11 -21 -31	Bad <i>bases</i> parameter. Bad data set reference. Bad <i>mode</i> .
COMMUNICATIONS ERRORS:	-101 -102 -106 -107	DSCLOSE failure. DSWRITE failure. Remote data inconsistent. DS procedure call error.
LOGGING SYSTEM FAILURES:	-111 -112 -152	WRITELOG intrinsic failure. CLOSELOG intrinsic failure. Transaction is in progress.
ILR LOG FILE ERROR:	-171	Cannot close ILR log file: file system error <i>nn</i> .
EXCEPTIONAL CONDITIONS:	63	Bad DBG.
		Consult Appendix A for more information about these condition codes.

Allows a process accessing the data base in exclusive mode (DBOPEN mode 3) to enable or disable the "output deferred" option.

### Syntax

`DBCONTROL,base,qualifier,mode,status`

### Parameters

*base* is the name of the array used as the base parameter when opening the data base. The first word of the array must contain the *base id* returned by DBOPEN. (Refer to DBOPEN for more information about the *base id*.)

*qualifier* is currently ignored by DBCONTROL.

*mode* must be an integer equal to 1 or 2, indicating the following:

- Mode 1: Turn on output deferred option (see note below).
- Mode 2: Turn off output deferred option (see note below).

*status* is the name of a ten-word array in which TurboIMAGE returns status information. If the procedure executes successfully, the status array contents are as follows:

Word	Contents
1	Condition word is 0.
2-4	Unchanged from previous procedure call using this array.
5-10	Procedure call information. Refer to Appendix A for a description of this information.

Table 5-6 lists the contents of Word 1 when the procedure does not succeed.

# DBCONTROL

## Discussion

In the default mode of operation, TurboIMAGE writes all data base modifications (calls to DBPUT, DBUPDATE, and DBDELETE) to the disc before returning to the calling program. In the output deferred mode of operation, however, TurboIMAGE will only write out modifications when necessary to free a TurboIMAGE buffer for further use.

### NOTE

If AUTODEFER is enabled on the data base (using DBUTIL >> ENABLE command), then modes 1 or 2 are overridden by the **automatic** output deferred option. AUTODEFER is disabled only by using the DBUTIL >>DISABLE command.

A program which opens the data base exclusively may call DBCONTROL (mode 1) to enter the deferred mode of operation. In deferred mode, data base modifications caused by calls to DBPUT, DBUPDATE, or DBDELETE may not be written to the disc (or may only be partially written) upon return from these procedures. Although TurboIMAGE generally operates more efficiently in this mode, a system failure while the data base is operating in this mode has a very high probability of causing internal structural damage to the data base.

A call to DBCONTROL (mode 2) will turn off the deferred mode of operation and will write the contents of all modified buffers to disc.

Table 5-6. DBCONTROL Condition Word Values

FILE SYSTEM FAILURES:	-4	FREADLABEL failure.
CALLING ERRORS:	-11 -14 -31 -80	Bad data base reference. Illegal intrinsic in current access mode. Bad mode. Output deferred not allowed when ILR enabled.
COMMUNICATIONS FAILURES:	-102	DSWRITE failure.
EXCEPTIONAL CONDITIONS:	63	Bad DBG.
		Consult Appendix A for more information about these condition codes.

Deletes the current entry from a manual master or detail data set. The data base must be open in access mode 1, 3, or 4.

### Syntax

DBDELETE, *base*, *dset*, *mode*, *status*

### Parameters

- base* is the name of the array used as the *base* parameter when opening the data base. The first word of the array must contain the *base id* returned by DBOPEN. (Refer to DBOPEN for more information about the *base id*.)
- dset* is the name of an array containing the left-justified name of the data set from which the entry is to be deleted or is an integer referencing the data set by number. The data set name may be 16 characters long or, if shorter, terminated by a semicolon or a blank.
- mode* must be an integer equal to 1.
- status* is the name of a ten-word array in which TurboIMAGE returns status information about the procedure. If the procedure executes successfully, the status array contents are:

Word	Contents
1	Condition word is 0.
2	Zero.
3-4	Unchanged current record address.
5-6	Number of entries in a chain.  If master data set, the number is zero unless the deleted entry was a primary entry with synonyms. In this case, the number is one less than its previous value.  If detail data set, the number is unchanged from the preceding procedure call.
7-10	Unchanged preceding and succeeding record numbers of a chain. If master data set and the new synonym chain count is greater than zero, the numbers reference the last and first synonym chain entries respectively.

# DBDELETE

## Discussion

When deleting entries from master data sets, the following rules apply:

- All pointer information for chains indexed by the entry must indicate that the chains are empty. In other words, there must not be any detail entries on the paths defined by the master which have the same search item value as the master entry to be deleted.
- If the data base is open in access mode 1, a lock must be in effect on the data set or the whole data base.

Because of the way TurboIMAGE handles synonym chains, it is possible to write a routine to read and delete all the entries in a master data set and still leave some entries in the set. If the deleted entry is a primary with synonyms, TurboIMAGE writes the first synonym in the chain to the deleted primary's location. A subsequent DBGET will read the next sequential entry, leaving an entry (the new primary) in the previous location.

A solution to this problem is to check words 5 and 6 of the status parameter following each DBDELETE call. If the synonym count in these words is not zero, reread the location (using DBGET, mode 1) and call DBDELETE again. Repeat the reread and DBDELETE until the count is zero, then continue reading and deleting serially. (Refer to Section 4 for a discussion of serial access and to Section 10 for a discussion of synonym chains.)

TurboIMAGE performs the required changes to chain linkages and other chain information, including the chain heads in related master data sets. If the last member of each detail chain linked to the same automatic master entry has been deleted, DBDELETE also deletes the master entry containing the chain heads - in this case, the synonym chain information for the automatic master is set to zero (refer to Section 10 for more information).

If the data base is open in access mode 1, you must establish a lock covering the data entry to be deleted before calling DBDELETE.

The current record is unchanged. If a primary data entry with synonyms is deleted from a master data set and a secondary migrates, the backward and forward pointers reflect the new primary. In all other cases, the backward and forward pointers are unchanged when an entry is deleted.

If the process is logging, a call to DBDELETE will cause a log record to be written, which includes such information as the time, date, user identification number, and a copy of the record to be deleted.

Table 5-7. DBDELETE Condition Word Values

FILE SYSTEM AND MEMORY MANAGEMENT FAILURES:	-1 -3 -4	FOPEN intrinsic failure. FREADDIR failure. FREADLABEL failure.
CALLING ERRORS:	-11 -12  -14 -21 -23 -31	Bad <i>base</i> parameter. No lock covers the data entry to be deleted. (Occurs only if open in access mode 1.) Illegal intrinsic in current access mode. Bad data set reference. Data set not writable. Bad <i>mode</i> .
COMMUNICATIONS ERRORS:	-102 -106 -107	DSWRITE failure. Remote data inconsistent. DS procedure call error.
LOGGING SYSTEM FAILURES:	-111	WRITELOG intrinsic failure.
EXCEPTIONAL CONDITIONS:	17 44 63	No entry. Chain head. Bad DBG.
		Consult Appendix A for more information about these condition codes.

# DBEND

## INTRINSIC NUMBER 413

Designates the end of a sequence of TurboIMAGE procedure calls which are regarded as a single logical transaction, for the purposes of logging and recovery. Text area may be used to log user information to the logfile.

## Syntax

DBEND,*base,text,mode,status,textlen*

## Parameters

*base* is the name of the array used as the base parameter when opening the data base. The first word of the array must contain the *base id* returned by DBOPEN. (Refer to DBOPEN for more information about the *base id*.)

*text* is an array up to 256 words long which contains user ASCII or binary data to be written to the logfile as part of the DBEND log record.

*mode* must be an integer equal to 1 or 2.

Mode 1: End of logical transaction.

Mode 2: End logical transaction and write contents of the logging buffer in memory to disc.

*status* is the name of a ten-word array in which TurboIMAGE returns status information about the procedure. If the procedure executes successfully, the status array contents are as follows:

Word	Contents
1	Condition word is 0.
2-4	Unchanged from previous procedure call using this array.
5-10	Procedure call information. Refer to Appendix A for a description of this information.

Table 5-8 lists the contents of Word 1 when the procedure does not succeed.

*textlen* is an integer equal to the number of words to be logged in the TEXT parameter, or if negative, equal to the number of bytes. Length may be zero.



## Discussion

DBEND is called to designate the end of a sequence of TurboIMAGE procedure calls which are collectively considered as a single logical transaction. (The beginning of such a sequence is designated by a previous call to DBBEGIN.) If the process is logging, DBEND causes a log record to be written to the logfile which includes such information as the time, date, and user text buffer. DBEND log records are used by the data base recovery program DBRECOV to identify the end of logical transactions. Failure to call DBEND will not cause a transaction to be suppressed in the event of a program abort and subsequent data base recovery.

If you call DBEND with mode equal to 2, DBEND will force the writing of the log buffer from memory to disc before returning to the calling process. This flush of the log buffer occurs after the intrinsic has logged the end of the logical transaction. Try to use this option only for critical transactions; too many mode 2 DBEND calls can degrade performance by causing a disc access each time a logical transaction ends.

DBEND will return an error condition if it is called without a prior matching call to DBBEGIN, whether the process is actually logging or not.

**Table 5-8. DBEND Condition Word Values**

CALLING ERRORS:	-11	Bad data base reference.
	-31	Bad mode.
	-151	Text length too large.
	-153	No transaction in progress to end.
COMMUNICATIONS FAILURES:	-102	DSWRITE failure.
	-106	Remote data inconsistent.
	-107	DS procedure call error.
EXCEPTIONAL CONDITIONS:	0	Logging not enabled for this user.
	62	DBG cannot expand.
	63	Bad DBG.
LOG SYSTEM FAILURES:	-111	WRITELOG intrinsic failure.
	-113	FLUSHLOG returned error number <i>nn</i> to DBEND.
		Consult Appendix A for more information about these condition codes.

# DBERROR

Moves an English language message, as an ASCII character string, to a buffer specified by the calling program. The message interprets the contents of the status array as set by a call to a TurboIMAGE procedure.

## Syntax

`DBERROR, status, buffer, length`

## Parameters

<i>status</i>	is the name of the array used as the <i>status</i> parameter in the TurboIMAGE procedure call about which information is requested.
<i>buffer</i>	is the name of an array in the calling program's data area, at least 36 words long, to which the message is returned.
<i>length</i>	is an integer variable which is set by DBERROR to the positive byte length of the message placed in the <i>buffer</i> array. The length will never exceed 72 characters.

## Discussion

Like DBEXPLAIN, DBERROR messages are intended and appropriate for use while debugging application programs. The errors they describe are, for the most part, errors that do not occur in a debugged and running program.

Some errors or exceptional conditions are expected to occur, even in a production environment. For example, the MPE intrinsic DBOPEN may fail due to concurrent data base access. In this case, printing the DBERROR message:

DATA BASE OPEN EXCLUSIVELY

may be perfectly acceptable, even to the person using the application program. However, in many cases a specific message produced by the application program is preferable to the one produced by DBERROR. A DBFIND error generated by the application program, such as:

THERE ARE NO ORDERS FOR THAT PART NUMBER

would be more meaningful to a user entering data at a terminal than the DBERROR message:

THERE IS NO CHAIN FOR THE SPECIFIED SEARCH ITEM VALUE

Table 5-9 lists all messages that can be returned by DBERROR with their corresponding condition word values. Several messages may correspond to one condition word and the interpretation of the code depends on the context in which it is returned. Variable information is represented by a lowercase word or phrase.

Table 5-9. DBERROR Messages

CONDITION WORD	DBERROR MESSAGE
0	SUCCESSFUL EXECUTION - NO ERROR
-1	NO SUCH DATA BASE DATA BASE OPEN IN AN INCOMPATIBLE MODE BAD ACCOUNT REFERENCE or BAD GROUP REFERENCE BAD ROOT FILE REFERENCE VIRTUAL MEMORY NOT SUFFICIENT TO OPEN ROOT FILE DATA BASE ALREADY OPEN FOR MORE THAN READ DATA BASE IN USE DATA BASE OPEN EXCLUSIVELY MPE SECURITY VIOLATION MPE FILE ERROR <i>decimal integer</i> RETURNED BY FOPEN ON {ROOT FILE DATA SET # <i>decimal integer</i> }
-2	MPE FILE ERROR <i>decimal integer</i> RETURNED BY FCLOSE ON {ROOT FILE DATA SET # <i>decimal integer</i> }
-3	MPE FILE ERROR <i>decimal integer</i> RETURNED BY FREADDIR ON {ROOT FILE DATA SET # <i>decimal integer</i> }
-4	MPE FILE ERROR <i>decimal integer</i> RETURNED BY FREADLABEL ON {ROOT FILE DATA SET # <i>decimal integer</i> }
-5	MPE FILE ERROR <i>decimal integer</i> RETURNED BY FWRITEDIR
-6	MPE FILE ERROR <i>decimal integer</i> RETURNED BY FWRITELABEL
-7	PREVIOUS MPE FILE ERROR <i>decimal integer</i> FOUND IN DESIRED BUFFER
-9	MPE ERROR <i>%octal integer</i> RETURNED BY GETDSEG OF <i>decimal integer</i> WORDS
-11	BAD DATA BASE NAME OR PRECEDING BLANKS MISSING BAD DATA BASE REFERENCE (FIRST 2 CHARACTERS)
-12	IMAGE <i>procedure name</i> CALLED WITHOUT COVERING LOCK IN EFFECT
-14	CALLS TO TurboIMAGE <i>procedure name</i> NOT ALLOWED IN ACCESS MODE <i>decimal integer</i>

# DBERROR

Table 5-9. DBERROR Messages (Continued)

CONDITION WORD	DBERROR MESSAGE
-21	BAD PASSWORD - GRANTS ACCESS TO NOTHING DATA ITEM NONEXISTENT OR INACCESSIBLE SPECIFIED SET IS NOT ALLOWED ON A MASTER SET DATA SET NONEXISTENT OR INACCESSIBLE BAD MAINTENANCE WORD (CONTAINS COMMA OR DOES NOT MATCH)
-22	MAINTENANCE WORD REQUIRED
-23	USER (CLASS) LACKS WRITE ACCESS TO DATA SET
-24	OPERATION NOT ALLOWED ON AUTOMATIC MASTER DATA SET
-31	DBGET MODE <i>decimal integer</i> ILLEGAL FOR DETAIL DATA SET DBGET MODE <i>decimal integer</i> BAD--SPECIFIED DATA SET LACKS CHAINS BAD (UNRECOGNIZED) TurboIMAGE <i>procedure name</i> MODE: <i>decimal integer</i>
-32	UNOBTAINABLE ACCESS MODE: AOPTIONS REQUESTED: <i>%octal integer</i> , GRANTED: <i>%octal integer</i>
-51	LIST TOO LONG OR NOT PROPERLY TERMINATED
-52	ITEM SPECIFIED IS NOT AN ACCESSIBLE SEARCH ITEM IN THE SPECIFIED SET BAD LIST - CONTAINS ILLEGAL OR DUPLICATED DATA ITEM REFERENCE
-53	DBPUT LIST IS MISSING A SEARCH OR SORT ITEM
-60	ILLEGAL FILE EQUATION ON ROOT FILE
-80	OUTPUT DEFERRED NOT ALLOWED WITH ILR
-90	ROOT FILE BAD: UNRECOGNIZED STATE: <i>%octal integer</i>
-91	ROOT FILE (DATA BASE) NOT COMPATIBLE WITH CURRENT TurboIMAGE INTRINSICS
-92	DATA BASE REQUIRES CREATION (VIRGIN ROOT FILE)
-94	DATA BASE BAD: WAS BEING MODIFIED WITH OUTPUT DEFERRED, MAY NOT BE ACCESSED IN MODE <i>decimal integer</i>
-95	DATA BASE BAD: CREATION WAS IN PROCESS (CREATE AGAIN)
-96	DATA BASE BAD: ERASE WAS IN PROCESS (ERASE AGAIN)

Table 5-9. DBERROR Messages (Continued)

CONDITION WORD	DBERROR MESSAGE
-97	DATA BASE BAD: ILR ENABLE IN PROCESS
-98	DATA BASE BAD: ILR DISABLE IN PROCESS
-100	MPE ERROR <i>decimal integer</i> RETURNED BY DSOPEN
-101	MPE ERROR <i>decimal integer</i> RETURNED BY DSCLOSE
-102	MPE ERROR <i>decimal integer</i> RETURNED BY DSWRITE
-103	REMOTE 3000 STACK SPACE INSUFFICIENT
-104	REMOTE 3000 DOES NOT SUPPORT TurboIMAGE
-105	REMOTE 3000 MPE ERROR <i>%octal integer</i> RETURNED BY GETDSEG OF <i>decimal integer</i> WORDS
-106	REMOTE 3000 DATA INCONSISTENT
-107	DS/3000 SYSTEM ERROR
-110	OPENLOG RETURNED ERROR NUMBER <i>NW</i> TO DBOPEN LOGGING ENABLED AND NO LOG PROCESS RUNNING (3) DATA BASE CONTAINS INVALID LOG ID PASSWORD (8) LOG FILE CAN'T OBTAIN NECESSARY DISC SPACE (12) MAXIMUM USER COUNT PER LOG PROCESS REACHED (13) END OF FILE ON LOGFILE (15) DATA BASE CONTAINS INVALID LOG IDENTIFIER (16)
-111	WRITELOG RETURNED ERROR NUMBER <i>NW</i> TO DBPROCEDURE LOG PROCESS TERMINATED (3) LOG FILE CAN'T OBTAIN NECESSARY DISC SPACE (12) END OF FILE ON LOGFILE (15) (Refer to the MPE WRITELOG intrinsic for additional error numbers)
-112	CLOSELOG RETURNED ERROR NUMBER <i>NW</i> TO DBCLOSE LOG PROCESS TERMINATED (3) LOG FILE CAN'T OBTAIN NECESSARY DISC SPACE (12) END OF FILE ON LOGFILE (15)
-113	WRITELOG RETURNED ERROR NUMBER <i>NW</i> TO DBEND
-120	INSUFFICIENT STACK SPACE FOR DBLOCK
-121	ILLEGAL LOCK DESCRIPTOR COUNT
-122	BOUNDS VIOLATION ON DESCRIPTOR LIST

# DBERROR

Table 5-9. DBERROR Messages (Continued)

CONDITION WORD	DBERROR MESSAGE
-123	ILLEGAL RELATIONAL OPERATOR
-124	DESCRIPTOR LENGTH ERROR: MUST BE 9 OR MORE
-125	ILLEGAL SET NAME OR NUMBER IN DESCRIPTOR
-126	ILLEGAL ITEM NAME OR NUMBER IN DESCRIPTOR
-127	ILLEGAL ATTEMPT TO LOCK ON A COMPOUND ITEM
-128	VALUE FIELD TOO SHORT FOR THE ITEM SPECIFIED
-129	P28 IS LONGEST P-TYPE ITEM THAT CAN BE LOCKED
-130	ILLEGAL DECIMAL DIGIT IN TYPE 'P' DATA VALUE
-131	LOWERCASE CHARACTER IN TYPE 'U' DATA VALUE
-132	ILLEGAL DIGIT IN TYPE 'Z' DATA VALUE
-133	ILLEGAL SIGN CHARACTER IN TYPE 'Z' DATA VALUE
-134	TWO LOCK DESCRIPTORS CONFLICT IN SAME REQUEST
-135	DBLOCK CALLED WITH LOCKS ALREADY IN EFFECT IN THIS JOB/SESSION
-136	DESCRIPTOR LIST LENGTH EXCEEDS 2047 WORDS
-151	TEXT LENGTH GREATER THAN 512 BYTES
-152	DBCLOSE CALLED WHILE A TRANSACTION IS IN PROGRESS DBBEGIN CALLED WHILE A TRANSACTION IS IN PROGRESS
-153	DBEND CALLED WHILE NO TRANSACTION IS IN PROGRESS
-160	FILE CONFLICT: A FILE ALREADY EXISTS WITH THE ILR LOG FILE NAME
-161	CANNOT CHECK FOR ILR LOG FILE CONFLICT: FILE SYSTEM ERROR <i>nn</i>
-162	CANNOT BUILD ILR LOG FILE: FILE SYSTEM ERROR <i>nn</i>
-163	CANNOT INITIALIZE ILR LOG FILE: FILE SYSTEM ERROR <i>nn</i>
-164	CANNOT INITIALIZE ILR LOG HEADER: FILE SYSTEM ERROR <i>nn</i>

Table 5-9. DBERROR Messages (Continued)

CONDITION WORD	DBERROR MESSAGE
-165	CANNOT SAVE ILR LOG FILE: FILE SYSTEM ERROR <i>nn</i>
-166	CANNOT PURGE ILR LOG FILE: FILE SYSTEM ERROR <i>nn</i>
-170	CANNOT OPEN ILR LOG FILE:FILE SYSTEM ERROR <i>nn</i>
-171	CANNOT CLOSE ILR LOG FILE: FILE SYSTEM ERROR <i>nn</i>
-172	CANNOT READ ILR LOG FILE: FILE SYSTEM ERROR <i>nn</i>
-180	ILR LOG INVALID - INTERNAL FILE NAME DOES NOT MATCH ROOT FILE
-181	ILR LOG INVALID - INTERNAL GROUP NAME DOES NOT MATCH ROOT FILE
-182	ILR LOG INVALID - INTERNAL ACCOUNT NAME DOES NOT MATCH ROOT FILE
-183	ILR LOG INVALID - INTERNAL CREATION DATE DOES NOT MATCH ROOT FILE
-184	ILR LOG INVALID - INTERNAL LAST ACCESS DATE DOES NOT MATCH ROOT FILE
-185	CANNOT GET EXTRA DATA SEGMENT OF SIZE %XXXXX FOR ILR
-187	ILR ALREADY ENABLED FOR THIS DATA BASE
-188	ILR ALREADY DISABLED FOR THIS DATA BASE
-192	INVALID DBU
-193	DBU CONTROL BLOCK IS FULL
-194	INVALID DBB
-195	INVALID DBG
-196	DBB CONTROL BLOCK IS FULL
-197	DBG CONTROL BLOCK IS FULL

# DBERROR

Table 5-9. DBERROR Messages (Continued)

CONDITION WORD	DBERROR MESSAGE
-200	DATA BASE LANGUAGE NOT SYSTEM SUPPORTED
-201	NATIVE LANGUAGE SUPPORT NOT INSTALLED
-202	MPE NATIVE LANGUAGE SUPPORT ERROR <i>nn</i> RETURNED BY NLINFO
-203	DSCB EXTENSION FULL
-204	STACK OVERFLOW WHILE RECOVERING IN DBOPEN
-205	INCOMPATIBLE DS/3000 VERSION WHILE IN DBOPEN
-206	REMOTE TurboIMAGE DATABASE EXCEEDS IMAGE/3000 LIMITATION
-3xx	INTERNAL TurboIMAGE ERROR RETURNED ( <i>#n</i> )
3	GENMESSAGE ERROR: SET NOT IN CATALOG
4	GENMESSAGE ERROR: MESSAGE NOT IN CATALOG
10	BEGINNING OF FILE
11	END OF FILE
12	DIRECTED BEGINNING OF FILE
13	DIRECTED END OF FILE
14	BEGINNING OF CHAIN
15	END OF CHAIN
16	THE DATA SET IS FULL
17	THERE IS NO CHAIN FOR THE SPECIFIED SEARCH ITEM VALUE THERE IS NO ENTRY WITH THE SPECIFIED KEY VALUE THERE IS NO PRIMARY SYNONYM FOR THE SPECIFIED KEY VALUE NO CURRENT RECORD OR THE CURRENT RECORD IS EMPTY (CONTAINS NO ENTRY) THE SELECTED RECORD IS EMPTY (CONTAINS NO ENTRY)



Table 5-9. DBERROR Messages (Continued)

CONDITION WORD	DBERROR MESSAGE
18	BROKEN CHAIN - FORWARD AND BACKWARD POINTERS NOT CONSISTENT
20	DATA BASE CURRENTLY LOCKED BY ANOTHER USER SETS OR ENTRIES LOCKED WITHIN DATA BASE
22	DATA SET ALREADY LOCKED
23	CANNOT LOCK SET DUE TO LOCKED ENTRIES WITHIN IT      Conditional
24	ENTRIES CURRENTLY LOCKED USING DIFFERENT ITEM      Locks Only
25	CONFLICTING ENTRY LOCK ALREADY IN EFFECT
41	DBUPDATE WILL NOT ALTER A SEARCH OR SORT ITEM
42	DBUPDATE WILL NOT ALTER A READ-ONLY DATA ITEM
43	DUPLICATE KEY VALUE IN MASTER
44	CAN'T DELETE A MASTER ENTRY WITH NON-EMPTY DETAIL CHAINS
50	USER'S BUFFER IS TOO SMALL FOR REQUESTED DATA (ONLY RETURNED IF BUFFER IS TOO SMALL AND THE DATA TRANSFER WOULD WRITE OVER STACK MARKERS IN THE USER'S STACK)
60	DATA BASE ACCESS DISABLED
61	PROCESS HAS THE DATA BASE OPEN 63 TIMES; NO MORE ALLOWED
62	IMAGE DATA BASE CONTROL BLOCK FULL
63	DBG DISABLED; POTENTIAL DAMAGE; ONLY DBCLOSE ALLOWED

## DBERROR

Table 5-9. DBERROR Messages (Continued)

CONDITION WORD	DBERROR MESSAGE
64	NO ROOM FOR DBG ENTRY IN PCBX (MPE PORTION OF STACK)
66	DBG POINTED TO BY ROOT FILE DOES NOT MATCH
67	DBU DISABLED; POTENTIAL DAMAGE; ONLY DBCLOSE ALLOWED
68	DBB DISABLED; POTENTIAL DAMAGE; ONLY DBCLOSE ALLOWED
71	LOGGING NOT ENABLED FOR THIS USER
1xx	THERE IS NO CHAIN HEAD (MASTER ENTRY) FOR PATH <i>decimal integer: xx</i>
2xx	THE CHAIN FOR PATH <i>decimal integer: xx</i> IS FULL (CONTAINS 2,147,483,647 ENTRIES)
3xx	THE AUTOMATIC MASTER FOR PATH <i>decimal integer: xx</i> IS FULL
Others	UNRECOGNIZED CONDITION WORD: <i>decimal integer</i>

For Condition Words -9xx, 944, 947, and 948 returned by DBERROR, please refer to the *TurboIMAGE Profiler User Guide*.

Prints a multi-line message on the \$STDLIST device which describes a TurboIMAGE procedure call and explains the call's results as recorded in the calling program's status array.

## Syntax

DBEXPLAIN, *status*

## Parameters

*status* is the name of the array used as the *status* parameter in the TurboIMAGE procedure call about which information is requested.

**NOTE**

The *base*, *qualifier*, *dset*, and *password* parameters, if required by the procedure which put the results in the status area, must be unchanged when the call is made to DBEXPLAIN since information is taken from them as well.

## Discussion

Table 5-10 contains the general format for lines 2 through 6 of the message which is sent to \$STDLIST. Elements surrounded by brackets are sometimes omitted. Braces indicate that only one of the choices shown will be printed. Lines 5 and 6 are printed only if, during the preparation of lines 2, 3, and 4, TurboIMAGE detects that the status array contents are invalid, unrecognizable or incomplete, or if a message must be truncated to fit on a single line.

If the status array contents appear to be the result of something other than a TurboIMAGE procedure call or if the array is used by the called procedure for information other than that discussed here, the second choice for line 3 is printed. This would be the case for a successful call to DBGET which uses all ten status words to return a condition word, lengths, and record numbers.

If the status array contains an unrecognized error code, the second line 4 choice is printed.

If the condition word is greater than or equal to zero, the word, ERROR in line 2 is replaced by RESULT because non-negative condition words indicate success or exceptional conditions such as end-of-chain. Condition word values are explained in Appendix A.

You can use the offset information to locate the specific call statement that generated the status array contents if the call is made with a programming language which enables you to determine displacements of program statements or labels within the code. The identity of the code segment is not printed because it cannot be determined by DBEXPLAIN. Therefore, you need to be familiar with the program's functioning in order to locate the correct call. The offset portion of line 2 is printed only if the status array appears to be set by a TurboIMAGE library procedure call and contains valid offset information.

# DBEXPLAIN

Table 5-10. DBEXPLAIN Message Format

LINE	FORMAT
1	(a blank line)
2	TurboIMAGE {ERROR RESULT} [AT <i>offset</i> ]: CONDITION WORD= <i>conword</i>
3	{ <i>intrinsicname</i> , MODE <i>x</i> , ON [ <i>setname</i> OF] <i>basename</i> [; PASSWORD= <i>password</i> ]} {TurboIMAGE CALL INFORMATION NOT AVAILABLE}
4	{ <i>message</i> UNRECOGNIZED CONDITION WORD: <i>conword</i> }
5	[OCTAL DUMP OF STATUS ARRAY FOLLOWS]
6	[ <i>octal display</i> ]
7	(a blank line)
PARAMETER	EXPLANATION
<i>offset</i>	is the octal PB-relative offset within the user's code segment of the TurboIMAGE procedure call. See the <i>MPE Intrinsics Reference Manual</i> for a discussion of PB (program base) relative addresses.
<i>conword</i>	is the condition word (from the first word of <i>status</i> ) printed as a decimal integer and corresponding to the condition words described in Appendix A.
<i>intrinsicname</i>	is the name of the TurboIMAGE library procedure (intrinsic) which was called and which set the contents of the <i>status</i> array.
<i>x</i>	is the value of the <i>mode</i> parameter as a decimal integer.
<i>setname</i>	is the value of the second parameter, usually a data set name or number, as passed to the procedure which set the <i>status</i> array contents. The second parameter can be a data item name or number if the procedure in question is DBINFO. If the procedure is DBOPEN, DBLOCK, DBUNLOCK, or certain modes of DBINFO or DBLCOSE, <i>setname</i> is omitted.
<i>password</i>	is printed at the end of line 3 only if the error relates to the <i>password</i> parameter of DBOPEN.
<i>basename</i>	is the data base specified in the procedure which was called and set the <i>status</i> array contents.

Table 5-10. DBEXPLAIN Message Format (Continued)

LINE	FORMAT
<i>message</i>	is an English language description of the result based on the condition word and other <i>status</i> array information. The message is generated by the DBERROR procedure which is also described in this section. See Table 5-2 for all possible line 4 messages.
<i>octal display</i>	is a listing of each word of <i>status</i> printed as a string of 6 octal digits. Adjacent <i>status</i> words are separated by a blank and the entire line is 69 characters long.

Figure 5-1 contains four examples of messages generated by DBEXPLAIN.

<p>IMAGE RESULT AT %001103: CONDITION WORD=0          DBLOCK,MODE1, ON ORDERS          SUCCESSFUL EXECUTION - NO ERROR</p>	<p><i>DBLOCK=intrinsic name</i>  <i>ORDERS=data base name</i>  <i>NO ERROR=message</i></p>
<p>IMAGE ERROR AT %001057: CONDITION WORD=-12          DBPUT,MODE1, ON DATE-MASTER OF ORDERS          DBPUT CALLED WITH DATA BASE NOT LOCKED</p>	<p><i>DATE-MASTER=data set name</i></p>
<p>IMAGE RESULT AT %001057: CONDITION WORD=16          DBPUT,MODE1, ON #1 OF ORDERS          THE DATA SET IS FULL</p>	<p><i>#1=data set number</i></p>
<p>IMAGE RESULT: CONDITION WORD=5349          IMAGE CALL INFORMATION NOT AVAILABLE          UNRECOGNIZED CONDITION WORD: 5349          OCTAL DUMP OF STATUS ARRAY FOLLOWS:          012345 054321 011111 022222 033333 044444 055555 066666 077777          .....<i>octal display</i>.....</p>	

Figure 5-1. Sample DBEXPLAIN Messages

# DBFIND

## INTRINSIC NUMBER 404

Locates master set entry that matches the specified search item value and sets up pointers to the first and last entries of a detail data set chain in preparation for chained access to the data entries which are members of the chain. The path is determined and the chain pointers located on the basis of a specified search item and its value.

## Syntax

`DBFIND,base,dset,mode,status,item,argument`

## Parameters

*base* is the name of the array used as the *base* parameter when opening the data base. The first word of the array must contain the *base id* returned by DBOPEN. (Refer to DBOPEN for more information about *base id*.)

*dset* is the name of an array containing the left-justified name of the detail data set to be accessed or is an integer referencing the data set by number. The data set name may be 16 characters long or, if shorter, terminated by a semicolon or blank.

*mode* must be an integer equal to 1.

*status* is the name of a ten-word array in which TurboIMAGE returns status information about the procedure. If the procedure executes successfully, the status array contents are:

Word	Contents
1	Condition word is 0.
2	Zero.
3-4	Doubleword current record number set to zero.
5-6	Doubleword count of number of entries in chain.
7-8	Doubleword record number of last entry in chain.
9-10	Doubleword record number of first entry in chain.

*item* is the name of an array containing a left-justified name of the detail data set search item or is an integer referencing the search item number that defines the path containing the desired chain. The name may be 16 characters long or, if shorter, terminated by a semicolon or blank. The specified search item defines the path to which the chain belongs.

*argument*

contains a value for the search item to be used in calculated access to locate the desired chain head in the master data set.

## Discussion

The current values of chain count, backward pointer, and forward pointer for the detail data set referenced in *dset* are replaced by the corresponding value from the chain head. A current path number, which is maintained internally, is set to the new path number and the current record number for the data set is set to zero. Refer to Section 10 for further information about chain heads and internally maintained data set information.

Note that although a master set entry exists with the specified search item value, the data set chain may be empty.

**Table 5-11. DBFIND Condition Word Values**

FILE SYSTEM AND MEMORY MANAGEMENT FAILURES:	-1 -3 -4	FOPEN intrinsic failure. FREADDIR failure. FREADLABEL failure.
CALLING ERRORS:	-11 -21 -31 -52	Bad <i>base</i> parameter. Bad data set reference. Bad <i>mode</i> . Bad <i>item</i> .
COMMUNICATIONS ERRORS:	-102 -106 -107	DSWRITE failure. Remote data inconsistent. DS procedure call error.
EXCEPTIONAL CONDITIONS:	17 63	No master entry. Bad DBG.
		Consult Appendix A for more information about these conditions.

### NOTE

A call to DBOPEN does not open individual data sets. Thus, a call to DBFIND (or DBGET) that accesses a data set for the first time (or after the data set has been closed), must open the data set. This causes extra overhead not incurred by subsequent calls to the same data set by DBFIND or DBGET.

# DBGET

## INTRINSIC NUMBER 405

Provides eight different methods for accessing the entries of a data set.

### Syntax

DBGET, *base*, *dset*, *mode*, *status*, *list*, *buffer*, *argument*

### Parameters

- base*

is the name of the array used as the *base* parameter when opening the data base. The first word of the array must contain the *base id* returned by DBOPEN. (Refer to DBOPEN for more information about *base id*.)
- dset*

is the name of an array containing the left-justified name of the data set to be read or is an integer referencing the data set by number. The data set name may be 16 characters long or, if shorter, terminated by a semicolon or blank.
- mode*

contains an integer between 1 and 8, inclusive, which indicates the reading method. The methods are:

	Mode	Method
1	Re-read	Read the entry at the internally maintained current record address (argument parameter is ignored).
2	Serial Read	Read the first entry whose record address is greater than the internally maintained current address (argument parameter is ignored).
3	Backward Serial Read	Read the first entry whose record address is less than the internally maintained current address (argument parameter is ignored).
4	Directed Read	Read the entry, if it exists, at the record address specified in the argument parameter (argument is treated as a doubleword record number).
5	Chained Read	Read the next entry in the current chain, the entry referenced by the internally maintained forward pointer (argument parameter is ignored).



	Mode	Method
6	Backward Chained Read	Read the previous entry in the current chain, the entry referenced by the internally maintained backward pointer. argument parameter is ignored.
7	Calculated Read	Read the entry with a search item value that matches the value specified in argument. The entry is in the master data set specified by <i>dset</i> .
8	Primary Calculated Read	Read the entry occupying the primary address of a synonym chain using the search item value specified in argument to locate the entry. If the entry is not a primary entry in a master data set specified by <i>dset</i> , it is not read. (Refer to Section 10 for synonym chain description.)

*status*

is the name of a ten-word array in which TurboIMAGE returns status information about the procedure. If the procedure executes successfully, the status array contents are:

Word	Contents
1	Condition word is 0.
2	Integer word length of the logical entry read into the buffer array.
3-4	Doubleword record number of the data entry read.
5-6	Doubleword zero, unless the entry read is a primary entry in which case it is the number of entries in the synonym chain.
7-8	Doubleword record number of the preceding entry in the chain of the current path.
9-10	Doubleword record number of the next entry in the chain of the current path.

Table 5-12 lists the contents of Word 1 when the procedure does not succeed.

*list*

is the name of an array containing an ordered set of data item identifiers, either names or numbers. The values for these data items are placed in the array specified by the buffer parameter in the same order as they appear in the *list* array.

## DBGET

The *list* array may contain a left-justified set of data item names, separated by commas and terminated by a semicolon or blank. No embedded blanks are allowed and no name may appear more than once.

When referencing by number, the first word of the list array is an integer *n* which is followed by *n* unique data item numbers (one-word positive integers).

The *list* not only specifies the data items to be retrieved immediately but is saved internally by TurboIMAGE as the *current list* for this data set. The current list is unchanged until a different list is specified in a subsequent call to DBGET, DBPUT, or DBUPDATE for the same access path and data set.

Some special list constructs are allowed. These are described in Table 5-20 with the DBPUT procedure. List processing is a relatively high overhead operation which may be shortened in subsequent calls by using the asterisk construct to specify that the current list is to be used. Use of this construct can save considerable processing time. However, be sure a *current list* exists before using the asterisk or TurboIMAGE will assume a null list.

### *buffer*

is the name of the array to which the values of data items specified in the list array are moved. The values are placed in the same order as specified in the list array. The number of words occupied by each value corresponds to the number required for each data type multiplied by the sub-item count.

### *argument*

is ignored except when mode equals 4, 7, or 8.

If mode is 4, *argument* contains a doubleword record number of the entry to be read. (Refer to Section 6 for suggestions on using a doubleword parameter in a BASIC program.)

If mode is 7 or 8, *argument* contains a search item value for the master data set referenced by *dset*.

## Discussion

The internal backward and forward pointers for the data set are replaced by the current path's chain pointers from the entry just read. If the data set is a master, they are synonym chain pointers (refer to Section 10). If it is a detail with at least one path, the current path is the one established by the last successful call to DBFIND, or if no call has been made it is the primary path. If there are no paths defined, the internal pointers are set to zeros.

The location of the entry just read becomes the current record for the data set.

## NOTE

A call to DBOPEN does not open individual data sets. Thus, a call to DBFIND (or DBGET) that accesses a data set for the first time (or after the data set has been closed), must open the data set. This causes extra overhead not incurred by subsequent calls to the same data set by DBFIND or DBGET.

Table 5-12. DBGET Condition Word Values

FILE SYSTEM AND MEMORY MANAGEMENT FAILURES:	-1	FOPEN intrinsic failure.
	-3	FREADDIR failure.
	-4	FREADLABEL failure.
CALLING ERRORS:	-11	Bad <i>base</i> parameter.
	-21	Bad data set reference.
	-31	Bad <i>mode</i> .
	-51	Bad <i>list</i> length.
	-52	Bad <i>list</i> or bad <i>item</i> .
COMMUNICATIONS ERRORS:	-102	DSWRITE failure.
	-106	Remote data inconsistent.
	-107	DS procedure call error.
EXCEPTIONAL CONDITIONS:	10	Beginning of file. (mode 3)
	11	End of file. (mode 2)
	12	Directed beginning of file. (mode 4)
	13	Directed end of file. (mode 4)
	14	Beginning of chain. (mode 6)
	15	End of chain. (mode 5)
	17	No entry. (modes 1, 4, 7, 8)
	18	Broken chain. (modes 5 or 6)
	50	Buffer is too small (will only be returned if buffer is too small and the data transfer would write over stack markers in the user's stack).
	62	DBG full.
	63	Bad DBG.
		Consult Appendix A for more information about these conditions.

# DBINFO

## INTRINSIC NUMBER 402

Provides information about the data base being accessed. The information returned is restricted by the user class number established when the data base is opened; any data items, data sets, or paths of the data base which are inaccessible to that user class are considered to be non-existent.

### Syntax

`DBINFO,base,qualifier,mode,status,buffer`

### Parameters

<i>base</i>	is the array name used as the <i>base</i> parameter when opening the data base; must contain the <i>base id</i> returned by DBOPEN. (Refer to DBOPEN for additional <i>base id</i> information.)
<i>qualifier</i>	is the name of an array containing a data set/data item name or an integer referencing a data item/data set, depending on the value of the <i>mode</i> parameter (refer to Table 5-13 for <i>mode/qualifier</i> relationship). This parameter form is identical to <i>dset</i> and <i>item</i> parameters for DBPUT and DBFIND.
<i>mode</i>	is an integer indicating the type of information is desired. Refer to Table 5-13 for <i>mode</i> integer information (data item modes <i>1nn</i> , data set modes <i>2nn</i> , path modes <i>3nn</i> , logging modes <i>4nn</i> , subsystem modes <i>5nn</i> ).
<i>status</i>	is the name of a ten-word array in which TurboIMAGE returns status information about the procedure. If the procedure executes successfully, the status array contents are:

Word	Contents
1	Condition word is 0.
2	Word length of information in buffer array.
3-4	Unchanged from previous procedure call using this array.
5-10	Information about the procedure call and its results. Refer to Appendix A for a description of this information.

Table 5-14 lists the contents of Word 1 when the procedure does not succeed.

<i>buffer</i>	is the name of an array in which the requested information is returned. The contents of the <i>buffer</i> array vary according to the <i>mode</i> parameter used. They are also described in Table 5-13.
---------------	--

Table 5-13. *mode* and *qualifier* Values and Results

<i>mode</i>	PURPOSE	<i>qualifier</i>	<i>buffer</i> ARRAY CONTENTS	COMMENTS
101	Defines type of access available for specific item.	data item name or number	word 1 $\pm$ data item number	If negative, data item can be updated or entry containing it can be added or deleted in at least one data set.
102	Describes specific data item.	data item name or number	word 1 . . 8 data item name 9 data type $\triangle$ 10 sub-item length 11 sub-item count 12 0 13 0	Left-justified and padded with blanks, if necessary.  (I,J,K,R,U,X,Z,P) $\triangle$ indicates blank  integers
103	Identifies all data items available in data base and type of access allowed.	(ignored)	word 1 n 2 $\pm$ data item number . . . . . n+1 $\pm$ data item number	n = number of data items available  Arranged in data item number order. If positive, read-only access. If negative, update or modify access in at least one data set.
104	Identifies all data items available in specific data set and type of access allowed.	data set name or number	(Same as <i>mode</i> 103)	(Same as <i>mode</i> 103 except arranged in order of occurrence in data entry)

Table 5-13. *mode* and *qualifier* Values and Results (Continued)

<i>mode</i>	PURPOSE	<i>qualifier</i>	<i>buffer</i> ARRAY CONTENTS	COMMENTS
201	Defines type of access available for specific data set.	data set name or number	word 1 $\pm$ data set number	If negative, entries can be added or deleted.
202	Describes specific data set.	data set name or number	word 1 . . 8 data set name 9 set type $\triangle$ 10 entry word-length 11 blocking factor 12 0 13 0 14 number of entries in set 15 16 17 capacity of set	Left-justified and padded with blanks, if necessary. (M,A,D) $\triangle$ indicates blank integers doubleword integers
203	Identifies all data sets available in data base and type of access allowed.	(ignored)	word 1 n 2 $\pm$ data set number . . . n+1 $\pm$ data set number	n = number of data sets available Arranged in data set number order. If positive, read and possibly data item up-date access. If negative, modify access allowed.
204	Identifies all data sets available which contain specified data item and type of access allowed.	data item name or number	(Same as <i>mode</i> 203)	(Same as <i>mode</i> 203)

Table 5-13. *mode* and *qualifier* Values and Results (Continued)

mode	PURPOSE	qualifier	buffer ARRAY CONTENTS	COMMENTS
301	Identifies paths defined for specified data set.	data set name or number	<div>word</div> <div><div><div>1</div><div>n</div></div><div><div>2</div><div>data set number</div></div><div><div>3</div><div>search item number</div></div><div><div>4</div><div>sort item number</div></div><div><div>.</div><div>.</div></div><div><div>.</div><div>.</div></div><div><div>.</div><div>.</div></div><div><div>.</div><div>.</div></div><div><div>.</div><div>.</div></div></div> <div><div>3n-1</div><div>data set number</div></div> <div><div>3n</div><div>search item number</div></div> <div><div>3n+1</div><div>sort item number</div></div> <div><div>} n = number of paths</div><div>} Repeat for each path. If <i>qualifier</i> refers to master, set number is for detail. If <i>qualifier</i> refers to detail, set number is for master. Item numbers identify items in detail.</div><div>} Path designators presented in order of their appearance in schema.</div></div> <div>Note: If sort item is zero, none exists or it is inaccessible. A path designator is not included if user does not have access to search item.</div>	
302	Identifies search item for specified data set.	<div>master data set name or number</div> <div>OR</div> <div>detail data set name</div>	<div>word</div> <div><div><div>1</div><div>search item number</div></div><div><div>2</div><div>0</div></div></div> <div>word</div> <div><div><div>1</div><div>data item number</div></div><div><div>2</div><div>data set number</div></div></div>	<div>In master set, zero if inaccessible</div> <div>In detail set. For primary path. Of related master.</div> <div>Both are zero if search item is inaccessible.</div>

Table 5-13. *mode* and *qualifier* Values and Results (Continued)

<i>mode</i>	PURPOSE	<i>qualifier</i>	<i>buffer</i> ARRAY CONTENTS	COMMENTS
401	Obtains information relating to logging	(ignored)	<div>word</div> <div><div><div>1</div><div>.</div><div>.</div><div>4</div></div><div>Log Identifier Name</div><div>5</div><div>Data Base Log Flag</div><div>6</div><div>User Log Flag</div><div>7</div><div>Transaction Flag</div><div>8</div><div>User Transaction Number</div><div>9</div></div>	<div><div>Left-justified and padded with blanks if necessary.</div><div>1 if data base enabled for logging, otherwise 0.</div><div>1 if user is logging, otherwise 0.</div><div>1 if user has a transaction in progress, otherwise 0.</div><div>Doubleword</div></div>
402	Returns information about ILR	(ignored)	<div>word</div> <div><div><div>1</div><div>2</div><div>3</div><div>4</div></div><div>ILR Log Flag</div><div>Calendar Date</div><div>Clock Time</div><div>5</div><div>ILR used</div><div>6</div><div><div>Intrinsic</div><div>△</div></div><div>7</div><div>.</div><div>.</div><div>.</div><div>14</div><div>Data Set Name</div><div>15</div><div>Reserved</div><div>16</div></div>	<div><div>1 if data base enabled for ILR, otherwise 0.</div><div>Date ILR enabled (mmddy).</div><div>Time ILR enabled, 2 words (hhmmss). </div><div>1 if ILR was used, otherwise 0.</div><div>P=DBPUT; D=DBDELETE △ indicates blank.</div><div>When ILR used, 8 words. Left-justified and padded with blanks if necessary.</div><div>Otherwise, words 6 through 14 are ASCII blanks.</div></div>



Table 5-13. *mode* and *qualifier* Values and Results (Continued)

<i>mode</i>	PURPOSE	<i>qualifier</i>	<i>buffer</i> ARRAY CONTENTS	COMMENTS
501	To check subsystem access to the data base. (Refer to DBUTIL SHOW/SET commands for more information)	(ignored)	word 1 <span style="border: 1px solid black; padding: 2px;">Subsystem Access</span>	0 = no access 1 = read access 3 = read/write access
901	To obtain the Native Language attribute of the data base. Returns MPE code for language attribute	(ignored)	word 1 <span style="border: 1px solid black; padding: 2px;">Language ID</span>	

Table 5-14. DBINFO Condition Word Values

FILE SYSTEM AND MEMORY MANAGEMENT FAILURES:	-1 -4	FOPEN intrinsic failure. FREADLABEL failure.
CALLING ERRORS:	-11 -21 -31	Bad <i>base</i> parameter. Bad <i>base</i> item reference. Bad <i>mode</i> .
COMMUNICATIONS ERRORS:	-102 -106 -107	DSWRITE failure. Remote data inconsistent. DS procedure call error.
EXCEPTIONAL CONDITIONS:	50  63	Buffer too small (will only be returned if buffer is too small and the data transfer would write over stack markers in the user's stack). Bad DBG.
		Consult Appendix A for more information about these conditions.

# DBLOCK

## INTRINSIC NUMBER 409

Applies a logical lock to a data base, one or more data sets, or one or more data entries.

### Syntax

`DBLOCK,base,qualifier,mode,status`

### Parameters

*base* is the name of the array used for the *base* parameter when opening the data base. The first word of the array must contain the *base id* returned by DBOPEN. (Refer to DBOPEN for more information about *base id*.)

*qualifier*

**Modes 1 and 2:** Ignored.

**Modes 3 and 4:** An integer variable referencing the data set number or the name of an array containing a data set name. Could also be "@", applying a data base lock.

**Modes 5 and 6:** The name of the array containing the lock descriptors. The format for lock descriptors is given in Figure 5-2.

Use care when changing modes. The *qualifier* parameter may also change.

*mode* contains an integer indicating the type of locking desired (refer to Table 5-15).

*status* is the name of a ten-word array in which TurboIMAGE returns status information about the procedure. If the procedure executes successfully, the status array contents are:

Word	Contents
1	Condition word is 0.
2	The number of lock descriptors that were successfully applied in the DBLOCK request. For successful locks in modes 1 through 4 this will be 1.
3	If condition word = 20, this word contains 0 if data base locked, 1 if data set or entries locked.

4	Reserved: Contents undefined.
5-10	Information about the procedure call and its results. Refer to Appendix A for a complete description of this information.

Table 5-17 lists the contents of Word 1 when the procedure does not succeed.

<b>NOTE</b>
-------------

Concurrent processes running in a process-handling environment must have MR capability if they are calling DBLOCK.

## Discussion

The format of the array containing a list of lock descriptors is illustrated in Figure 5-2 and applies only for Locking Modes 5 or 6. The number of lock descriptors ( $n$ ) is a one-word binary integer. Only the first  $n$  lock descriptors are processed. If  $n$  is zero, DBLOCK returns without taking any action. The format of a lock descriptor is illustrated in Figure 5-3, and the lock descriptor fields are described in Table 5-16.

The shortest possible descriptor is 9 words long consisting of the length field and a dset field containing @. Although the dset field only contains an at-sign, it must still be 8 words long. The length of the entire descriptor array may not exceed 2047 words.

Lock descriptors are sorted by data set number, then by value provided for the lock item. TurboIMAGE does not sort by item within the set, because more than one item per data set constitutes a conflicting lock descriptor (TurboIMAGE error -134).

# DBLOCK

Table 5-15. Locking *mode* Options

LOCK MODE	LOCK LEVEL	LOCKING TYPE	DESCRIPTION
1	Base	Unconditional	DBLOCK applies an unconditional lock to the whole data base, returning to the calling program only after the lock is successful (or if an error occurs). The <i>qualifier</i> parameter is ignored.
2	Base	Conditional	DBLOCK applies a conditional lock to the data base and returns immediately. A condition word of zero indicates success. A non-zero condition word indicates the reason for failure. (Refer to Table 5-17.)
3	Set	Unconditional	<p>DBLOCK applies an unconditional lock to a data set. The <i>qualifier</i> parameter must specify the name of an array containing the left-justified name of the data set or the name of an integer referencing the data set number. The data set name may be 16 characters long or, if shorter, terminated by a semicolon or blank.</p> <p>The data set need not be accessible for read or write access to the user requesting the lock.</p>
4	Set	Conditional	DBLOCK applies a conditional lock of the same type as mode 3. It always returns to the calling program immediately. A condition word of zero indicates success and a non-zero condition word indicates a reason for failure. (Refer to Table 5-17.)

Table 5-15. Locking *mode* Options (Continued)

LOCK MODE	LOCK LEVEL	LOCKING TYPE	DESCRIPTION
5	Entry	Unconditional	DBLOCK applies unconditional locks to the data entries specified by lock descriptors. The <i>qualifier</i> parameter must specify the name of an array containing the lock descriptors. The format of the array is shown in Figure 5-2. It returns only when all the locks have been acquired.
6	Entry	Conditional	<p>DBLOCK applies conditional locks of the same type as mode 5. If multiple lock descriptors are specified, a return is made when DBLOCK encounters a lock descriptor that it cannot apply. All locks that have been applied until that point are retained.</p> <p>Since the locks are not executed in the order supplied by the user, it is not predictable which locks are held and which are not after an unsuccessful mode 6 DBLOCK. Status word 2 indicates how many lock descriptors were actually successful. It is recommended that a DBUNLOCK be issued after any unsuccessful mode 6 DBLOCK.</p>
			NOTE: Be careful when changing modes. The <i>qualifier</i> parameter may change.

DBLOCK

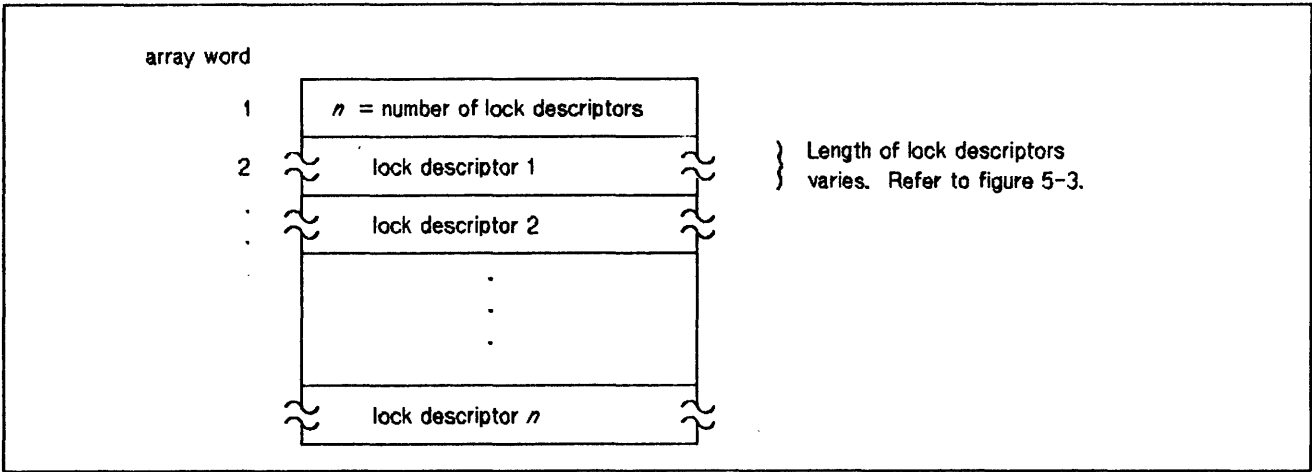


Figure 5-2. Qualifier Array Format For Locking Modes 5 and 6

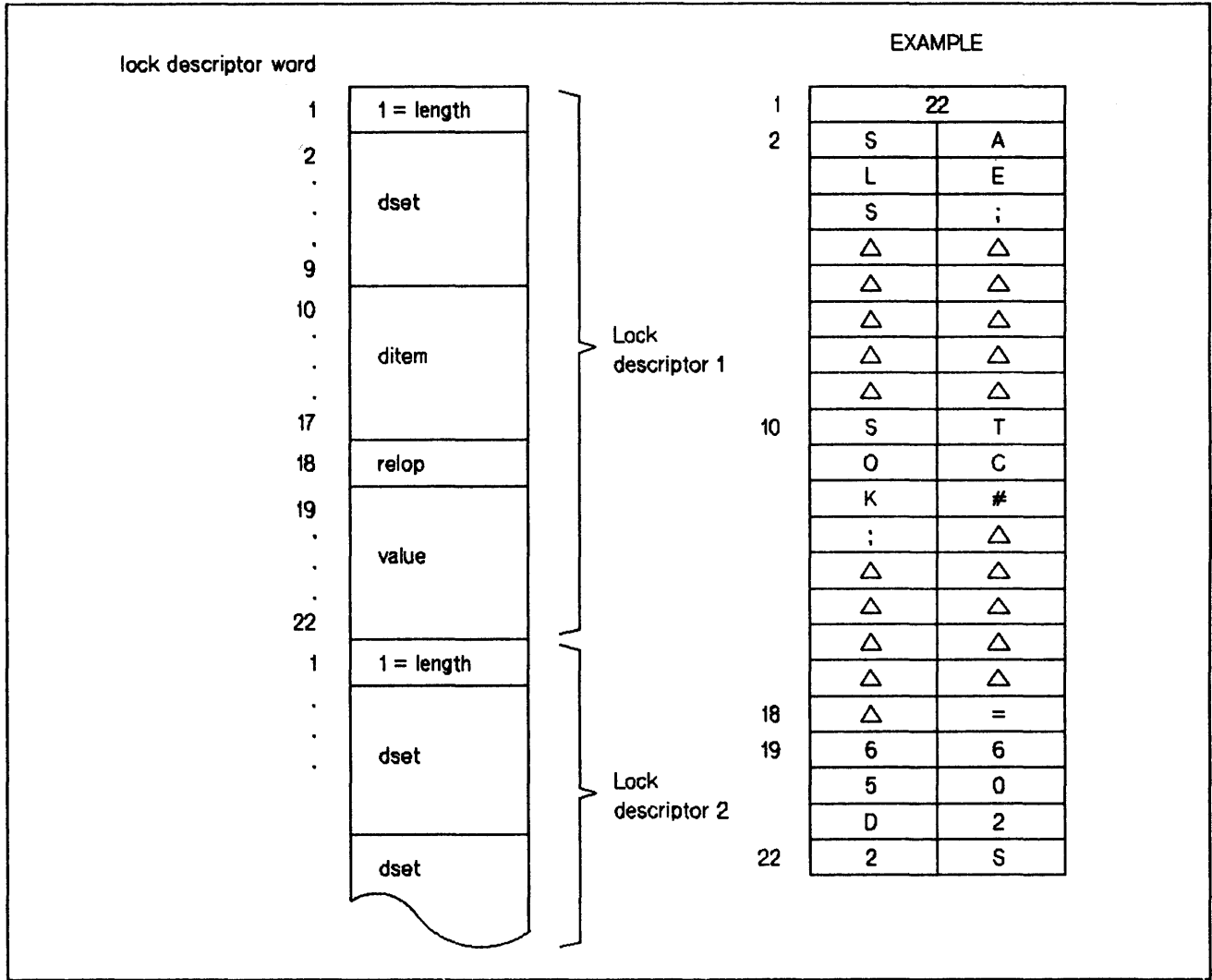


Figure 5-3. Lock Descriptor Format

Table 5-16. Lock Descriptor Fields

FIELD NAME	DESCRIPTION
<i>length</i>	is a one-word binary integer specifying the physical length in words of the lock descriptor, including the <i>length</i> field itself.
<i>dset</i>	<p>is always 8 words long and describes the data set in which locks are placed. It may be one of the following:</p> <p>A data set name, left-justified, 16 characters long or, if shorter, terminated with a blank or semicolon. For example: SALES;</p> <p>A data set number, a binary integer in the range of 1 to 199 stored in the first word.</p> <p>An at-sign (@) stored in the first byte of the <i>dset</i> and a lock descriptor length of two will indicate that the whole data base is to be locked. All unusual bytes are ignored. In this case, the <i>ditem</i>, <i>relop</i>, and <i>value</i> fields are ignored and may be omitted if desired.</p> <p>A blank or semicolon (first byte) or binary zero (first word) indicating that the whole lock descriptor is to be ignored. (It is counted as one of the <i>n</i> descriptors.)</p> <p>The data set, if specified, need not be accessible for read or write access to the user requesting the lock.</p>
<i>ditem</i>	<p>is always 8 words long unless an @ is stored in the first byte. It may be one of the following:</p> <p>A data item name, left-justified, 16 characters long or, if shorter, terminated with a blank or semicolon.</p> <p>A data item number stored as a binary integer in the first word. It may be in the range of 1 to 255.</p> <p>An at-sign (@) stored in the first byte of the <i>dset</i> indicating that the whole data set specified in <i>dset</i> is to be locked. All unused bytes are ignored and may be omitted if desired.</p> <p>The data item need not be a search item, nor does it have to be accessible to the user requesting the lock. However, it cannot be a compound item or a P-type item longer than P28.</p>
<i>relop</i>	<p>is one word long and contains one of the three relational operators represented as two ASCII characters:</p> <p>&lt;= less than or equal          &gt;= greater than or equal          =Δ or Δ = equal (Δ indicated space character)</p>

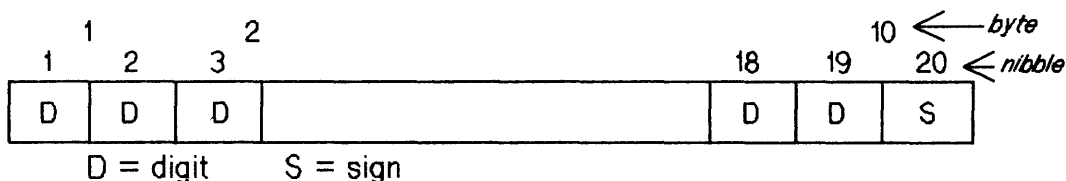
## DBLOCK

Table 5-16. Lock Descriptor Fields (Continued)

FIELD NAME	DESCRIPTION
<i>value</i>	is the value of the data item to be locked. It must be stored in exactly the same way as it is stored in the data base. IMAGE extracts as many words as required by the corresponding data item definition (in the schema). The rest (if any) are ignored.

If you specify a data item of type P, U, or Z in a lock descriptor, TurboIMAGE checks that the value is valid for that data item type. The following checks are made:

- If the data item is type P, the right half of the rightmost byte must contain a sign and all preceding nibbles must contain decimal digits represented in Binary Coded Decimal (BCD) format. For example, if a data item is defined as type P with a length of 20, the format must be:



This would be declared in COBOL as 19 digits plus a sign or 20 nibbles (P20 in the schema):

```
S9(19)    COMP-3
```

Type P data item used in a lock descriptor may not exceed 28 nibbles (7 words) in length. The locking system treats all sign digits other than  $1101_2$  as identical.  $1101_2$  is assumed to be a negative sign.

- If the data item is type U, the value must not contain any lowercase alphabetic characters in the range of a through z (for non-native language use only).
- If the data item is type U or X, and a lock specifies an inequality, the language of the data base will be used.
- If the data item is type Z, each byte preceding the last one must contain an 8-bit digit represented in ASCII format and the last byte must contain a value representing a digit and a sign. (Refer to the description of packed decimal numbers in Section 3 of the *Machine Instruction Set Manual*.)



Table 5-17. DBLOCK Condition Word Values

FILE SYSTEM AND MEMORY MANAGEMENT FAILURES:	-7	FLOCK failure.
CALLING ERRORS:	-11 -31 -120 -121 -122 -123 -124 -125 -126 -127 -128  -129 -130 -131 -132 -133 -134 -135 -136	Bad <i>base</i> parameter. Bad <i>mode</i> value. Not enough stack to perform DBLOCK. Descriptor count error. Descriptor list bad. Is not entirely within stack. Illegal <i>relop</i> in a descriptor. Descriptor too short. Must be greater than or equal to 9. Bad set name/number. Bad item name/number. Attempt to lock using a compound item. Value field too short in a descriptor.  P-type item longer than P28 specified. Illegal digit in a P-type value. Lowercase character in type U value. Illegal digit in type Z value. Illegal sign in type Z value. Two descriptors conflict. DBLOCK called with locks already in effect in this job/session. Descriptor list exceeds 2047 words.
COMMUNICATIONS ERRORS:	-102 -103 -106 -107	DSWRITE failure. Remote stack too small. Remote data inconsistent. DS procedure call error.
EXCEPTIONAL CONDITIONS:	20  22 23 24 25 62 63	<div style="text-align: right;">Applicable Modes</div> Data base locked or contains locks (2,4,6) (Status word 3:0 = data base locked 1 = data set or entries locked) (2) Data set locked by another process (4,6) Entries locked within set (4) Item conflicts with current locks (6) Entry or entries already locked (6) DBG full*. (3,4,5,6) Bad DBG.
		Appendix A contains more information about these condition codes.

\*NOTE: If error 62 occurs when multiple lock descriptors are specified, some of the descriptors may have been successfully completed. If so, they are not unlocked by TurboIMAGE before returning the error. Therefore, issue a DBUNLOCK after any positive-numbered error unless you have reason to do otherwise.

# DBMEMO

## INTRINSIC NUMBER 414

Used to log user data (ASCII or binary) to the log file.

### Syntax

`DBMEMO,base,text,mode,status,textlen`

### Parameters

*base* is the name of the array used as the *base* parameter when opening the data base. The first word of the array must contain the *base id* returned by DBOPEN. (Refer to DBOPEN for more information about the *base id*.)

*text* is an array of up to 256 words which contains user data (ASCII or binary) to be written to the logfile as part of the DBMEMO log record.

*mode* must be an integer equal to 1.

*status* is the name of a ten-word array in which TurboIMAGE returns status information. If the procedure executes successfully, the status array contents are:

Word	Contents
1	Condition word is 0.
2-4	Unchanged from previous procedure call using this array.
5-10	Procedure call information. Refer to Appendix A for a description of this information.

Table 5-18 lists the contents of Word 1 when the procedure does not succeed.

*textlen* is an integer equal to the number of words to be logged in the *text* parameter, or if negative, equal to the number of bytes. Length may be zero.

### Discussion

DBMEMO is used to log user data to the logfile when the user process is logging. No action occurs if the process is not logging. DBMEMO may be used to add additional auditing information to the logfile or to facilitate the identification of transactions in the event of a failure and subsequent recovery.

Table 5-18. DBMEMO Condition Word Values

CALLING ERRORS:	-11 -31 -151	Bad data base reference. Bad mode. Text length too large.
COMMUNICATIONS FAILURES:	-102 -106 -107	DSWRITE failure. Remote data inconsistent. DS procedure call error.
EXCEPTIONAL CONDITIONS:	0 62 63	Logging not enabled for this user. DBG cannot expand. Bad DBG.
LOG SYSTEM FAILURES:	-111	WRITELOG intrinsic failure.
		Consult Appendix A for more information about these conditions.

# DBOPEN

## INTRINSIC NUMBER 401

Initiates access to the data base and establishes the user class number and access mode for all subsequent data base access.

## Syntax

`DBOPEN, base, mode, status`

## Parameters

*base*

is the name of a word array containing a string of ASCII characters. The string must consist of a pair of blanks followed by a left-justified data base name (maximum 6 characters) and terminated by a semicolon or blank ( $\Delta$ ), for example, " $\Delta\Delta$ orders;". If the data base is successfully opened, TurboIMAGE replaces the pair of blanks with a value called the *base id*. The *base id* uniquely identifies this access path between the data base and the process calling DBOpen. In all subsequent accesses to the data base, the first word of *base* must be this *base id*; therefore, the array should not be modified. Note that the *base id* contains a number that distinguishes between the 63 access paths allowed for each process for accessing a given data base.

**NOTE**

The access path to the data base is defined by the base id returned by DBOpen together with the PIN of the calling process. As the PIN defines the data base access path for that particular process, the base id cannot be passed between processes in an attempt to reduce the quantity of required DBOpen calls.

To access a data base catalogued in a group other than the user's log-on group, the data base name must be followed by a period and the group name; for example, ORDERS.GROUPX. If the data base is in an account other than the user's account, the group name must be followed by a period and the account name; for example, ORDERS.GROUPX.ACCOUNT1.

You may use a :FILE command before executing the application program to equate the data base name or the data-base-access file name to another data base or data-base-access file name. Only the formal file designator, actual file designator, and the DEV= parameter may be used.

*password*

is the name of a word array containing a left justified string of ASCII characters consisting of an optional password followed by an optional user identifier.

The following constructs are valid for the password and user identifier (a  $\Delta$  stands for a blank):

$\Delta$ [/USERIDENT]	Access Class Zero (0).
;[/USERIDENT]	Creator Access.
password[/USERIDENT]	Password Access.

If either the password or the user identifier strings are less than eight characters long, they must be terminated with a semicolon or blank.

The password establishes a user class number as described in Section 2. A semicolon supplied as the password implies creator class 64. The user identifier is used by the program DBRECOV to distinguish between users logged on under the same name and account.

The following are valid examples:

```

;
CLERK $\Delta$ 
CLERK;
CLERK;/JOE;
CLERK $\Delta$ /JOE;
 $\Delta$ /DBA

```

*mode*

is an integer between 1 and 8, inclusive, corresponding to the valid TurboIMAGE access modes described in Section 4. Here is a brief summary:

Access Mode	Associated Capabilities	Concurrent Modes Allowed
1	Modify with enforced locking. Allow concurrent modify.	1,5
2	Update, allow concurrent update.	2,6
3	Modify exclusive.	none
4	Modify, allow concurrent read.	6
5	Read, allow concurrent modify.	1,5
6	Read, allow concurrent modify.	6 and either 2, one 4, or 8
7	Read, exclusive.	none
8	Read, allow concurrent read.	6,8

## DBOPEN

The table in Appendix B summarizes the results of multiple access to the same data base. If a data base cannot be opened successfully in a particular mode, this table can be used to determine the problem and to select an alternate mode.

*status*

is the name of a ten-word array in which TurboIMAGE returns status information about the procedure. If the procedure executes successfully, the status array contents are:

Word	Contents
1	Condition word is 0.
2	User class number, 0 to 63 (or a 64 if data base creator with ";" password).
3	Current word size of the DBG.
4	Word size of the DBU.
5-10	Information about the current procedure call and its results. This same information is returned for all TurboIMAGE procedures if an error occurs. It is described in Appendix A with the summary of condition words.

Table 5-19 lists the contents of Word 1 when the procedure does not succeed.

## Discussion

A process may concurrently use the data base through independent, unique access paths by issuing as many as 63 calls to DBOpen and specifying different base arrays in each call. Subsequent calls to other TurboIMAGE procedures must use the appropriate base array so that the correct base id is used.

The data base activity controlled on one access path relates to that controlled on other access paths in the same way the data base activity of one process relates to that of another. The access modes established by each DBOpen call must be compatible but otherwise the activity controlled by each access path and the pointers maintained by it are completely independent. The only exception to this access path independence relates to locking. If a process makes a lock request on one access path it cannot issue a lock on another access path unless the program has multiple RIN capability (CAP=MR) or first calls DBUNLOCK to release the locks on the first access path.

If the data base is enabled for logging, and the program calls DBOpen in one of modes 1-4, then TurboIMAGE will attempt to access a logfile using the MPE OPENLOG intrinsic. OPENLOG will succeed only if the following have been completed:

1. A valid log identifier and log password have been set into the data base root file using the DBUTIL >>SET command,
2. A corresponding system log process has been initiated by the console operator to handle any calls to the logging system.

If OPENLOG fails, DBOpen will also fail and return an appropriate error condition. If OPENLOG succeeds, DBOpen will cause a log record to be written which includes such information as time, date, user name, user program, mode, and security class. (Refer to Appendix E for a full description of log record contents and formats.)

A process is logging if it successfully opens a data base in one of modes 1-4, and the data base is enabled for logging. A program does not log if it opens in one of modes 5-8, or if the data base is not enabled for logging.

If the data base is enabled for Intrinsic Level Recovery (ILR), by using the DBUTIL >>ENABLE command, the first DBOpen for the data base also opens the ILR log file associated with the data base. At this time, DBOpen performs the following steps:

1. Opens the ILR log file and allocates an extra data segment (the ILCB or Intrinsic Level Control Block) to be used for Intrinsic Level Recovery and run time storage.
2. Verifies that the ILR log file matches the data base root file; to match, the ILR log file must have the same name as the root file with two ASCII zeros added to the end. For example, if the root file is called ORDERS, the associated ILR log file is called ORDERS00. In addition, the ILR log file must have the same creation date as the root file and the same last access date.
3. Checks whether a prior system failure interrupted a DBPUT or a DBDELETE to the data base. If so, TurboIMAGE performs the Intrinsic Level Recovery by removing any change made by the incomplete DBPUT or DBDELETE.
4. Checks whether the data base is opened for read-only access. If so, TurboIMAGE closes the ILR log file for that user and releases the extra data segment. Otherwise the ILCB remains allocated because it is used by DBPUT and DBDELETE for storage of blocks.

# DBOPEN

Table 5-19. DBOPEN Condition Word Values

FILE SYSTEM AND MEMORY MANAGEMENT FAILURES:	-1	FOPEN intrinsic failure.
	-2	FCLOSE failure.
	-3	FREADDIR failure.
	-4	FREADLABEL failure.
	-9	GETDSEG failure.
CALLING ERRORS:	-11	Bad <i>base</i> parameter.
	-21	Bad password.
	-31	Bad <i>mode</i> .
	-32	Unobtainable <i>mode</i> .
	-90	Root file bad: Unrecognized state: % octal integer.
	-91	Bad root modification level.
	-92	Data base not created.
	-94	Data base bad: Was being modified with output deferred, may not be accessed in mode <i>decimal integer</i> .
	-95	Data base bad: Creation was in process (create again).
	-96	Data base bad: Erase was in process (erase again).
COMMUNICATIONS ERRORS:	-97	Data base bad: ILR enable in process (enable again).
	-98	Data base bad: ILR disable in process (disable again).
	-60	Illegal file equation on root file.
	-100	DSOPEN failure.
	-101	DSCLOSE failure.
	-102	DSWRITE failure.
	-103	Remote stack too small.
	-104	Remote system does not support TurboIMAGE.
LOGGING SYSTEM FAILURES:	-105	MPE intrinsic GETDSEG failure on remote HP3000.
	-106	Remote data inconsistent.
	-107	DS procedure call error.
	-110	OPENLOG intrinsic failure.
	-111	WRITELOG intrinsic failure.



Table 5-19. DBOPEN Condition Word Values (Continued)

INTRINSIC LEVEL RECOVERY FILE ERRORS:	-163	Cannot initialize ILR log file:file system error <i>decimal integer</i> .
	-164	Cannot initialize ILR log header: file system error <i>decimal integer</i> .
	-170	Cannot open ILR log file: file system error <i>decimal integer</i> .
	-171	Cannot close ILR log file: file system error <i>decimal integer</i> .
	-172	Cannot read ILR log file: file system error <i>decimal integer</i> .
	-180	ILR log file invalid - internal file name does not match root file.
	-181	ILR log file invalid - internal group name does not match root file.
	-182	ILR log file invalid - internal account name does not match root file.
	-183	ILR log file invalid - internal creation date does not match root file.
	-184	ILR log file invalid - internal last access date does not match root file.
	-185	Cannot get extra data segment of size %XXXXXX for ILR.
	-200	Data Base Language not system supported.
	-201	Native Language Support not installed.
	-202	MPE Native Language Support error <i>nn</i> returned by NLINFO.
EXCEPTIONAL CONDITIONS:	60	Data base access disabled.
	61	This data base opened more than 63 times by the same process.
	62	DBG full.
	63	Bad DBG.
	64	PCBX data segment area full.
	66	The current DBG for the data base does not appear correct (TurboIMAGE internal error).
		Consult Appendix A for more information about these conditions and Appendix B for results of multiple access.

# DBPUT

## INTRINSIC NUMBER 407

Adds new entries to a manual master or detail data set. The data base must be open in access mode 1, 3, or 4.

### Syntax

DBPUT,*base,dset,mode,status,list,buffer*

### Parameters

- base*

is the name of the array used as the *base* parameter when opening the data base. The first word of the array must contain the *base id* returned by DBOPEN. (Refer to DBOPEN for more information about *base id*.)
- dset*

is the name of an array containing the left-justified name of the data set to which the entry is to be added or is an integer referencing the data set by number. The data set name may be 16 characters long or, if shorter, terminated by a semicolon or a blank ( $\Delta$ ), for example: CUSTOMER; or SALES $\Delta$ .
- mode*

must be an integer equal to 1.
- status*

is the name of a ten-word array in which TurboIMAGE returns status information about the procedure. If the procedure executes successfully, the status array contents are:

Word	Contents
1	Condition word is 0.
2	Word length of logical entry in buffer array.
3-4	Doubleword record number of new entry.
5-6	Doubleword count of number of entries in chain. If master data set, chain is synonym chain. If detail data set, chain is current chain of new entry.
7-8	If master, doubleword record address of predecessor on synonym chain. If detail, doubleword record number of predecessor on current detail chain.
9-10	If detail, doubleword record number of successor on current chain. If master, doubleword zero.

Table 5-21 lists the contents of Word 1 when the procedure does not succeed.

*list*

is the name of an array containing an ordered set of data item identifiers; names or numbers. The new entry contains values supplied in the buffer array for data items in the *list* array. Search or sort items defined for the entry must be included in the *list* array. Fields of unreferenced items are filled with binary zeros.

The *list* array can contain a left-justified set of data item names, separated by commas and terminated by a semicolon or blank. No embedded blanks are allowed and no name can appear more than once. Example: ACCOUNT, LAST-NAME, CITY, STATE,;

When referencing by number, the first word of the *list* array is an integer *n* that is followed by *n* single positive integers identifying unique data item numbers. Example: 4 1 10 3 16 lists for the four data item numbers 1, 10, 3, and 16.

The *list* specifies data items for which values are supplied in the buffer array, and is saved internally by TurboIMAGE as the current list for the data set. The current list is unchanged until a different list is specified in a subsequent call to DBGET, DBPUT, or DBUPDATE for the same access path and data set.

Some special list constructs are allowed. These are described in Table 5-20 and illustrated in the SPL programs in Section 6. List processing is a relatively high overhead operation which may be shortened in subsequent calls by using the asterisk construct to specify that the current list is to be used. Be sure a current list exists before using the asterisk construct, or a null list is assumed.

*buffer*

is the name of an array containing data item values to be added. The values are concatenated in the same order as their data item identifiers in the list array. The number of words for each value must correspond to the number required by its type; for example, 12 values must be 2 words long.

Table 5-20. Special *list* Parameter Constructs

CONSTRUCT	<i>list</i> ARRAY CONTENTS	PURPOSE
Empty	0; or 0Δ or ; or Δ (Note: Zero must be ASCII)	Request no data transfer.
Empty Numeric	0 ( <i>n</i> , length of data item identifier list, is zero)	Request no data transfer.
Asterisk	*; or *Δ	Requests procedure to use previous <i>list</i> and apply it to same data set. This construct saves TurboIMAGE processing time, especially if more than one or two items are being dealt with. If "*" is used to define the list in the first call to DBGET and DBPUT, TurboIMAGE will treat it as a zero.
Commercial At-Sign	@; or @Δ	Requests procedure to use all data items of the data set in the order of their occurrence in the entry.  (Note: Δ indicates blank.)

## Discussion

When adding entries to master data sets the following rules apply:

- The data set must be a manual master.
- The search item must be referenced in the list array and its value in the buffer array must be unique in relation to other entries in the master.
- There must be space in the master set to add an entry.
- The order of data item values in the new entry is determined by the set definition in the schema and not by the order of the items' occurrence in the list and buffer arrays.
- Unreferenced data items are filled with binary zeros.
- The caller must have a lock on the data set or the data base if the data base is opened in access mode 1.

Table 5-21. DBPUT Condition Word Values

FILE SYSTEM AND MEMORY MANAGEMENT FAILURES:	-1 -3 -4	FOPEN intrinsic failure. FREADDIR failure. FREADLABEL failure.
CALLING ERRORS:	-11 -12  -14 -21 -23 -24 -31 -51 -52 -53	Bad <i>base</i> parameter. No lock covering entry to be added. (DBOPEN mode 1 only.) Illegal intrinsic in current access mode. Bad data set reference. Data set not writable. Data set is an automatic master. Bad <i>mode</i> . Bad <i>list</i> length. Bad <i>list</i> or bad <i>item</i> . Missing search or sort item.
COMMUNICATIONS ERRORS:	-102 -106 -107	DSWRITE failure. Remote data inconsistent. DS procedure call error.
LOGGING SYSTEM FAILURES:	-111	WRITELOG intrinsic failure.
EXCEPTIONAL CONDITIONS:	16 43 62 63 1xx 2xx 3xx	Data set full. Duplicate search item. DBG full. Bad DBG. Missing chain head for path number <i>xx</i> . Full chain for path number <i>xx</i> . Full master for path number <i>xx</i> .
		Refer to Appendix A for more information about these conditions.

## DBPUT

When adding entries to detail data sets the following rules apply:

- The data set must have free space for the entry.
- If the data base is opened in access mode 1, the caller must have a lock covering the entry to be added.
- All search and sort items defined for the entry must be referenced in the list array.
- Each related manual master data set must contain a matching entry for the corresponding search item value. If any automatic master does not have a matching entry, it must have space to add one. This addition occurs automatically.
- The order of data item values in the new entry is determined by the set definition in the schema and not by the order of the items' occurrence in the list and buffer arrays.
- Unreferenced data items are filled with binary zeros.
- The new entry is linked into one chain for each search item, or path, defined according to the search item value. It is linked; to the end of chains having no sort items and into its sorted position according to the collating sequence of the sort item values in the chain. If two or more entries have the same sort item value, their position in the chain is determined by the values of the items following the sort item in the entry.

The position of an entry on a sorted chain is determined by a backward search of the chain beginning at the last entry. The position is maintained by logical pointers rather than physical placement in the file.

- Maintains proper Native Language collating sequence for chain sorting.

The record in which the new data entry is placed becomes the current record for the data set. The forward and backward pointers reflect the new entry's position. Refer to the description of status words 7 through 10.

If the process is logging, a call to DBPUT will cause a log record to be written, which includes such information as the time, date, user identification number, and a copy of the new record to be added.

Relinquishes the locks acquired by all previous calls to DBLOCK. Redundant calls are ignored. If the calling process has the same data base open multiple times, only those locks put into effect for the specified access path are unlocked.

### Syntax

DBUNLOCK, *base*, *dset*, *mode*, *status*

### Parameters

<i>base</i>	is the name of the array used for the <i>base</i> parameter when opening the data base. The first word of the array must contain the <i>base id</i> returned by DBOPEN.
<i>dset</i>	is currently unused. Use the DUMMY variable as recommended at the beginning of this section or any <i>dset</i> array used for other procedures.
<i>mode</i>	must be an integer equal to 1.
<i>status</i>	is the name of a ten-word array in which TurboIMAGE returns status information about the procedure. If the procedure executes successfully, the status array contents are:

Word	Contents
1	Condition word is 0.
2	Number of lock descriptors released by this call. Each data set lock or data base lock is counted as one descriptor.
3-4	Reserved for internal use.
5-10	Information about the procedure call and its results. Refer to Appendix A for more information on condition words.

Table 5-22 lists the contents of Word 1 when the procedure does not succeed.

# DBUNLOCK

Table 5-22. DBUNLOCK Condition Word Values

CALLING ERRORS:	-11 -31	Bad <i>base</i> parameter. Bad <i>mode</i> .
COMMUNICATIONS ERRORS:	-102 -106 -107	DSWRITE failure. Remote data inconsistent. DS procedure call error.
EXCEPTIONAL CONDITIONS:	63	Bad DBG.
		Appendix A contains more information about these conditions.



Modifies values of data items in the entry residing at the current record address of a specified data set. Search and sort item values cannot be modified. The data base must be open in access mode 1, 2, 3, or 4. The update will always be carried out correctly against the latest version of the data, regardless of modifications that may be made by other users.

### Syntax

DBUPDATE, *base*, *dset*, *mode*, *status*, *list*, *buffer*

### Parameters

*base* is the name of the array used as the *base* parameter when opening the data base. The first word of the array must contain the *base id* returned by DBOPEN. (Refer to DBOPEN for more information about *base id*.)

*dset* is the name of an array containing the left-justified name of the data set to be read or is an integer referencing the data set by number. The data set name may be 16 characters long or, if shorter, terminated by a semicolon or blank.

*mode* must be an integer equal to 1.

*status* is the name of a ten-word array in which TurboIMAGE returns status information about the procedure. If the procedure operates successfully, the status array contents are:

Word	Contents
1	Condition word is 0.
2	Word length of the values in buffer.
3-10	Same doubleword values set by preceding procedure call which positioned the data set at the current entry.

Table 5-23 lists the contents of Word 1 when the procedure does not succeed.

*list* is the name of an array containing an ordered set of data item identifiers, either names or numbers. Values supplied in the buffer array replace the values of data items occupying the same relative position in the *list* array. The user class established when the data base is opened must allow at least read access to all the items included in the *list* array.

## DBUPDATE

If the corresponding buffer array values are the same as the current data item values, the *list* array can include data items the user can read but is not permitted to alter, as well as search and sort items. This feature permits reading and updating with the same *list* array contents as well as search and sort items.

The list array may contain a left-justified set of data item names, separated by commas and terminated by a semicolon or blank. No embedded blanks are allowed and no name may appear more than once.

When referencing by number, the first word of the *list* array is an integer *n* followed by *n* unique data item numbers (one-word positive integers).

The *list* not only specifies the data items to be updated immediately but is saved internally by TurboIMAGE as the current list for this data set. The current list is unchanged until a different list is specified in a subsequent call to DBGET, DBPUT, or DBUPDATE for the same access path and data set.

Some special list constructs are allowed. These are described in Table 5-20 with the DBPUT procedure. List processing is a relatively high overhead operation which may be shortened substantially in subsequent calls by using the asterisk construct to specify that the current list is to be used.

*buffer*

is the name of an array containing concatenated values to replace the values of data items occupying the same relative position in the list array. The number of words for each value must correspond to the number of words required by its type multiplied by the sub-item count. Search and sort items must not be included in this update list.

## Discussion

Before performing an update for a data base opened in access mode 1, TurboIMAGE verifies that locks are in effect to cover the data entry both before and after it is modified.

The current record number, forward and backward pointers are unchanged. (Refer to the description of status words 3 through 10.)

If the process is logging, a call to DBUPDATE will cause a log record to be written, which includes such information as the time, date, user identification number, and a copy of both the old and new data item values.

**Table 5-23. DBUPDATE Condition Word Values**

<b>FILE SYSTEM AND MEMORY MANAGEMENT FAILURES:</b>	-1 -3 -4	FOPEN intrinsic failure. FREADDIR failure. FREADLABEL failure.
<b>CALLING ERRORS:</b>	-11 -12  -14 -21 -31 -51 -52	Bad <i>base</i> parameter. No locks cover entry to be updated. (DBOPEN mode 1 only.) Illegal intrinsic in current access mode. Bad data set reference. Bad <i>mode</i> . Bad <i>list</i> length. Bad <i>list</i> or bad <i>item</i> .
<b>COMMUNICATIONS ERRORS:</b>	-102 -106 -107	DSWRITE failure. Remote data inconsistent. DS procedure call error.
<b>LOGGING SYSTEM FAILURES:</b>	-111	WRITELOG intrinsic failure.
<b>EXCEPTIONAL CONDITIONS:</b>	17 41 42 62 63	No entry. Critical item. Read only item. DBG full. Bad DBG.
		Appendix A contains more information about these conditions.

# HOST LANGUAGE ACCESS

SECTION

6

This section is divided into six separate discussions, each covering the use of TurboIMAGE with a specific programming language: COBOL, FORTRAN, Pascal, SPL, BASIC, and RPG.

The examples in each language are designed to illustrate simply and directly the way TurboIMAGE procedures are called. They are not intended as modules of the best way to code the task which is illustrated since this will vary with the application requirements and an individual programmer's coding methods.

A knowledge of the programming language is assumed. If you have questions about the language itself, consult the appropriate language manual:

- COBOL/3000 Reference Manual
- FORTRAN Reference Manual
- System Programming Language Reference Manual
- Pascal/3000 Reference Manual
- BASIC Interpreter Reference Manual
- BASIC/3000 Compiler Reference Manual
- RPG/3000 Compiler Reference and Application Manual

All examples presented in this section perform operations on the ORDERS data base. Figures 2-5 and 2-6 in Section 2 and Figure 3-5 in Section 3 should be consulted if questions about the data base structure arise in relation to the examples.

# COBOL

To illustrate the use of TurboIMAGE procedures through COBOL programs, sample lines of code that perform a specific task are given. The TurboIMAGE procedure calling parameters are described by the way they are defined in the data division and their value when the procedure is called or, in some cases, after it is executed. All parameters must start on word boundaries.

The BASE-NAME record is described only in the first two examples. Once the data base has been opened and the data base identifier has been moved to the first word as shown in the ADD ENTRY example, it remains the same for all subsequent calls illustrated.

The DB-STATUS record is defined in the same way for all examples but its content varies depending upon which procedure is called and the results of that procedure. The DB-STATUS record is defined as:

```
05  DB-STATUS.          PIC S9999 COMP.
    05 CONDITION-WORD    PIC S9999 COMP.
    05 STAT1             PIC S9(9) COMP.
    05 STAT2-3          PIC S9(9) COMP.
    05 STAT4-5          PIC S9(9) COMP.
    05 STAT6-7          PIC S9(9) COMP.
    05 STAT8-9          PIC S9(9) COMP.
```

The DUMMY parameter appears as a reminder when a parameter is not used by a procedure performing the task being illustrated. DUMMY can be defined as PIC S9999 COMP.

When GOTO ASK-FOR-IP appears in the code, it indicates that the program continues and prompts the user for further instructions, for example, it may request the type of data base operation the user wants to perform.

## Open Data Base

```
PROCEDURE DIVISION.
  FIRST-PARAGRAPH-NAME.
    CALL "DBOPEN" USING BASE-NAME, PASSWORD, MODE3, DB-STATUS.
    IF CONDITION-WORD NOT = 0 DISPLAY "DBOPEN-FAIL"
    CALL "DBEXPLAIN" USING DB-STATUS STOP RUN.
```

PARAMETER	DEFINITION	VALUE
BASE-NAME	PIC X(10)	"ΔΔORDERS;"
PASSWORD	PIC X(8)	"DO-ALL;Δ" or "DO-ALLΔΔ"
MODE3	PIC 9999 COMP	3

In this example, the ORDERS data base is opened in access mode 3 with the password DO-ALL that establishes user class number 18. The value of PASSWORD may be specified in the data division or it may be requested from the application program user and moved into PASSWORD. If the password is fewer than 8 characters it must be followed by a blank or semi-colon. In this program, the first word of the DB-STATUS array, CONDITION-WORD, is tested and if it is not zero a failure message is printed and the DBEXPLAIN procedure is executed.

## Add Entry

```

CALL "DBPUT" USING BASE-NAME, DATA-SET-P, MODE1,
  DB-STATUS, ALL-ITEMS, PR-BUFFER.
IF CONDITION-WORD = 43 DISPLAY "DUPLICATE STOCK NUMBER"
  GO TO ASK-FOR-IP.
IF CONDITION-WORD = 16 DISPLAY "DATA SET FULL"
  GO TO ASK-FOR-IP.
IF CONDITION-WORD = -23 DISPLAY "CANNOT ADD WITH CURRENT PASSWORD"
  GO TO ASK-FOR-IP.
IF CONDITION-WORD NOT = 0 GO TO DISPLAY-STATUS.

```

PARAMETER	DEFINITION	VALUE
BASE-NAME	PIC X(8)	"23 ORDERS;" <i>(data base identifier in first word)</i>
DATA-SET-P	PIC X(8)	"PRODUCT;"
MODE1	PIC 9999 COMP	1
ALL-ITEMS	PIC X(2)	"@"
PR-BUFFER		
STOCK-NO	PIC X(8)	"7474Z74Z"
DESCRIPTN	PIC X(20)	"ORANGE CRATE" (16 A's)

This sample code adds a data entry to the **PRODUCT** manual master data set. Note that the first word of **BASE-NAME** now contains the data base id. **ALL-ITEMS** contains an "@" sign indicating that **PR-BUFFER** contains a value for all items in the data entry. The values for the **STOCK#** and **DESCRIPTION** data items are concatenated in **PR-BUFFER**.

A program may be designed to prompt for both the data set name and the data item values that are moved into **PR-BUFFER** and added to the data set. In the example, the condition word of the status array is tested for a value of 43, indicating that an entry with the search item value 7474Z74Z already exists in the data set, or 16, indicating that the data set is full. If the user class is not in the data set write class list, a condition word of -23 is returned.

If an entry is to be added to a detail set, the program may first check to see if the required entries exist in the manual masters linked to the detail set. Values must be provided for all search items and the sort item, if one is defined, of a detail data set entry.

# COBOL

## Read Entry (Serially)

```
READ-NEXT.  
  CALL "DBGET" USING BASE-NAME, DATA-SET-C, MODE2, DB-STATUS,  
    LIST-OF-ITEMS, CU-BUFFER, DUMMY.  
  IF CONDITION-WORD = 11 PERFORM REWIND  
    GO TO READ-NEXT.  
  IF CONDITION-WORD = -21 DISPLAY "NO READ ACCESS TO DATA"  
    GO TO ASK-FOR-IP.  
  IF CONDITION-WORD NOT = 0 GO TO DISPLAY-STATUS.
```

(Process entry and decide whether or not to continue.)

PARAMETER	DEFINITION	VALUE
DATA-SET-C	PIC X(10)	"CUSTOMER;"
MODE2	PIC 9999 COMP	2
LIST-OF-ITEMS	PIC X(80)	"ACCOUNT,FIRST-NAME,LAST-NAME;.."
CU-BUFFER		
ACCT	PIC 9(9) COMP	12345678
F-NAME	PIC X(10)	"GEORGE" <i>values which are read.</i>
L-NAME	PIC X(16)	"PADERSON"

To read the next entry of the CUSTOMER data set, a mode of 2 is used. This directs the DBGET procedure to perform a forward serial read. In the example, LIST-OF-ITEMS contains the names of three data items. After DBGET returns to the calling program, CU-BUFFER contains the values shown. If an end-of-file is encountered, the condition word is set to 11. In this case, the routine rewinds the data set and tries the read again. A rewind routine is shown later in the examples of the DBCLOSE procedure. The rewind reinitializes the current record pointer so that the next request for a forward serial read will read the first entry in the data set. If the condition word -21 is returned, the user's password does not grant read access to data.

The DUMMY variable merely signifies that the argument parameter is not used with mode 2.

## Read Entry (Directly)

```

CALL "DBGET" USING BASE-NAME, DATA-SET-I, MODE4, DB-STATUS,
  ALL-ITEMS, IN-BUFFER, RECORD-NUMBER.
IF CONDITION-WORD = 12 OR = 13 DISPLAY "INCORRECT RECORD NUMBER"
  GO TO DISPLAY-STATUS.
IF CONDITION-WORD = 17 DISPLAY "RECORD CONTAINS NO DATA ENTRY"
  GO TO DISPLAY-STATUS.
IF CONDITION-WORD NOT = 0 DISPLAY "DBGET FAILURE"
  GO TO DISPLAY-STATUS.

```

PARAMETER	DEFINITION	VALUE
DATA-SET-I	PIC X(10)	"INVENTORY;"
MODE4	PIC 9999 COMP	4
ALL-ITEMS	PIC X(2)	"@"
RECORD-NUMBER	PIC 9(9) COMP	33
IN-BUFFER		
STOCK-NO-I	PIC X(8)	"3333A33A"
QTY	PIC 9(9) COMP	452
SUPPLIER	PIC X(16)	"H & S SURPLUS "
UNIT-COST	PIC S9(7) COMP-3	0000349E (3495 in 8 nibbles)
LASTSHIPDATE	PIC X(6)	"841214"
BINNUM	PIC X(2)	"03"

The code in this example reads all data items of the entry in record number 33 of the INVENTORY data set using a directed read, mode 4. The program may have saved the record number while reading down the chain of all data entries with STOCK# equal to 3333A33A looking for the latest LASTSHIPDATE. It then reads all data items of the entry which has the desired last shipping date. It is more efficient to read it directly than to search down the chain again.

If the record number is less than 1, the condition word is set to 12. If it is greater than the highest numbered record in the data set, the condition word is set to 13. The condition word is 17 if the record contains no data entry.



# COBOL

## Read Entry (Calculated)

```
CALL "DBGET" USING BASE-NAME, DATA-SET-P, MODE7, DB-STATUS,  
  LIST-OF-ITEMS, DESCRIPTN, STOCK-SEARCH.  
IF CONDITION-WORD = 17 DISPLAY "NO SUCH STOCK NUMBER"  
  GO TO ASK-FOR-IP.  
IF CONDITION-WORD = -21 DISPLAY "NO SUCH READ ACCESS TO DATA"  
  GO TO ASK-FOR-IP.  
IF CONDITION-WORD NOT = 0 GO TO DISPLAY-STATUS.
```

PARAMETER	DEFINITION	VALUE
DATA-SET-P	PIC X(8)	"PRODUCT;"
MODE7	PIC 9999 COMP	7
LIST-OF-ITEMS	PIC X(80)	"DESCRIPTION;"
PR-BUFFER		
STOCK-NO	PIC X(8)	- - -
DESCRIPTN	PIC X(20)	"CLIPBOARD"
STOCK-SEARCH	PIC X(8)	"2222B22B"

To locate the **PRODUCT** data set entry which has **STOCK#** search item value of **2222B22B**, a calculated read is used. The mode is 7 and the item to be read is **DESCRIPTION**. After **DBGET** returns control to the calling program, the description of stock number **2222B22B** is in the **DESCRIPTN** buffer. If no entry exists with **STOCK#** equal to **2222B22B**, the condition word is 17. If the user does not have read access to the **DESCRIPTION** data item, condition word -21 is returned.

**Read Entry (Backward Chain)**

```
CALL "DBFIND" USING BASE-NAME, DATA-SET-S, MODE1, DB-STATUS,  
    ITEM-NAME, ITEM-VALUE.  
IF CONDITION-WORD = 17 DISPLAY "NO PURCHASES ON THAT DATE"  
    GO TO ASK-FOR-IP.  
IF CONDITION-WORD = -21 OR -52  
    DISPLAY "PASSWORD OR ACCESS MODE DOES NOT GRANT ACCESS"  
    GO TO ASK-FOR-IP.  
IF CONDITION-WORD NOT = 0 DISPLAY "DBFIND FAILURE"  
    GO TO DISPLAY-STATUS.  
NEXT-IN-CHAIN.  
CALL "DBGET" USING BASE-NAME, DATA-SET-S, MODE6, DB-STATUS,  
    ALL-ITEMS, SA-BUFFER, DUMMY.  
IF CONDITION-WORD = 14 DISPLAY "NO MORE PURCHASES ON THIS DATE"  
    GO TO NEXT-ACCOUNT.  
IF CONDITION-WORD = 0 GO TO REPORT-SALES.  
.  
.  
REPORT-SALES.
```

(Routine to print sales information)

```
GO TO NEXT-IN-CHAIN.
```

# COBOL

PARAMETER	DEFINITION	VALUE
DATA-SET-S	PIC X(6)	"SALES;"
MODE1	PIC 9999 COMP	1
ITEM-NAME	PIC X(12)	"PURCH-DATE;"
ITEM-VALUE	PIC X(6)	"841214"
MODE6	PIC 9999 COMP	6
ALL-ITEMS	PIC X(2)	"@"
SA-BUFFER		
ACCOUNT-S	PIC (9) COMP	12345678
STOCK-NO-S	PIC X(8)	"2222B22B"
QUANTITY	PIC 9999 COMP	3
PRICE	PIC 9(9) COMP	425
TAX	PIC 9(9) COMP	25
TOTAL	PIC 9(9) COMP	450
PURCH-DATE	PIC X(6)	"841214"
DELIV-DATE	PIC X(6)	"841220"

*sample values  
read from one  
entry in chain*

First the DBFIND procedure is called to determine the location of the first and last entries in the chain. The call parameters include the detail data set name, the name of the detail search item used to define a path with the DATE-MASTER data set, and the search item value 841214 of both the master entry containing the chain head and the detail entries making up the chain. If no entry in the DATE-MASTER has a search item value 841214, the condition word will be 17. If the user's password or access mode does not allow read access to the data, condition word -21 or -52 is returned.

If the DBFIND procedure executes successfully, a call to the DBGET procedure with a mode parameter of 6 reads the last entry in the chain. Successive calls to DBGET with the same mode read the next-to-last entry and so forth until the first entry in the chain has been read. A subsequent call to DBGET returns condition word 14, indicating the beginning of the chain has been reached and no more entries are available. If an entry has been successfully read, the program executes the REPORT-SALES routine and prints the information. It then goes to the NEXT-IN-CHAIN routine and reads another entry.

If no entries exist in the chain, the condition word is also 14.

## Update Entry

```
CALL "DBGET" USING BASE-NAME, DATA-SET-C, MODE7, DB-STATUS,
    ITEM-NAME, ADDRESS-VALUE, ACCT-SEARCH.
```

(Determine if entry successfully read, print current address, and prompt for new address.)

```
CALL "DBUPDATE" USING BASE-NAME, DATA-SET-C, MODE1, DB-STATUS,
    ITEM-NAME, ADDRESS-VALUE.
IF CONDITION-WORD = 42 DISPLAY "NOT ALLOWED TO ALTER THIS ITEM"
GO TO ASK-FOR-IP.
IF CONDITION-WORD NOT = 0 GO TO DISPLAY-STATUS.
```

PARAMETER	DEFINITION	VALUE
DATA-SET-C	PIC X(10)	"CUSTOMER;"
MODE7	PIC 9999 COMP	7
MODE1	PIC 9999 COMP	1
ACCT-SEARCH	PIC 9(9) COMP	12345678
ITEM-NAME	PIC X(16)	"STREET-ADDRESS;"
ADDRESS-VALUE	PIC X(26)	"12 SUTTON PLACE "

In order to update an entry it must first be located. In this example, the entry is located by using a calculated DBGET to read the STREET-ADDRESS item in the CUSTOMER data set. The entry is located by using the ACCOUNT search item with a value of 12345678. If the read is successful, the current address is printed and the application program user is prompted for the new address which is moved into ADDRESS-VALUE. The DBUPDATE routine is then called to alter the STREET-ADDRESS data item in the entry.

If the current user class number does not allow this item to be altered or the access mode does not allow updates to take place, the condition word 42 is returned.

A null list can be used when calling DBGET to locate an entry to be updated.

# COBOL

## Delete Entry

(Locate appropriate entry as with DBGET.)

```
CALL "DBDELETE" USING BASE-NAME, DATA-SET-C, MODE1, DB-STATUS.  
IF CONDITION-WORD = 44  
    DISPLAY "SALES ENTRIES EXIST, CANNOT DELETE CUSTOMER"  
    GO TO ASK-FOR-IP.  
IF CONDITION-WORD = -23 DISPLAY "PASSWORD DOES NOT ALLOW DELETE"  
    GO TO ASK-FOR-IP.  
IF CONDITION-WORD NOT = 0 GO TO DISPLAY-STATUS.
```

PARAMETER	DEFINITION	VALUE
DATA-SET-C	PIC X(10)	"CUSTOMER;"

Before an entry can be deleted, the current record of the data set must be that of the entry to be deleted. This record may be located by calling DBGET. In this example, the program may have requested the account number of the customer to be deleted and then used a calculated DBGET to locate the appropriate entry. If entries in the SALES data set exist which have the same account number as the entry to be deleted, the condition word is set to 44 and the entry is not deleted. Condition word -23 indicates that the user does not have the capability of deleting an entry from the CUSTOMER data set.

A null list can be used when calling DBGET to locate an entry to be deleted.

## Lock and Unlock (Data Base)

```

CALL "DBLOCK" USING BASE-NAME, DUMMY, MODE2, DB-STATUS.
IF CONDITION-WORD = 20 DISPLAY "DATA BASE IS BUSY. TRY AGAIN LATER."
  GO TO CLOSE.
IF CONDITION-WORD = 0 GO TO USE-BASE.
DISPLAY "DBLOCK FAILURE" GO TO DISPLAY-STATUS.
  USE-BASE.
.
.
.
CALL "DBUNLOCK" USING BASE-NAME, DUMMY, MODE1, DB-STATUS.
IF CONDITION-WORD NOT = 0 DISPLAY "DBUNLOCK FAILURE"
  GO TO DISPLAY-STATUS.

```

PARAMETER	DEFINITION	VALUE
MODE2	PIC 9999 COMP	2
MODE1	PIC 9999 COMP	1

In this example the program calls DBLOCK to lock the data base. Since mode 2 is used, the program must check the condition word when DBLOCK returns control to verify that the data base is locked. If it is locked the condition word is 0; if it is busy the condition word is 20.

If the data base is successfully locked, the program goes to the USE-BASE routine. After the data base operations have been completed, the program unlocks the data base by calling the DBUNLOCK procedure.

An example of data entry locking appears in the sample COBOL program, Figure 6-1.

# COBOL

## Request Data Item Information

```
CALL "DBINFO" USING BASE-NAME, ITEM-NAME, MODE, DB-STATUS,  
INFO-BUFFER.  
IF CONDITION-WORD NOT = 0 DISPLAY "DBINFO FAILURE"  
GO TO DISPLAY-STATUS.
```

PARAMETER	DEFINITION	VALUE
ITEM-NAME	PIC X(12)	"PURCH-DATE;"
MODE	PIC 9999 COMP	102
INFO-BUFFER		
NAM-TYP	PIC X(18)	"PURCH-DATE X"
SUB-LENG	PIC 9999 COMP	6
SUB-COUNT	PIC 9999 COMP	1

The procedure call in this example obtains information about the PURCH-DATE data item by specifying mode 102. The item name and type are returned in the first 9 words of INFO-BUFFER and the sub-item length and sub-item count in words 10 and 11.

## Rewind Data Set

```
REWIND  
CALL "DBCLOSE" USING BASE-NAME, DATA-SET-C. MODE3, DB-STATUS.  
IF CONDITION-WORD NOT = 0 DISPLAY "DBCLOSE FAILURE"  
GO TO DISPLAY-STATUS.  
.  
.  
.
```

PARAMETER	DEFINITION	VALUE
DATA-SET-C	PIC X(10)	"CUSTOMER;"
MODE3	PIC 9999 COMP	3

To rewind the CUSTOMER data set, a call to DBCLOSE is made with mode equal to 3. The dynamic status information in the DBU for CUSTOMER is reset, including the current record number. If a serial read request encounters an end-of-file, this call resets the current record to the beginning of the data set and another serial read request will read the first entry in the data set.

## Close Data Base

```

CLOSE.
  CALL "DBCLOSE" USING BASE-NAME, DUMMY, MODE1,DB-STATUS.
  IF CONDITION-WORD NOT = 0 CALL "DBEXPLAIN" USING DB-STATUS.
  STOP RUN.

```

PARAMETER	DEFINITION	VALUE
MODE1	PIC 9999 COMP	1

## Print Error

```

DISPLAY-STATUS.
  CALL "DBEXPLAIN" USING DB-STATUS.
  GO TO CLOSE.

```

The call to DBEXPLAIN prints a message on the \$STDLIST device which interprets the contents of the DB-STATUS array. This routine may be used while debugging the application if a procedure call fails.

## Move Error to Buffer

```

CALL "DBERROR" USING DB-STATUS, ERR-BUFFER, LENGTH.

```

PARAMETER	DEFINITION	VALUE
ERR-BUFFER	PIC X(72)	"DATA BASE IN USE"
LENGTH	PIC 9999 COMP	16

In this example, a call to DBERROR has returned one of the messages appropriate when the condition word is equal to -1. The length of the message is 16 bytes as indicated by the value of LENGTH returned by DBERROR.



# COBOL

## Sample Cobol Program

Figure 6-1 contains a sample data base application, a program to update the inventory records, which is coded in COBOL. The program is called **RECEIVE** and updates on-hand quantities and adjusts unit costs in the **INVENTORY** data set of the **ORDERS** data base. The data base is opened in mode 2. Sample output from **RECEIVE** is illustrated in Figure 6-2.

Locking is performed at the data entry level to ensure that two users do not attempt to modify the same data entry simultaneously. Also, presuming that transactions against the data base are being logged to a logfile, **DBBEGIN** and **DBEND** are used to mark the beginning and end of the transaction. This technique should always be used to delimit a multiple-step logged transaction. It is used in this example to illustrate the proper order of calling the procedures, as outlined in Section 5.

```

*****
*   THIS PROGRAM ILLUSTRATES THE USE OF COBOL CALLS TO IMAGE.   *
*   IT USES THE DATA BASE "ORDERS", ACCESSING THE DETAIL DATA SET *
*   "INVENTORY" TO UPDATE THE ON-HAND QUANTITY AND UNIT COST TO *
*   REFLECT THE RECEIPT OF A NEW SHIPMENT. NOTICE THAT THE PASS- *
*   WORD USED WAS "BUYER" SINCE THE TWO FIELDS BEING CHANGED HAVE *
*   2 AS A WRITE CLASS. NOTICE ALSO THAT THE DATA BASE IS OPENED *
*   IN MODE 2, WHICH IS ADEQUATE FOR READING AND UPDATING THE TWO *
*   FIELDS INVOLVED, WHILE ALLOWING OTHERS TO ACCESS THE DATA BASE *
*   CONCURRENTLY. THE USER CAN ONLY MODIFY ENTRIES WHOSE STOCK# *
*   AND SUPPLIER HAVE ALREADY BEEN ESTABLISHED IN THE PRODUCT *
*   MANUAL MASTER AND SUP-MASTER MANUAL MASTER RESPECTIVELY. *
*   TO KEEP THIS EXAMPLE SIMPLE THE "ACCEPT" VERB HAS BEEN USED *
*   FOR ENTERING TRANSACTIONS. *
* *
*   ENTRY LOCKS ARE USED TO ENSURE THAT TWO USERS DO NOT ATTEMPT *
*   TO MODIFY SIMULTANEOUSLY AN EXISTING ENTRY BASED ON ITS OLD *
*   CONTENTS. CALLS TO DBBEGIN AND DBEND ARE USED TO INDICATE *
*   THE BEGINNING AND END OF A LOGICAL TRANSACTION ON THE *
*   LOGFILE FOR THE RECOVERY SYSTEM IF IT IS EXECUTED. *
*****

```

IDENTIFICATION DIVISION.

PROGRAM-ID. RECEIVE.

DATE-COMPILED.

FRI, DEC 7, 1984, 11:30 AM.

ENVIRONMENT DIVISION.

DATA DIVISION.

WORKING-STORAGE SECTION.

01 IMAGE-FIELDS.

05	BASE-NAME	PIC	X(10)	VALUE	" ORDERS;"
05	LIST-OF-ITEMS	PIC	X(20)	VALUE	"ONHANDQTY,UNIT-COST;"
05	PREVIOUS-LIST	PIC	X(02)	VALUE	"*;"
05	PASSWORD	PIC	X(10)	VALUE	"BUYER;"
05	MODE1	PIC	S9(4)	COMP	VALUE 1.
05	MODE2	PIC	S9(4)	COMP	VALUE 2.
05	MODE5	PIC	S9(4)	COMP	VALUE 5.
05	TEXT-LENGTH	PIC	S9(4)	COMP.	
05	SEARCH-ITEM	PIC	X(08)	VALUE	"STOCK#; "
05	DATA-SET	PIC	X(10)	VALUE	"INVENTORY;"
05	TEXT1	PIC	X(18)	VALUE	"BEGIN STOCK UPDATE".
05	TEXT2	PIC	X(16)	VALUE	"END STOCK UPDATE".
05	DB-STATUS.				
10	CONDITION-WORD	PIC	S9(4)	COMP.	
10	STAT1	PIC	S9(4)	COMP.	

Figure 6-1. Inventory Update Program

# COBOL

```

        10  STAT2-3          PIC  S9(9)  COMP.
        10  STAT4-5          PIC  S9(9)  COMP.
        10  STAT6-7          PIC  S9(9)  COMP.
        10  STAT8-9          PIC  S9(9)  COMP.
01  LOCK-DESCRIPTOR.
    05  NUM-OF-LOCKS          PIC  S9(4)  COMP  VALUE 1.
    05  LOCK-STOCK-ENTRY.
        10  WORD-LENGTH      PIC  S9(4)  COMP  VALUE 22.
        10  LOCK-SET-NAME     PIC  X(16)  VALUE "INVENTORY;    ".
        10  LOCK-ITEM-NAME    PIC  X(16)  VALUE "STOCK#;       ".
        10  RELOP             PIC  X(02)  VALUE "=".
        10  LOCK-VALUE        PIC  X(08).
01  ACCEPT-FIELDS.
    05  STOCK-NO              PIC  X(08).
    05  NEW-QUANTITY          PIC  9(08).
    05  NEW-COST              PIC  9(08).
01  EDIT-FIELDS.
    05  EDITED-COST           PIC  $$,$$$,$$$,$$$$.99.
    05  EDITED-VALUE          PIC  $$,$$$,$$$,$$$$.99.
    05  EDITED-QTY            PIC  Z(8)9.
01  IP-BUFFER.
    05  ON-HAND-QTY           PIC  9(9)   COMP.
    05  UNIT-COST             PIC  S9(7)  COMP-3.
```

## PROCEDURE DIVISION.

### 10-FIRST-PARAGRAPH-NAME.

```

    CALL "DBOPEN" USING BASE-NAME, PASSWORD, MODE2, DB-STATUS.
    IF CONDITION-WORD NOT = 0
        DISPLAY "DBOPEN-FAIL"
        PERFORM 90-DISPLAY-STATUS
    STOP RUN.
```

### 20-ASK-FOR-IP.

```

    MOVE SPACES TO STOCK-NO.
    DISPLAY "ENTER 8 CHARACTER STOCK NUMBER OR TYPE EXIT".
    ACCEPT STOCK-NO FREE.
    IF STOCK-NO = "EXIT"
        GO TO FINISH.
    PERFORM 30-FIND-STOCK-RECORD.
    IF CONDITION-WORD = 17 OR = 15
        DISPLAY "NO SUCH STOCK NUMBER"
        PERFORM 70-UNLOCK
        GO TO 20-ASK-FOR-IP.
    DISPLAY "NOW ENTER QUANTITY RECEIVED - ".
    ACCEPT NEW-QUANTITY FREE.
```

Figure 6-1. Inventory Update Program (Continued)

```
    DISPLAY "NOW ENTER UNIT COST IN CENTS - ".
    ACCEPT NEW-COST FREE.
    PERFORM 60-UPDATE-STOCK.
    PERFORM 50-DISPLAY-NEW-STOCK.
    DISPLAY " " DISPLAY " "
    GO TO 20-ASK-FOR-IP.

30-FIND-STOCK-RECORD.
    MOVE STOCK-NO TO LOCK-VALUE.
    CALL "DBLOCK" USING BASE-NAME, LOCK-DESCRIPTOR, MODE5,
        DB-STATUS.
    IF CONDITION-WORD NOT = 0
        DISPLAY "LOCK FAILED"
        PERFORM 90-DISPLAY-STATUS
        GO TO FINISH.
    CALL "DBFIND" USING BASE-NAME, DATA-SET, MODE1, DB-STATUS
        SEARCH-ITEM, STOCK-NO.
    IF CONDITION-WORD = 0
        PERFORM 40-GET-STOCK-RECORD
    ELSE
        IF CONDITION-WORD NOT = 17
            DISPLAY "FIND FAIL"
            PERFORM 90-DISPLAY-STATUS
            GO TO FINISH.

40-GET-STOCK-RECORD.
    CALL "DBGET" USING BASE-NAME, DATA-SET, MODE5, DB-STATUS,
        LIST-OF-ITEMS, IP-BUFFER, STOCK-NO.
    IF CONDITION-WORD NOT = 0 AND NOT = 15
        DISPLAY "GET FAIL"
        PERFORM 90-DISPLAY-STATUS
        GO TO FINISH.

50-DISPLAY-NEW-STOCK.
    MOVE ON-HAND-QTY TO EDITED-QTY.
    COMPUTE EDITED-COST = UNIT-COST / 100.
    COMPUTE EDITED-VALUE = ON-HAND-QTY * UNIT-COST / 100.
    DISPLAY "NEW ON HAND QUANTITY = ", EDITED-QTY.
    DISPLAY "NEW UNIT COST = ", EDITED-COST.
    DISPLAY "NEW STOCK VALUE = ", EDITED-VALUE.

60-UPDATE-STOCK.
    COMPUTE UNIT-COST = (UNIT-COST * ON-HAND-QTY + NEW-QUANTITY
        * NEW-COST) / (ON-HAND-QTY + NEW-QUANTITY).
    COMPUTE ON-HAND-QTY = ON-HAND-QTY + NEW-QUANTITY.
    MOVE -18 TO TEXT-LENGTH
```

Figure 6-1. Inventory Update Program (Continued)

```

CALL "DBBEGIN" USING BASE-NAME, TEXT1, MODE1, DB-STATUS,
    TEXT-LENGTH.
IF CONDITION-WORD NOT = 0
    DISPLAY "DBBEGIN FAIL"
    PERFORM 90-DISPLAY-STATUS
    GO TO FINISH.
CALL "DBUPDATE" USING BASE-NAME, DATA-SET, MODE1, DB-STATUS,
    PREVIOUS-LIST, IP-BUFFER.
IF CONDITION-WORD NOT = 0
    DISPLAY "UPDATE FAIL"
    PERFORM 90-DISPLAY-STATUS
    GO TO FINISH.
MOVE -16 TO TEXT-LENGTH
CALL "DBEND" USING BASE-NAME, TEXT2, MODE1, DB-STATUS,
    TEXT-LENGTH.
IF CONDITION-WORD NOT = 0
    DISPLAY "DBEND FAIL"
    PERFORM 90-DISPLAY-STATUS
    GO TO FINISH.
PERFORM 70-UNLOCK.

70-UNLOCK.
CALL "DBUNLOCK" USING BASE-NAME, LOCK-DESCRIPTOR, MODE1,
    DB-STATUS.
IF CONDITION-WORD NOT = 0
    DISPLAY "UNLOCK FAIL"
    PERFORM 90-DISPLAY-STATUS
    GO TO FINISH.

90-DISPLAY-STATUS.
CALL "DBEXPLAIN" USING DB-STATUS.

FINISH.
CALL "DBCLOSE" USING BASE-NAME, DATA-SET, MODE1, DB-STATUS.
IF CONDITION-WORD NOT = 0
    DISPLAY "DATA BASE CLOSE FAILED"
    PERFORM 90-DISPLAY-STATUS.
STOP RUN.

```

**Figure 6-1. Inventory Update Program (Continued)**

```
:RUN RECEIVE

ENTER 8 CHARACTER STOCK NUMBER OR TYPE EXIT
4397D13P
NO SUCH STOCK NUMBER
ENTER 8 CHARACTER STOCK NUMBER OR TYPE EXIT
12345678
NO SUCH STOCK NUMBER
ENTER 8 CHARACTER STOCK NUMBER OR TYPE EXIT
6650DD2S
NO SUCH STOCK NUMBER
ENTER 8 CHARACTER STOCK NUMBER OR TYPE EXIT
6650D22S
NOW ENTER QUANTITY RECEIVED -
100
NOW ENTER UNIT COST IN CENTS -
150

NEW ON HAND QUANTITY =      306
NEW UNIT COST =           $2.44
NEW STOCK VALUE =        $746.63

ENTER 8 CHARACTER STOCK NUMBER OR TYPE EXIT
6650D22S
NOW ENTER QUANTITY RECEIVED -
5000
NOW ENTER UNIT COST IN CENTS -
1500

NEW ON HAND QUANTITY =      5306
NEW UNIT COST =           $14.27
NEW STOCK VALUE =        $75,716.62

ENTER 8 CHARACTER STOCK NUMBER OR TYPE EXIT
2457A11C
NOW ENTER QUANTITY RECEIVED -
10000000
NOW ENTER UNIT COST IN CENTS -
4000

NEW ON HAND QUANTITY =    11001345
NEW UNIT COST =           $50.31
NEW STOCK VALUE =        $553,477,666.95

ENTER 8 CHARACTER STOCK NUMBER OR TYPE EXIT
EXIT
END OF PROGRAM
```

Figure 6-2. Sample RECEIVE Execution

# FORTRAN

In the FORTRAN examples which follow, all variables are integer unless declared otherwise. The DUMMY parameter is an integer and appears when a parameter is not used by the procedure for the task which is being performed.

The code at statement 9900 closes the data base. It is not included in each example but is implied to be there.

Since TurboIMAGE requires that the parameters be at word addresses, they must be integer arrays equivalenced to character strings if necessary. For example, the BASE integer array is 5 words long and is equivalenced to the CS1 character string which is 10 bytes long. This array contains the name of the ORDERS data base preceded by one word of blanks to which a data base identifier within the control block is moved when the data base is opened.

## Open Data Base

```
PROGRAM FTIM1
  IMPLICIT INTEGER (A-Z)
  DIMENSION STATUS(10),PASSWORD(5),BASE(5)
  CHARACTER*10 CS1,CS2
  EQUIVALENCE (BASE(1),CS1),(PASSWORD(1),CS2)
  CS1 = "  ORDERS;"
  CS2 = "      "
  DISPLAY "ENTER PASSWORD "
  ACCEPT CS2
  DISPLAY "ENTER ACCESS MODE (1-8) "
  ACCEPT MODE
  CALL DBOPEN (BASE,PASSWORD,MODE,STATUS)
  IF (STATUS(1).NE.0) GOTO 9300
  DISPLAY "DATA BASE OPENED"
  GOTO 9900
9300 DISPLAY "DBOPEN FAILURE"
9310 CALL DBEXPLAIN (STATUS)
.
.
.
```

In this example the ORDERS data base is opened in the access mode entered by the application user and with the user class number corresponding to the password entered. For example, the access mode may be 3 and the password DO-ALL. Since TurboIMAGE parameters must have word addresses, the character string must be equivalenced to an integer array before being passed to the TurboIMAGE procedure.

If the procedure fails, the first word of STATUS is an integer other than zero. In this case, the sample program prints a message and executes DBEXPLAIN to display status information.

If the password is less than 10 characters long, it must be followed by a semicolon or blank. Therefore, the character string CS2 is initialized to 8 blanks.

# **Add Entry**

```

PROGRAM FTIM2
  IMPLICIT INTEGER (A-Z)
  DIMENSION STATUS(10),PASSWORD(5),BASE(5)
  DIMENSION DSTEP(4),PRBUFF(14)
  CHARACTER DESCRIPN*20,ALLITEMS*2
  CHARACTER*10 CS1,CS2,CS3,STOCKNO
  EQUIVALENCE (BASE(1),CS1), (PASSWORD(1),CS2)
  EQUIVALENCE (DSTEP(1),CS3), (PRBUFF(1),STOCKNO),
C      (PRBUFF(5),DESCRIPN),(AI,ALLITEMS)
  CS1 = "  ORDERS;"
  CS2 = "  "
  CS3 = "PRODUCT;"
  MODE1=1
  ALLITEMS = "@;"
  .
  .
  .
  (code to open data base in access mode 1, 3, or 4 and prompt for data item values)
  .
  .
  .
  CALL DBPUT (BASE,DSTEP,MODE1,STATUS,AI,PRBUFF)
  IF (STATUS(1).NE.43) GOTO 120
  DISPLAY "DUPLICATE STOCK NUMBER"
  GOTO 110
120 IF (STATUS(1).NE.16) GOTO 130
  DISPLAY "DATA SET FULL"
  GOTO 9900
130 IF (STATUS(1).NE.-23) GOTO 140
  DISPLAY "PASSWORD DOES NOT ALLOW ADDING ENTRIES"
  GOTO 9900
140 IF (STATUS(1).NE.0) GOTO 160
  DISPLAY "NEW PRODUCT HAS BEEN ENTERED"
  GOTO 9900
160 DISPLAY "DBPUT FAILURE"
  GOTO 9310
9300 DISPLAY "DBOPEN FAILURE"
9310 CALL DBEXPLAIN (STATUS)
  .
  .
  .

```

This sample code adds a data entry to the PRODUCT manual master data set. The first word of the BASE array now contains the number of the data base identifier. ALLITEMS contains an "@" sign indicating that PRBUFF contains a value for all items in the data entry. The values for the STOCK# and DESCRIPTION data items are concatenated in PRBUFF, for example, 7474Z74ZORANGE CRATE△△△△△△△△.



## FORTRAN

A typical application will prompt for the data item values which are moved into PRBUFF and added to the data set. In this example, the condition word of the STATUS array is tested for a value of 43, indicating that an entry with search item value 7474Z74Z already exists in the data set, or 16, indicating that the data set is full. If the user's password does not allow entries to be added, condition word -23 is returned.

If an entry is to be added to a detail set, a value must be provided for all search items and the sort item if one is defined. The program may first check to see if the required entries exist in the manual masters linked to the detail data set, or it can check for condition word 1xx after attempting to add the detail entry.

If the access is mode 1, the data base must be locked before an entry can be added.

### Read Entry (Serially)

```
PROGRAM FTIM3
IMPLICIT INTEGER (A-Z)
DIMENSION STATUS(10),PASSWORD(5),BASE(5)
DIMENSION DSETC(5),LIST(15),CBUFF(15)
CHARACTER CS4*10,CS5*30,FNAME*10,LNAME*14
CHARACTER*10 CS1,CS2
EQUIVALENCE (DSETC(1),CS4), (LIST(1),CS5), (CBUFF(3),FNAME),
C      (CBUFF(8),LNAME)
EQUIVALENCE (BASE(1),CS1), (PASSWORD(1),CS2)
CS1 = "  ORDERS;"
CS2 = "      "
CS4 = "CUSTOMER; "
CS5 = "ACCOUNT,FIRST-NAME, LAST-NAME; "
DUMMY = 1
MODE2 = 2
.
.
.
(Code to open data base.)
.
.
.
200 CALL DBGET (BASE,DSETC,MODE2,STATUS,LIST,CBUFF,DUMMY)
IF (STATUS(1).NE.11) GOTO 210
DISPLAY "CONTINUE"
ACCEPT I
IF (I.EQ.0) GOTO 9900
.
.
.
```

(Code to determine whether to continue, if so, rewind data set.)

```

210  IF (STATUS(1).NE.-21.AND.STATUS(1).NE.-52) GOTO 220
      DISPLAY "YOU DO NOT HAVE ACCESS TO DATA"
      GOTO 9900
220  IF (STATUS(1).EQ.0) GOTO 230
      DISPLAY "DBGET FAILURE"
      GOTO 9310
230  WRITE (6,*) FNAME,LNAME,CBUFF(1)
      GOTO 200
9300  DISPLAY "DBOPEN FAILURE"
9310  CALL DBEXPLAIN (STATUS)
      .
      .
      .

```

To read the next entry of the CUSTOMER data set, a mode of 2 is used. This directs the DBGET procedure to perform a forward serial read. In the example, the LIST array contains the names of three data items. After DBGET returns to the calling program, CBUFF contains values such as:

CBUFF(1)—CBUFF(2)	12345678 (double integer)
CBUFF(3)—CBUFF(7)	GEORGEΔΔΔΔ
CBUFF(8)—CBUFF(14)	PADERSONΔΔΔΔΔΔ

If an end-of-file is encountered the condition word is set to 11. In this case, the routine rewinds the data set and tries the read again. A rewind routine is shown later in the examples of the DBCLOSE procedure. The rewind reinitializes the current record pointer so that the next request for a forward serial read reads the first entry in the data set. If the user does not have read access to the data items, condition word -21 is returned.

The DUMMY variable signifies that the *argument* parameter is not used with mode 2.

# FORTRAN

## Read Entry (Directly)

```
PROGRAM FTIM11
IMPLICIT INTEGER (A-Z)
DIMENSION STATUS (10),PASSWORD(5),BASE(5)
DIMENSION DSTEP(4), PRBUFF(14)
CHARACTER*10 CS1,CS2,CS3,STOCKNO
CHARACTER DESCRIPN*20, ALLITEMS*2
EQUIVALENCE (BASE(1),CS1),(PASSWORD(1),CS2)
EQUIVALENCE (DSTEP(1),CS3), (PRBUFF(1),STOCKNO),
C      (PRBUFF(5),DESCRIPN), (AI,ALLITEMS)
INTEGER*4 RECNO
CS1 = "  ORDERS;"
CS2 = "          "
CS3 = "PRODUCT;"
ALLITEMS = "@;"
MODE4 = 4
```

(Code to open data base.)

```
210  DISPLAY "REC"
      ACCEPT RECNO
      IF (RECNO.EQ.0) GOTO 9900
      CALL DBGET (BASE, DSTEP, MODE4, STATUS, AI, PRBUFF, RECNO)
      IF (STATUS(1).EQ.12.OR.STATUS(1).EQ.13) GOTO 280
      IF (STATUS(1).NE.17) GOTO 270
      DISPLAY "RECORD CONTAINS NO DATA ENTRY"
      GOTO 210
270  IF (STATUS(1).EQ.0) GOTO 290
      DISPLAY "DBGET FAILURE"
      GOTO 9310
280  DISPLAY "INCORRECT RECORD NUMBER"
      GOTO 210
290  DISPLAY DESCRIPN
      GOTO 210
9300  DISPLAY "DBOPEN FAILURE"
9310  CALL DBEXPLAIN (STATUS)
```

The code in this example reads all data items of the entry in the specified record number of the **PRODUCT** data set using a directed read, mode 4. If the condition word is equal to 12 or 13, the record number is not within the range of records in the file. If the condition word is 17 the record contains no entry.

<b>NOTE</b>
-------------

This is not the normal method for using directed reads but is used to simplify the example.

# **Read Entry (Calculated)**

```

PROGRAM FTIM4
IMPLICIT INTEGER (A-Z)
DIMENSION STATUS(10),PASSWORD(5),BASE(5)
DIMENSION DSETP(4),LISTA(6),PRBUFF(10),STOCKSRCH(4)
CHARACTER*10 CS1,CS2,CS3,CS7
CHARACTER DESCRIPN*20, CS6*12
EQUIVALENCE (DSETP(1),CS3),(PRBUFF(1),DESCRIPN),
C      (LISTA(1),CS6),(STOCKSRCH(1),CS7)
EQUIVALENCE (BASE(1),CS1),(PASSWORD(1),CS2)
CS1 = "  ORDERS;"
CS2 = "          "
CS3 = "PRODUCT;"
CS6 = "DESCRIPTION;"
MODE7 = 7
.
.
.
(Code to open data base.)

20  DISPLAY "STOCK NUMBER"
    ACCEPT CS7
    CALL DBGET (BASE,DSETP,MODE7,STATUS,LISTA,PRBUFF(1),STOCKSRCH)
    IF (STATUS(1).NE.17) GOTO 300
    DISPLAY "NO SUCH STOCK NUMBER"
    GOTO 20
300  IF (STATUS(1).NE.-21) GOTO 310
    DISPLAY "PASSWORD DOES NOT GRANT ACCESS TO DATA"
    GOTO 9900
310  IF (STATUS(1).EQ.0) GOTO 320
    DISPLAY "DBGET FAILURE"
    GOTO 9310
320  (Code to use data from entry just read.)
.
.
.
9310 CALL DBEXPLAIN (STATUS)
.
.
.

```

A calculated read is used to locate the PRODUCT data set entry which has the STOCK# search item value entered in CS7. The mode is 7 and the item to be read is DESCRIPTION. After DBGET returns control to the calling program, the description for the specified stock number is in DESCRIPN. If no entry exists with STOCK# equal to the specified value, the condition word is 17. If the user does not have read access to the DESCRIPTION data item, the condition word is -21.

# FORTRAN

## Read Entry (Forward Chain)

```
PROGRAM FTIM5
IMPLICIT INTEGER (A-Z)
DIMENSION STATUS(10),PASSWORD(5),BASE(5)
DIMENSION DSETS(3), INAME(6), IVAL(3), SABUFF(19)
CHARACTER CS8*6, CS9*12, CS10*6, SASTOCK*8, PURCHDT*8,
C      DELIVDT*8, ALLITEMS*2
CHARACTER*10 CS1,CS2
EQUIVALENCE (BASE(1),CS1),(PASSWORD(1),CS2)
EQUIVALENCE (DSETS(1),CS8), (INAME(1),CS9), (IVAL(1),CS10),
C      (SABUFF(1),ACCTS), (SABUFF(3),SASTOCK),
C      (SABUFF(7),QTY), (SABUFF(8),PRICE),
C      (SABUFF(10),TAX), (SABUFF(12),TOTAL),
C      (SABUFF(14),PURCHDT),(SABUFF(17),DELIVDT),(AI,ALLITEMS)
INTEGER*4 ACCTS, PRICE, TAX, TOTAL
CS1 = "  ORDERS;"
CS2 = "      "
CS8 = "SALES;"
CS9 = "PURCH-DATE; "
CS10 = "760314" ← Program would normally prompt for this value.
ALL ITEMS = "@;"
MODE1 = 1
MODE5 = 5
```

(Code to open data base.)

```
CALL DBFIND (BASE,DSETS,MODE1,STATUS,INAME,IVAL)
IF (STATUS(1).NE.17) GOTO 345
DISPLAY "NO PURCHASES ON THAT DATE."
GOTO 9900
345 IF (STATUS(1).NE.-21.AND.STATUS(1).NE.-52) GOTO 355
DISPLAY "PASSWORD OR ACCESS MODE DOES NOT GRANT ACCESS"
GOTO 9900
355 IF (STATUS(1).EQ.0) GOTO 360
DISPLAY "DBFIND FAILURE"
GOTO 9310
360 CALL DBGET (BASE, DSETS, MODE5, STATUS, AI, SABUFF, DUMMY)
IF (STATUS(1).NE.15) GOTO 365
DISPLAY "NO MORE PURCHASES ON THIS DATE"
GOTO 9900
365 IF (STATUS(1).NE.0) GOTO 380
```

(Code to use sales information from entry, for example, in a report.)

```
GOTO 360
380 DISPLAY "DBGET FAILURE"
GOTO 9310
9300 DISPLAY "DBOPEN FAILURE"
9310 CALL DBEXPLAIN (STATUS)
```

First the DBFIND procedure is called to determine the location of the first and last entries in the chain. The call parameters include the detail data set name, the name of the detail search item used to define a path with the DATE-MASTER data set, and the search item value 760314 of both the master entry containing the chain head and the detail entries making up the chain. If no entry in the DATE-MASTER has a search item value of 760314, the condition word will be 17. If the user's password or access mode does not grant read access to the data set or data items, condition word -21 or -52 is returned.

If the DBFIND procedure executes successfully, a call to the DBGET procedure with a mode parameter of 5 reads the first entry in the chain if one exists. Subsequent calls to DBGET with the same mode read the succeeding entries to the chain until the last entry in the chain has been read. If the condition word is 15, the end of the chain has been reached and no more entries are available, or no entries exist in the chain.

If an entry is successfully read the program uses the information and then returns to statement 360 to read another entry in the chain. After an entry has been read the SABUFF array contains information like this:

SABUFF(1)	- SABUFF(2)	ACCTS	12345678	(doubleword integer)
SABUFF(3)	- SABUFF(6)	SASTOCK	2222B22B	(character string)
SABUFF(7)		QTY	3	(integer)
SABUFF(8)	- SABUFF(9)	PRICE	425	(doubleword integer)
SABUFF(10)	- SABUFF(11)	TAX	25	(doubleword integer)
SABUFF(12)	- SABUFF(13)	TOTAL	450	(doubleword integer)
SABUFF(14)	- SABUFF(16)	PURCHDT	760314	(character string)
SABUFF(17)	- SABUFF(19)	DELIVDT	760320	(character string)

# FORTRAN

## Update Entry

```
PROGRAM FTIM6
IMPLICIT INTEGER (A-Z)
DIMENSION STATUS(10),PASSWORD(5),BASE(5)
DIMENSION DSETC(5), INAME2(8), ADDVAL(13)
INTEGER*4 ACCTSRCH
CHARACTER CS4*10, CS11*16, ADDSTRING*26
CHARACTER*10 CS1,CS2
EQUIVALENCE (BASE(1),CS1), (PASSWORD(1),CS2)
EQUIVALENCE (DSETC(1),CS4), (INAME2(1),CS11),
C      (ADDVAL(1),ADDSTRING)
CS1 = " ORDERS;"
CS2 = " "
CS4 = "CUSTOMER; "
CS11 = "STREET-ADDRESS; "
MODE1 = 1
MODE7 = 7
ACCTSRCH = 12345678
```

(code to open data base in access mode 1, 2, 3, or 4)

```
CALL DBGET (BASE,DSETC,MODE7,STATUS,INAME2,ADDVAL,ACCTSRCH)
```

(Code to determine if read is successful and print current address.)

```
DISPLAY "NEW ADDRESS"
ACCEPT ADDSTRING
CALL DBUPDATE (BASE,DSETC,MODE1,STATUS,INAME2,ADDVAL)
IF (STATUS(1).NE.42) GOTO 420
DISPLAY "YOU ARE NOT ALLOWED TO ALTER THIS ITEM"
GOTO 9900
DISPLAY "DBUPDATE FAILURE"
GOTO 9310
440 DISPLAY "ADDRESS CHANGED"
GOTO 9900
9300 DISPLAY "DBOPEN FAILURE"
9310 CALL DBEXPLAIN (STATUS)
```

Before an entry can be updated it must be located. In this example, the entry is located by using a calculated DBGET to read the STREET-ADDRESS item in the CUSTOMER data set. The entry is located by using the ACCOUNT search item with a value of 12345678. If the read is successful, the current address is printed and the application program user is prompted for the new address which is moved into ADDRESS-VALUE. The DBUPDATE routine is then called to alter the STREET-ADDRESS data item in the entry.

If the current user class number does not allow this item to be altered or the access mode does not allow updates to take place, the condition word 42 is returned.

A null list can be used with DBGET to locate an entry to be updated.

## Delete Entry

```

PROGRAM FTIM7
  IMPLICIT INTEGER (A-Z)
  DIMENSION STATUS(10),PASSWORD(5),BASE(5)
  DIMENSION DSETC(5), INAME2(8), ADDVAL(13)
  INTEGER*4 ACCTSRCH
  CHARACTER CS4*10, CS11*16, ADDSTRING*26
  CHARACTER*10 CS1,CS2
  EQUIVALENCE (BASE(1),CS1),(PASSWORD(1),CS2)
  EQUIVALENCE (DSETC(1),CS4), (INAME2(1),CS11),
C      (ADDVAL(1),ADDSTRING)
  CS1 = "  ORDERS;"
  CS2 = "      "
  CS4 = "CUSTOMER; "
  CS11 = "; "
  MODE1 = 1
  MODE7 = 7
  .
  (Code to open data base in access mode 1, 3, or 4.)
  .
20  DISPLAY "ACCOUNT OR ZERO TO TERMINATE"
    ACCEPT ACCTSRCH
    IF (ACCTSRCH.EQ.0) GOTO 9900
    CALL DBGET (BASE,DSETC,MODE7,STATUS,INAME2,DUMMY,ACCTSRCH)
    IF (STATUS(1).NE.0) GOTO 9310
    CALL DBDELETE (BASE,DSETC,MODE1,STATUS)
    IF (STATUS(1).NE.44) GOTO 530
    DISPLAY "SALES ENTRIES EXIST, CUSTOMER CANNOT BE DELETED"
    GOTO 20
530  IF (STATUS(1).NE.-23) GOTO 540
    DISPLAY "PASSWORD DOES NOT GRANT ACCESS TO DATA SET"
    GOTO 9900
540  IF (STATUS(1).EQ.0) GOTO 560
    DISPLAY "DBDELETE FAILURE"
    GOTO 9310
560  DISPLAY "CUSTOMER ENTRY DELETED"
    GOTO 20
9300 DISPLAY "DBOPEN FAILURE"
9310 CALL DBEXPLAIN (STATUS)
  .

```

Before an entry can be deleted, the current record of the data set must be that of the entry to be deleted. This record may be located by calling DBGET. In this example, the program may have requested the account number of the customer to be deleted and then used a calculated DBGET to locate the appropriate entry. If entries in the SALES data set exist which have the same account number as the entry to be deleted, the condition word is set to 44 and the entry is not deleted.

A null list can be used with DBGET to locate an entry to be deleted.

If the access mode is 1, the data base must be locked before the entry is deleted.



# FORTRAN

## Lock and Unlock (Data Base)

```
PROGRAM FTIM8
IMPLICIT INTEGER (A-Z)
DIMENSION STATUS(10),PASSWORD(5),BASE(5)
CHARACTER*10 CS1,CS2
EQUIVALENCE (BASE(1),CS1),(PASSWORD(1),CS2)
CS1 = "  ORDERS;"
CS2 = "      "
.
.
(Code to open data base in access mode 1 or 5.)
.
.
MODE1 = 1
MODE2 = 2
CALL DBLOCK (BASE,DUMMY,MODE2,STATUS)
IF (STATUS(1).NE.20) GOTO 640
DISPLAY "DATA BASE IS BUSY.  TRY AGAIN LATER."
GOTO 9900
640 IF (STATUS(1).EQ.0) GOTO 680
DISPLAY "DBLOCK FAILURE"
GOTO 9310
680 (Code to use data base.)
.
.
CALL DBUNLOCK (BASE,DUMMY,MODE1,STATUS)
IF (STATUS(1).EQ.0) GOTO 9900
DISPLAY "DBUNLOCK FAILURE"
GOTO 9310
9300 DISPLAY "DBOPEN FAILURE"
9310 CALL DBEXPLAIN (STATUS)
.
.
.
```

In this example, the program calls DBLOCK to lock the data base. Since mode 2 is used, the program must check the condition word when DBLOCK returns control to verify that the data base is locked and the calling program has exclusive access. If this is so, the condition word is 0; if it is busy the condition word is 20.

If the data base is successfully locked, the program performs the necessary data base operations and then unlocks the data base by calling the DBUNLOCK procedure. In the example the programs terminates after unlocking the data base.

## Lock (Data Entries)

```

        PROGRAM FTIM8A
        IMPLICIT INTEGER (A-Z)
        DIMENSION STATUS(10),PASSWORD(5),BASE(5),IP(40)
        CHARACTER*10 CS1,CS2,VAL
        CHARACTER*2 RELOP
        CHARACTER*16 SETNAME,ITEMNAME
        EQUIVALENCE (BASE(1),CS1),(PASSWORD(1),CS2),
C              (NUM,IP(1)),
C              (LENGTH,IP(2)),
C              (SETNAME,IP(3)),
C              (ITEMNAME,IP(11)),
C              (RELOP,IP(19)),
C              (VAL,IP(20))
        CS1 = "  ORDERS;"
        CS2 = "      "
        MODE1 = 1
        .
        .
        .
(Code to open data base.)
        .
        .
        .
        NUM=1
        LENGTH=22
        SETNAME="INVENTORY      "
        ITEMNAME="STOCK#          "
        RELOP="= "
        VAL="6650D22S"
        CALL DBLOCK (BASE,IP,5,STATUS)
640 IF (STATUS(1).EQ.0) GOTO 680
        DISPLAY "DBLOCK FAILURE"
        GOTO 9310
680 (Code to modify the locked data entry or entries.)
        .
        .
        .
9310 CALL DBEXPLAIN (STATUS)
        .
        .
        .

```

This example illustrates locking at the data entry level. All data entries in the INVENTORY data set with a STOCK# value of 6650D22S are locked unconditionally (mode 5). If the lock request succeeds, the condition word is 0. If the DBLOCK procedure detects a calling error or an exceptional condition such as DBCB full, the DBLOCK failure message is displayed and DBEXPLAIN is called.

# FORTRAN

## Request Data Set Information

```
PROGRAM FTIM9
IMPLICIT INTEGER (A-Z)
DIMENSION STATUS(10),PASSWORD(5),BASE(5)
DIMENSION INFOBUF(8)
CHARACTER*10 CS1,CS2
EQUIVALENCE (BASE(1),CS2),(PASSWORD(1),CS2)
CS1 = "  ORDERS;"
CS2 = "          "
```

(Code to open data base.)

```

.
.
.
MODE=203
CALL DBINFO (BASE,DUMMY,MODE,STATUS,INFOBUF)
IF (STATUS(1).EQ.0) GOTO 700
DISPLAY "DBINFO FAILURE"
GOTO 9310
700 (Code to use data set numbers returned in INFOBUF.)
.
.
.
GOTO 9900
9300 DISPLAY "DBOPEN FAILURE"
9310 CALL DBEXPLAIN (STATUS)
.
.
```

The procedure call in this example obtains the numbers of the data sets available to the current user class by specifying mode 203. If the user class number is 12, after the call has been successfully executed the INFOBUF array contains:

INFOBUF(1)	4	Access to 4 data sets.
INFOBUF(2)	2	Read access to data set 2.
INFOBUF(3)	-3	Modify access to data set 3
INFOBUF(4)	-5	and data set 5.
INFOBUF(5)	6	Read and possibly update access to data set 6.

If the user class number is 8 it contains:

INFOBUF(1)	6	Access to 6 data sets.
INFOBUF(2)	-1	Modify access to data set 1.
INFOBUF(3)	2	Read access to data set 2, an automatic master.
INFOBUF(4)	-3	Modify access to all other data sets.
INFOBUF(5)	-4	
INFOBUF(6)	-5	
INFOBUF(7)	-6	

Refer to the schema in Figure 3-5 to help you interpret this procedure call in relation to the ORDERS data base.

## **Rewind Data Set**

```

PROGRAM FTIM3
IMPLICIT INTEGER (A-Z)
DIMENSION STATUS(10),PASSWORD(5),BASE(5)
DIMENSION DSETC(5),LIST(15),CBUFF(15)
CHARACTER CS4*10,CS5*30,FNAME*10,LNAME*14
CHARACTER*10 CS1,CS2
EQUIVALENCE (DSETC(1),CS4), (LIST(1),CS5), (CBUFF(3),FNAME),
C      (CBUFF(8),LNAME)
EQUIVALENCE (BASE(1),CS1),(PASSWORD(1),CS2)
CS1 = "  ORDERS;"
CS2 = "      "
CS4 = "CUSTOMER; "
.
.
.
MODE3 = 3
CALL DBCLOSE (BASE,DSETC,MODE3,STATUS)
IF (STATUS(1).EQ.0) GOTO 200
DISPLAY "DBCLOSE FAILURE"
GOTO 9310

```

To rewind the CUSTOMER data set, a call to DBCLOSE is made with mode equal to 3. The dynamic status information in the DBU for CUSTOMER is reset, including the current record number. If a serial read request encounters an end-of-file, this call resets the current record to the beginning of the data set and another serial read request will read the first entry in the data set.

## **Close Data Base**

```

9900 MODE1 = 1
      CALL DBCLOSE (BASE,DUMMY,MODE1,STATUS)
      IF (STATUS(1).EQ.0) GOTO 9980
      DISPLAY "DBCLOSE FAILURE"
      CALL DBEXPLAIN (STATUS)
9980 STOP
      END

```

This call closes the data base. It is issued after the program has completed all data base operations and before program termination.

# FORTRAN

## Print Error

```
.  
.   
.   
CALL DBEXPLAIN (STATUS)  
9980 STOP  
END
```

A call to DBEXPLAIN prints a message on the \$STDLIST device which interprets the contents of the STATUS array.

## Move Error to Buffer

```
PROGRAM FTIM10  
IMPLICIT INTEGER (A-Z)  
DIMENSION STATUS(10),PASSWORD(5),BASE(5)  
DIMENSION ERBUFF(36)  
CHARACTER ERSTRING*72  
CHARACTER*10 CS1,CS2  
EQUIVALENCE (BASE(1),CS1),(PASSWORD(1),CS2)  
EQUIVALENCE (ERBUFF(1),ERSTRING)  
CS1 = "  ORDERS;"  
CS2 = "  "  
DISPLAY "ENTER PASSWORD "  
ACCEPT CS2  
DISPLAY "ENTER ACCESS MODE (1-8) "  
ACCEPT MODE  
CALL DBOPEN (BASE,PASSWORD,MODE,STATUS)  
IF (STATUS(1).NE.0) GOTO 9300  
.   
.   
.   
9300 DISPLAY "DBOPEN FAILURE"  
9310 CALL DBERROR (STATUS,ERBUFF,LENG)  
DISPLAY ERSTRING [1:LENG]  
.   
.   
. 
```

In this example, a call to DBERROR returns one of the messages appropriate to the condition word returned by the DBOPEN procedure if it fails. For example, the message in ERSTRING may be DATA BASE OPEN IN AN INCOMPATIBLE MODE if the condition word is -1. The value of LENG in this case is 38.

Sample lines of code that perform a specified task are given below to illustrate the use of TurboIMAGE procedures through Pascal programs.

Type and variable declarations are defined at the beginning of the sample program. TurboIMAGE intrinsics must be declared for Pascal as external procedures. The procedure name is followed by the word INTRINSIC. No significant error verification is performed on the parameters by the Pascal compiler. Because TurboIMAGE parameters do not have a fixed format, calling TurboIMAGE intrinsics in Pascal is non-standard. Warning messages will be printed against procedure statements when the Pascal program is compiled.

Type declarations declare names for data structure forms that will be used in allocating variables. Type declarations may be used as parameters to procedures also. Variable declarations allocate the variables of the program. Variables are defined with precise types or forms. Pascal string literals are delimited with single quotes ( ' '). Variable name and field name are separated with a dot (.), when referenced. For example, "base\_\_name.baseid".

Table 6-1 displays TurboIMAGE type designators and sub-item lengths and the data types typically used to process them in Pascal. For more information on TurboIMAGE data item lengths and type designators refer to Section 3.

**Table 6-1. TurboIMAGE and Pascal Data Structures**

IMAGE	PASCAL TYPE
J1	-32766 .. 32767 [Subrange]
J2	Integer
I1	-32766 .. 32767 [Subrange]
I2	Integer
K1	BOOLEAN (must be either TRUE or FALSE) TRUE = odd number; FALSE = even number
K2	Integer *
P4	Packed Array [1..2] of CHAR *
P8	Packed Array [1..4] of CHAR *
R2	Real
R4	Longreal (is an extension for four-word floating point)
Zn	Packed Array [1..n] of CHAR *
Xn	Packed Array [1..n] of CHAR
	*NOTE: Type does not correspond with the TurboIMAGE type, however, storage is allocated correctly.

The following is an example of defining type declarations, variable declarations and TurboIMAGE intrinsics in the sample Pascal program in this section:

```

$TABLES ON$
$CODE_OFFSETS ON$
PROGRAM Pascal_TurboIMAGE(input,output);
LABEL 100; { Error Exit }

TYPE

  { Set up TurboIMAGE parameter data type }
  single_integer = -32768..32767;
  base_type      = RECORD
                      baseid      : PACKED ARRAY [1..2] OF char;
                      name_only   : PACKED ARRAY [1..16] OF char;
                      terminator: PACKED ARRAY [1..2] OF char;
                    END;
  password_type  = PACKED ARRAY [1..10] OF char;
  status_type    = ARRAY[1..10] OF single_integer;

```

```

ds_name_type  = PACKED ARRAY [1..12] OF char;
list_type     = PACKED ARRAY [1..80] OF char;
buffer_type1  = RECORD
    stock_no   : PACKED ARRAY [1..8] OF char;
    description: PACKED ARRAY [1..20] OF char;
END;
buffer_type2  = RECORD
    acct       : integer;
    f_name     : PACKED ARRAY [1..10] OF char;
    l_name     : PACKED ARRAY [1..16] OF char;
END;
buffer_type3  = RECORD
    stock_no_I : PACKED ARRAY [1..8] OF char;
    qty        : integer;
    supplier   : PACKED ARRAY [1..16] OF char;
    unit_cost  : real;
    lastshipdate: PACKED ARRAY [1..6] OF char;
    binnum     : PACKED ARRAY [1..2] OF char;
END;
info_buff_type = RECORD
    nam_typ    : PACKED ARRAY [1..18] OF char;
    sub_leng   : single_integer;
    sub_count  : single_integer;
    dummy1     : single_integer;
    dummy2     : single_integer;
END;
sa_buff_type  = RECORD
    account_s  : single_integer;
    stock_no_s : PACKED ARRAY [1..8] OF char;
    quantity   : single_integer;
    price      : integer;
    tax        : integer;
    total      : integer;
    purch_date : PACKED ARRAY [1..6] OF char;
    deliv_date : PACKED ARRAY [1..6] OF char;
END;
key_type      = PACKED ARRAY [1..40] OF char;
item_type     = PACKED ARRAY [1..8] OF char;

```

VAR

```

base_name      : base_type;
db_password    : password_type;
mode           : single_integer;
db_status      : status_type;
dataset_name   : ds_name_type;
item_list      : list_type;
item_name      : PACKED ARRAY [1..16] OF char;
item_value     : PACKED ARRAY [1..26] OF char;
ds_data_buff1  : buffer_type1;
ds_data_buff2  : buffer_type2;
ds_data_buff3  : buffer_type3;
repeat_prompt  : boolean;
yes_no         : PACKED ARRAY [1..3] OF char;
argument       : PACKED ARRAY [1..8] OF char;

```



## PASCAL

```
err_length      : single_integer;
err_buffer      : PACKED_ARRAY [1..80] OF char;
record_no       : integer;
stock_search    : PACKED_ARRAY [1..8] OF char;
sa_buffer       : sa_buff_type;
info_buffer     : info_buff_type;
acct_search     : integer;
```

```
PROCEDURE dbopen; INTRINSIC;
PROCEDURE dbclose; INTRINSIC;
PROCEDURE dbget; INTRINSIC;
PROCEDURE dbput; INTRINSIC;
PROCEDURE dbfind; INTRINSIC;
PROCEDURE dbexplain; INTRINSIC;
PROCEDURE dberror; INTRINSIC;
PROCEDURE dbdelete; INTRINSIC;
PROCEDURE dbupdate; INTRINSIC;
PROCEDURE dblock; INTRINSIC;
PROCEDURE dbunlock; INTRINSIC;
PROCEDURE dbinfo; INTRINSIC;
```

\* Type BOOLEAN variables must be either TRUE or FALSE (False = 0, True = 1).

## Open Data Base

```
BEGIN

{ Initialize intrinsic parameters }

prompt('ENTER DATA BASE NAME ');
readln(base_name.name_only);
base_name.baseid:= ' ';
base_name.terminator:= ' ';
db_password:= 'DO-ALL';
mode:= 3;

dbopen(base_name, db_password, mode, db_status);
IF db_status[1] <> 0 THEN
BEGIN
  writeln('DBOPEN-FAIL');
  GOTO 100;
END;
```

In the above example, the user is prompted for the data base name. The data base is then opened in mode 3 with the password DO-ALL. If the password is less than 8 characters it must be followed by a blank or semi-colon. The first word of db\_\_status is tested and if it is not zero a failure message is printed.

## Add Entry

```

dataset_name:= 'PRODUCT;';
item_list:= '@;';
mode:= 1;

prompt('ENTER STOCK-NO ');
readln(ds_data_buff1.stock_no);
prompt('ENTER DESCRIPTION ');
readln(ds_data_buff1.description);

dbput(base_name, dataset_name, mode, db_status, item_list,
      ds_data_buff1);

IF db_status[1] <> 0 THEN
BEGIN
  CASE db_status[1] OF

    43: writeln('DUPLICATE STOCK NUMBER');
    16: writeln('DATA SET FULL');
    -23: writeln('CANNOT ADD WITH CURRENT PASSWORD');

    OTHERWISE BEGIN
      writeln('DBPUT failure');
      GOTO 100;
    END;
  END; {Case}
END; {If}

```

This sample code adds a data entry to the PRODUCT manual master data set. The item\_list contains an at-sign (@) which requests TurboIMAGE to return all fields of the data set in the order defined in the data base schema. Other valid lists are the null list ('0;') which returns no data, and same list ('\*') which returns the same fields as were listed in the previous call.

The user will be prompted to enter a STOCK# and corresponding DESCRIPTION. These data items will be moved into the specified ds\_data\_buff1. In the example above, the condition word of db\_status is tested for a value of 43, indicating the entry already exists in the data set, 16, indicating that the data set is full, or -23, indicating that the add cannot be performed with the current password entry.

# PASCAL

## Read Entry (Serially)

```
mode:= 2;
item_list:= 'ACCOUNT,FIRST-NAME,LAST-NAME;';
WHILE db_status[1]= 0 DO
BEGIN
dbget( base_name, dataset_name, mode, db_status, item_list,
      ds_data_buff2, argument );

IF db_status[1] = 0 THEN
BEGIN
  { Display data }
  writeln( 'ACCT : ', ds_data_buff2.acct);
  writeln( 'F-NAME : ', ds_data_buff2.f_name);
  writeln( 'L-NAME : ', ds_data_buff2.l_name);
END;

END; { WHILE db_status }

IF db_status[1] <> 0 THEN
BEGIN
  IF db_status[1] = -21 THEN
    writeln('NO READ ACCESS TO DATA');
END;
```

To read the next entry of the CUSTOMER data set, a mode of 2 is used. This directs DBGET to perform a forward serial read. Item\_list contains the "ACCOUNT,FIRST-NAME,LAST-NAME" data items. After DBGET returns to the calling program, the ds\_data\_buff2 contains ACCT, F-NAME and L-NAME. Argument is a dummy parameter for this read mode. Condition word -21 is returned if the user's password does not grant access to read the data.

**Read Entry (Directly)**

```

mode:= 4;
record_no:= 33;
dataset_name:= 'INVENTORY;';
item_list:= '@;';

dbget( base_name, dataset_name, mode, db_status, item_list,
       ds_data_buff3, record_no );

IF db_status[1] <> 0 THEN
BEGIN
  CASE db_status[1] OF

    12,13:      writeln('INCORRECT RECORD NUMBER');
    17   :      writeln('RECORD CONTAINS NO DATA ENTRY');

    OTHERWISE   writeln('DBGET FAILURE');

  END;
END; {If db_status}

```

This example reads all data items of the entry in record number 33 of the INVENTORY data set using mode 4, directed read. The program may have saved a record number while reading down the chain looking for the latest LASTSHIPDATE (in ds\_\_data\_\_buff3).

If the condition word is set to 12 or 13 the message "INCORRECT RECORD NUMBER" will be printed. If the condition word is set to 17, "RECORD CONTAINS NO DATA ENTRY" will be printed.

# PASCAL

## Read Entry (Calculated)

```
mode:= 7;
dataset_name:= 'PRODUCT;';
stock_search:= '22B22B';
item_list:= 'DESCRIPTION;';

dbget( base_name, dataset_name, mode, db_status, item_list,
       ds_data_buff1.description, stock_search);

IF db_status[1] <> 0 THEN
BEGIN
  CASE db_status[1] OF

    17:           writeln('NO SUCH STOCK NUMBER');
    -21  :        writeln('NO SUCH READ ACCESS TO DATA');

    OTHERWISE     writeln('DBGET FAILURE');

  END; {Case}
END; {If}
```

To locate the PRODUCT data set entry which has a STOCK# search item value of 22B22B, a calculated read is used. Mode 7 is used and the item to be read is DESCRIPTION (item\_list). If 22B22B does not exist condition word 17 is returned "NO SUCH STOCK NUMBER". If the user does not have read access to this data item, condition word -21 is returned and the corresponding message is printed.

**Read Entry (Backward Chain)**

```

dataset_name:= 'SALES;';
mode:= 1;
item_name:= 'PURCH-DATE;';
item_value:= '841210';

dbfind( base_name, dataset_name, mode, db_status, item_name,
        item_value );
IF db_status[1] <> 0 THEN
BEGIN
    CASE db_status[1] OF

        17: writeln('NO PURCHASES ON THAT DATE');
        -21,-52: writeln('PASSWORD OR ACCESS MODE DOES NOT GRANT ACCESS');

        OTHERWISE writeln('DBFIND FAILURE');

    END; { CASE }

    prompt('CONTINUE? ');
    readln( yes_no );

    IF NOT( (yes_no[1]='Y') OR (yes_no[1]='y') )
    THEN GOTO 100
END;

mode:= 6;
item_list:= '@;';
WHILE db_status[1] = 0 DO
BEGIN
    dbget( base_name, dataset_name, mode, db_status, item_list,
           sa_buffer, argument {dummy} );

    IF db_status[1] = 14 THEN
        writeln('NO MORE PURCHASES ON THIS DATE');

    IF db_status[1] = 0 THEN
    BEGIN
        { Report of sales }
    END;
END;

END;

```

In the above example the DBFIND procedure is called to determine the location of the first and last entries in the chain. In this program the detail dataset\_name is SALES, the detail search item is PURCH-DATE (used to define a path with the DATE-MASTER data set), and the search item value is 841210. These call parameters belong to both the master entry containing the chain head and the detail entries making up the chain. If no entry in DATE-MASTER has the search item value of 841210, condition word 17 is returned. Condition word -21 or -52 is returned if the user's password or access mode does not allow read access to the data.

## PASCAL

If DBFIND is successful, a call to DBGET with mode 6 reads the last entry in the chain. Successive DBGET calls with the same mode read the next-to-last entry and so forth, until the first entry in the chain is read. The next call to DBGET will return the message "NO MORE PURCHASES ON THIS DATE".

### Locate and Update Entry

```
mode:= 7;
dataset_name:= 'CUSTOMER;';
item_name:= 'STREET-ADDRESS;';
item_value:= '12 SUTTON PLACE ';
acct_search:= 1220;

dbget( base_name, dataset_name, mode, db_status, item_name,
       item_value, acct_search );
IF db_status[1] <> 0 THEN
BEGIN
    writeln( 'ITEM NOT FOUND' );
    GOTO 100;
END;

{Update entry}
mode:= 1;

BEGIN
prompt ( 'ENTER NEW ADDRESS' );

dbupdate( base_name, dataset_name, mode, db_status, item_name,
          item_value );

if db_status[1] = 42 THEN
BEGIN
    writeln('NOT ALLOWED TO ALTER THIS ITEM');
    GOTO 100;
END;
```

The entry to be updated must be located first. In this example, the entry is located by using a calculated DBGET to read the STREET-ADDRESS (item\_\_name) in the CUSTOMER data set. For example, the entry is located by using the ACCOUNT item with a value of 1220. If the read is successful, the current address is printed and the user is prompted for the new address. The new address is moved into item\_\_value. The DBUPDATE routine is called to alter the STREET-ADDRESS data item in the entry.

Condition word 42 is returned if the current user class number does not allow this item to be altered, or the access mode does not allow updates to take place.

## Delete Entry

```

mode:= 1;

dbdelete( base_name, dataset_name, mode, db_status );

IF db_status[1] <> 0 THEN
BEGIN
    CASE db_status[1] OF

        44: writeln('SALES ENTRIES EXIST, CANNOT DELETE CUSTOMER');
        -23: writeln('PASSWORD DOES NOT ALLOW DELETE');

        OTHERWISE writeln('DBDELETE FAILURE');

    END; {Case}
END; {If}

```

Before an entry can be deleted, the current record of the data set must be that of the entry to be deleted. In this example, the program may have requested the account number of the customer to be deleted and then used a calculated DBGET to locate the appropriate entry. Condition word 44 is returned if the entries in the data set do not exist and the entry is not deleted. "PASSWORD DOES NOT ALLOW DELETE" will be printed if the user does not have the capability of deleting an entry from the specified data set.

## Lock and Unlock (Data Base)

```

mode:= 2;
dblock( base_name, argument {dummy}, mode, db_status );
IF db_status[1] = 20 THEN
BEGIN
    writeln( 'DATA BASE IS BUSY, TRY AGAIN LATER' );
    GOTO 100;
END;
IF db_status[1] <> 0 THEN
BEGIN
    writeln( 'DBLOCK FAILURE' );
    GOTO 100;
END;

mode:= 1;
dbunlock( base_name, argument {dummy}, mode, db_status );
IF db_status[1] <> 0 THEN
BEGIN
    writeln( 'DBUNLOCK FAILURE' );
    GOTO 100;
END;

```



## PASCAL

This program calls DBLOCK to lock the data base. The program must check the condition word when DBLOCK returns control to verify that the data base is locked, since mode 2 is used. If the data base is locked, condition word 0 is returned. If the data base is busy, condition word 20 is returned.

After the data base operations have been completed, the program unlocks the data base by calling the DBUNLOCK procedure.

### Request Data Item Information

```
mode:= 102;
item_list:= 'PURCH-DATE';

dbinfo( base_name, item_list, mode, db_status, info_buffer );
IF db_status[1] <> 0 THEN
BEGIN
    writeln( 'DBINFO FAILURE' );
    GOTO 100;
END;
```

The procedure call in this example obtains information about the PURCH-DATE (item\_list) data item by specifying mode 102. The item name and type are returned in info\_buffer.

### Rewind Data Set

```
mode:= 3;
dataset_name:= 'CUSTOMER';

dbclose ( base_name, dataset_name, mode, db_status );
IF db_status[1] <> 0 THEN
BEGIN
    writeln('DBCLOSE FAILURE');
    GOTO 100;
END;
```

To rewind the CUSTOMER data set, a call to DBCLOSE is made with mode 3. The dynamic status information in the DBU for CUSTOMER is reset, including the current record number.

## **Print Error**

```
IF db_status[1] <> 0 THEN
    dbexplain( db_status );
```

The call to DBEXPLAIN prints a message on the \$STDLIST device which interprets the contents of db\_status. This routine may be used during debugging if a procedure call fails.

## **Move Error to Buffer**

```
IF db_status[1] <> 0 THEN
BEGIN
    dberror( db_status, err_buffer, err_length );
    writeln( err_buffer );
END;
```

In the above example, a call to DBERROR has returned one of the messages appropriate when the condition word is not equal to zero.

## **Close Data Base**

```
mode:= 1;
dbcclose( base_name, dataset_name {dummy}, mode, db_status );
IF db_status[1] <> 0 THEN
    dbexplain( db_status );

END.
```

The data base is closed using mode 1. The program ends with a period.

# SPL

Figures 6-3 and 6-5 illustrate the use of TurboIMAGE procedures with SPL programs. The program in Figure 6-3 is called SUPPLMOD. It opens the data base in access mode 1 and allows the user to update, add, and delete entries of the master data set containing information on suppliers. Sample output from SUPPLMOD is illustrated in Figure 6-4.

Figure 6-5 contains a program called SHOWSALE, which displays credit card purchase transactions from the detail data set containing these entries. SHOWSALE opens the data base in access mode 6 thereby avoiding the necessity of locking and unlocking the data base. Figure 6-6 shows the output from SHOWSALE. The following numbered sequence corresponds to the numbered call-outs in Figure 6-3 and Figure 6-5.

- 1** TurboIMAGE procedures must be named in an INTRINSIC statement or, alternatively, declared as EXTERNAL procedures.
- 2** OPEN DATA BASE. The ORDERS data base is opened with access mode 1 and BUYER password. If the condition code is not zero, an error message is printed and the program terminates. (See (13) for another example of opening a data base.)
- 3** MOVE ERROR TO BUFFER. A message explaining the condition word returned by DBOPEN is moved to OUTBUF and I is set equal to the number of characters or length of the message.
- 4** REQUEST DATA SET INFORMATION. A call to DBINFO with mode 201 and data set name SUP-MASTER returns the data set number in DSET. For efficiency, this is done only once at the beginning of the program. (See (14) and (16) for other DBINFO examples.)
- 5** LOCK DATA SET. If the condition code is not CCE, the status information is printed indicating that the lock could not be obtained.
- 6** READ ENTRY (CALCULATED). This call locates an entry in the SUP-MASTER data set based on the search item value in SUPBUF. The entry need not be read since it is to be updated or deleted, therefore, the list is null and no data is actually transferred. SUPBUF is also used as the buffer parameter since no data is moved into it. If the condition word is 17, there is no search item with the specified value. Since the BUYER password allows access to the data, it is not necessary to check for condition word -21.
- 7** DELETE ENTRY. The entry located with DBGET is deleted. If the condition word is 44, the detail data sets linked to SUP-MASTER contain entries with the specified search item value, therefore, the master entry cannot be deleted. Since the BUYER password and access mode 1 allow the user to delete SUP-MASTER entries, it is not necessary to check for condition word -23.

```
BEGIN
```

```
<<*****>>
<<THIS PROGRAM OPENS THE "ORDERS" DATA BASE IN MODE 1 AND ALLOWS>>
<<THE USER INTERACTIVELY TO ADD, DELETE, OR UPDATE (CHANGE    >>
<<ADDRESSES OF) SUPPLIERS IN THE SUP-MASTER DATA SET. THE    >>
<<USER IS PROMPTED FOR THE DESIRED FUNCTION AND THEN FOR THE    >>
<<NECESSARY FIELD VALUES.                                     >>
<<THE PROGRAM CAN BE RUN FROM MULTIPLE TERMINALS (SESSIONS)    >>
<<SIMULTANEOUSLY, AND CAN ACCESS THE DATA BASE CONCURRENTLY WITH>>
<<OTHER PROGRAMS WHICH HAVE MODE 1 OR MODE 5 ACCESS TO IT.    >>
<<*****>>
```

```
INTEGER    MODE1      := 1,      <<MODES FOR >>
           MODE4      := 4,      <<USE IN    >>
           MODE7      := 7,      <<IMAGE CALLS>>
           MODE201    := 201,
           DSET,      <<NUMBER OF SUP-MASTER DATA SET>>
           I;
```

```
LOGICAL NULL'LIST    := ";";    <<SPECIAL "NO DATA" LIST>>
LOGICAL FULLREC      := "@;";    <<SPECIAL "COMPLETE ENTRY" LIST>>
```

```
ARRAY SBASE(0:4)      := " ORDERS;";    <<DATA BASE          >>
ARRAY PASSWORD(0:2)    := "BUYER;";      <<QUALIFIER - PASSWORD >>
ARRAY DSETNAME(0:5)    := "SUP-MASTER;";<<QUALIFIER-DATA SET NAME>
ARRAY STATUS(0:9);      <<STATUS AREA          >>
ARRAY SUPBUF(0:30);      <<BUFFER              >>
ARRAY INBUF(0:4);        <<INPUT BUFFER;FOR USER TO>>
BYTE ARRAY FUNCTION(*)=INBUF;    <<INPUT DESIRED FUNCTION>>
ARRAY OUTBUF(0:39);      <<OUTPUT BUFFER; FOR    >>
BYTE ARRAY BOUTBUF(*)=OUTBUF;    <<MESSAGES TO USER    >>
ARRAY FPROMPT(0:4) := "FUNCTION? ";
ARRAY PROMPT(0:18) := "SUPPLIER? STREET? CITY? STATE? ZIP? ";
ARRAY NOSUCH(0:7) := "NO SUCH SUPPLIER";
ARRAY CHAINS(0:21) := "CAN'T DELETE:PRODUCT(S) STILL IN INVENTORY";
ARRAY SETFULL(0:17) := "CAN'T ADD: SUPPLIER DATA SET IS FULL";
ARRAY DUPE(0:16) := "CAN'T ADD: DUPLICATE SUPPLIER NAME";
```

**Figure 6-3. Supplier Modification Program**

```

INTRINSIC DBOPEN,DBINFO,DBLOCK,DBGET,DBUPDATE,DBPUT,      1
          DBDELETE,DBUNLOCK,DBCLOSE,DBEXPLAIN,DBERROR;
INTRINSIC READ,PRINT,QUIT,TERMINATE;
  <<BEGINNING OF MAIN PROGRAM>>

      DBOPEN(SBASE,PASSWORD,MODE1,STATUS);      2
      <<OPEN ORDERS DATA BASE IN MODE 1.>>
      IF STATUS <> 0 THEN
        BEGIN
3      DBERROR(STATUS,OUTBUF,I);      <<GET ERROR MESSAGE>>
        IF STATUS <> 0 THEN GO TO DBFAIL; <<EVEN DBERROR FAILED>>
        PRINT(OUTBUF,-I,0);
        TERMINATE;
        END;
4      DBINFO(SBASE,DSETNAME,MODE201,STATUS,DSET);<<GET NUMBER>>
        IF STATUS <> 0 THEN GO TO DBFAIL;      <<OF SUP-MASTER>>
        DSET :=\DSET\;      <<MAKE SURE DSET# IS POSITIVE>>

ASK:   PRINT(FPROMPT,0,0);      <<SKIP A LINE>>
        PRINT(FPROMPT,5,%320);      <<ASK FOR FUNCTION>>
        I := READ (INBUF,-10);      <<READ DESIRED FUNCTION>>
        IF > THEN GO TO OUT;      <<EOF - MIGHT AS WELL LEAVE>>
        IF I = 0 THEN GO TO ASK;      <<NO INPUT OR I/O ERROR>>
        IF FUNCTION = "/E" THEN GO TO OUT; <<SPECIAL "END" SIGNAL>>
        IF FUNCTION <> "A" AND FUNCTION <> "D"
          AND FUNCTION <> "C" THEN GO TO ASK;
          <<FUNCTION MUST BE "ADD" OR "DELETE" OR "CHANGE">>
        SUPBUF := " ";      <<BLANK SUP-MASTER >>
        MOVE SUPBUF(1) := SUPBUF,(30);      <<BUFFER>>

        PRINT(PROMPT,5,%320);      <<REQUEST AND READ>>
        READ(SUPBUF,-16);      <<SUPPLIER NAME >>
        IF FUNCTION = "D" THEN GO TO LOCKIT; <<DELETE:GO DO IT>>
        PRINT(PROMPT(5),4,%320);      <<REQUEST AND READ>>
        READ(SUPBUF(8),-26);      <<STREET ADDRESS >>
        PRINT(PROMPT(9),3,%320);
        READ(SUPBUF(21),-12);      <<CITY, >>
        PRINT(PROMPT(12),-7,%320);
        READ(SUPBUF(27),-2);      <<STATE, >>
        PRINT(PROMPT(16),-5,%320);
        READ(SUPBUF(28),-5);      <<AND ZIP CODE >>

```

Figure 6-3. Supplier Modification Program (Continued)

```

LOCKIT:DBLOCK(SBASE,DSET,MODE4,STATUS);<<ADD:GO TO DBPUT>> 5
  IF STATUS <> 0 THEN GO TO DBFAIL;
  IF FUNCTION = "A" THEN GO TO NEWSUP; <<ADD: GO TO DBPUT>>

6  DBGET(SBASE,DSET,MODE7,STATUS,NULL'LIST,SUPBUF,SUPBUF);
    <<PRIOR TO UPDATING OR DELETING, MUST GET>>
    <<ASSOCIATIVE READ; SEARCH ITEM VALUE IN>>
    <<SUPBUF; TRANSFER NO DATA>>
  IF STATUS <> 0 THEN
    IF STATUS = 17 THEN
      BEGIN
        PRINT(NOSUCH,8,0); <<NO SUCH SUPPLIER IN SUP-MASTER>>
        GO TO UNLOCKIT;
      END
    ELSE GO TO DBFAIL;

    IF FUNCTION = "D"
      THEN DBDELETE(SBASE,DSET,MODE1,STATUS) 7

8  ELSE DBUPDATE(SBASE,DSET,MODE1,STATUS,FULLREC,SUPBUF);
    <<DELETE OR CHANGE (UPDATE),DEPENDING ON REQUEST>>
  IF STATUS <> 0 THEN
    IF STATUS = 44 THEN PRINT(CHAINS,-43,0) <<CAN'T DELETE>>
    ELSE GO TO DBFAIL;
  GO TO UNLOCKIT;

NEWSUP:DBPUT(SBASE,DSET,MODE1,STATUS,FULLREC,SUPBUF); 9
  IF STATUS <> 0 THEN
    IF STATUS = 16 THEN PRINT(SETFULL,18,0) <<NO ROOM >>
    ELSE IF STATUS = 43 THEN PRINT(DUPE,17,0) <<DUPLICATE>>
    ELSE GO TO DBFAIL;

UNLOCKIT: DBUNLOCK(SBASE,DSET,MODE1,STATUS); 10
  IF STATUS = 0 THEN GO TO ASK;

DBFAIL:<<COME HERE ON UNEXPECTED OR IRRECOVERABLE ERROR>>
  <<RETURNED BY ANY IMAGE PROCEDURE. THIS IS >>
  <<APPARENTLY A PROGRAM BUG, SO PRINT ALL AVAIL- >>
  <<ABLE INFORMATION ON THE ERROR BEFORE QUITTING.>>
  DBEXPLAIN(STATUS); 11
  QUIT(1); <<IRRECOVERABLE: GET OUT.>>

OUT: DBCLOSE(SBASE,DSET,MODE1,STATUS); 12
  IF STATUS <> 0 THEN GO TO DBFAIL;
  END.

```

Figure 6-3. Supplier Modification Program (Continued)

## **SPL**

- 8**    **UPDATE ENTRY.** The entry located with **DBGET** is updated with the data in **SUPBUF**. The user is prompted for this data prior to the call to **DBGET**. **FULLREC** contains @; which indicates the entire entry is to be updated. In this case, the search item value must equal the value that is already in the entry. Since the **BUYER** password and access mode 1 allow updates to this data set, it is not necessary to check for condition word 42.
- 9**    **ADD ENTRY.** This call adds an entry to the **SUP-MASTER** data set using the data in **SUPBUF**. The list parameter in **FULLREC** is @; specifying that values are provided for all data items in the entry. If the condition word is 16, the data set is full and, if it is 43, there is already an entry with the specified search item value. Since the **BUYER** password and access mode 1 allow adding entries to the data set, it is not necessary to check for condition word -23.
- 10**   **UNLOCK DATA SET.** This call unlocks the data set. The **DSET** parameter is ignored. If the call is successful, the condition code is **CCE** and the program branches to **ASK**.
- 11**   **PRINT ERROR.** A call to **DBEXPLAIN** prints a message interpreting the **STATUS** array contents.
- 12**   **CLOSE DATA BASE.** This call closes the **ORDERS** data base. The **DSET** parameter is ignored in mode 1.

```
:RUN SUPPLMOD

FUNCTION? ADD
SUPPLIER? ACME
STREET? 2587 BIRD ST.
CITY? INDIANOLA
STATE? IA
ZIP? 50125

FUNCTION? ADD
SUPPLIER? ACME
STREET? 140 CORYDON AVE.
CITY? BROOKLYN
STATE? NY
ZIP? 11208
CAN'T ADD: DUPLICATE SUPPLIER NAME

FUNCTION? CHA
SUPPLIER? ACME
STREET? 140 CORYDON AVE.
CITY? BROOKLYN
STATE? NY
ZIP? 11208

FUNCTION? DELETE
SUPPLIER? ACME

FUNCTION? DEL
SUPPLIER? ACME
NO SUCH SUPPLIER

FUNCTION? /E

END OF PROGRAM
```

Figure 6-4. Sample SUPPLMOD Execution



BEGIN

```
<<*****>>
<<THIS PROGRAM OPENS THE "ORDERS" DATA BASE IN MODE 6      >>
<<AND ALLOWS THE USER INTERACTIVELY TO REQUEST DISPLAYS OF SALES>>
<<TRANSACTIONS FROM THE SALES DATA SET.  FOR EACH TRANSACTION, >>
<<THE ITEMS FROM WITHIN THE CORRESPONDING ENTRY WHICH ARE    >>
<<DISPLAYED ARE ACCOUNT, QUANTITY, STOCK#, TOTAL (TOTAL PRICE >>
<<IN PENNIES), PURCH-DATE, AND DELIV-DATE.  ALSO, THE DESCRIP- >>
<<TION OF THE PRODUCT IS OBTAINED FROM THE PRODUCT DATA SET AND >>
<<PRINTED NEXT TO THE STOCK#.  AFTER THE SALES LINES, A GRAND >>
<<TOTAL PRICE LINE IS PRINTED.                                >>
<<THERE ARE FIVE WAYS OF SELECTING SALES ENTRIES TO BE PRINTED. >>
<<THEY ARE:  1) ALL SALES TRANSACTIONS IN THE DATA SET      >>
<<              2) ALL SALES TO A PARTICULAR ACCOUNT (CUSTOMER) >>
<<              3) ALL SALES OF A PARTICULAR STOCK# (PRODUCT) >>
<<              4) ALL SALES WITH A PARTICULAR PURCHASE DATE  >>
<<              5) ALL SALES WITH A PARTICULAR DELIVERY DATE  >>
<<FOR (1) ABOVE, THE DATA SET IS READ SERIALY, WITH EACH ENTRY >>
<<ENCOUNTERED BEING PRINTED.  THE OTHER SELECTION METHODS    >>
<<REQUIRE A SPECIFIC VALUE FOR A CERTAIN ITEM WITHIN THE ENTRY. >>
<<SINCE ALL OF THE ITEMS IN QUESTION ARE SEARCH ITEMS WITHIN THE>>
<<SALES DATA SET, THE PROGRAM MERELY CALLS DBFIND FOR THE     >>
<<PARTICULAR ITEM AND VALUE, AND THEN DOES CHAINED DBGETS TO  >>
<<RETRIEVE THE DESIRED ENTRIES.  BEFORE DOING SO, OF COURSE, THE>>
<<PROGRAM PROMPTS THE USER FOR THE ITEM NAME AND ITS VALUE.   >>
<<THE PROGRAM CAN BE RUN FROM MULTIPLE TERMINALS (SESSIONS)   >>
<<SIMULTANEOUSLY, AND CAN ACCESS THE DATA BASE CONCURRENTLY WITH>>
<<OTHER PROGRAMS WHICH OPEN IT IN MODE 2, 4, 6, OR 8.         >>
<<*****>>
```

```
INTEGER    MODE,
            MODE1    :=1,      <<MODES              >>
            MODE2    :=2,      <<      FOR              >>
            MODE3    :=3,      <<      USE              >>
            MODE4    :=4,      <<      IN              >>
            MODE5    :=5,      <<      TurboIMAGE      >>
            MODE7    :=7,      <<      CALLS           >>
            MODE201  :=201,
            SALES,      <<DATA SET NUMBER - SALES>>
            PRODUCT,   <<DATA SET NUMBER - PRODUCT>>
            ARGLGTH,
            I;          <<HANDY-DANDY VARIABLE>>

LOGICAL    SAMELIST := "*,*"; <<SPECIAL "SAME AS LAST TIME" LIST>>
```

Figure 6-5. Purchase Transaction Display Program

```

DOUBLE GRANDTOTAL;

ARRAY SBASE(0:4)      := "  ORDERS;";      <<DATA BASE>>
ARRAY PASSWORD(0:2) := "CLERK;";          <<QUALIFIER - PASSWORD>>
ARRAY SALENAME(0:2) := "SALES;";          <<QUALIFIER-DATA SET NAME>>
ARRAY PRODNAME(0:3) := "PRODUCT;";        <<QUALIFIER-DATA SET NAME>>
ARRAY STATUS(0:9);      <<STATUS AREA>>
ARRAY SALESLIST(0:25) := "ACCOUNT,QUANTITY,STOCK#,TOTAL,";
                        "PURCH-DATE,DELIV-DATE;";
ARRAY PRODLIST(0:5) := "DESCRIPTION;";
ARRAY ITEM(0:8);      <<ITEM NAME FOR DBFIND>>
ARRAY SALESBUF(0:14);  <<BUFFER - SALES DATA SET>>
DOUBLE ARRAY ACCOUNT(*)=SALESBUF;
DOUBLE ARRAY TOTALCOST(*)=SALESBUF(7);
BYTE ARRAY BSALESBUF(*)=SALESBUF;
ARRAY ARG(0:4);      <<ARGUMENT FOR DBFIND>>
DOUBLE ARRAY DARG(*)=ARG;
BYTE ARRAY BARG(*)=ARG;
ARRAY INBUF(0:7);      <<INPUT BUFFER;FOR USER TO>>
BYTE ARRAY SELECT(*)=INBUF;    <<ENTER SALES SELECT TYPE>>
ARRAY OUTBUF(0:39);      <<OUTPUT BUFFER; FOR>>
BYTE ARRAY BOUTBUF(*)=OUTBUF;  <<MESSAGES TO USER>>
BYTE ARRAY WORKBUF(0:10);
ARRAY SPROMPT(0:7) := "ALL SALES FOR? ";
ARRAY WPROMPT(0:5) := "WHICH ONE? ";

INTRINSIC DBOPEN,DBINFO,DBCLOSE,DBFIND,DBGET,DBEXPLAIN,DBERROR,
          READ,PRINT,ASCII,QUIT,DASCII,DBINARY,TERMINATE;

<<BEGINNING OF MAIN PROGRAM>>

MODE := 6;
DBOPEN(SBASE,PASSWORD,MODE,STATUS);      13
  <<OPEN ORDERS DATA BASE IN MODE 6>>
IF STATUS <> 0 THEN
  BEGIN
    DBERROR(STATUS,OUTBUF,I);      <<GET OPEN ERROR MESS.>>
    IF STATUS <> 0 THEN GO TO DBFAIL;    <<DBERROR FAILED>>
    PRINT(OUTBUF,-I,0);
    TERMINATE;
  END;

```

Figure 6-5. Purchase Transaction Display Program (Continued)

```

DBINFO(SBASE,SALENAME,MODE201,STATUS,SALES);          14
  <<FOR EFFICIENCY, GET NUMBER OF SALES DATA SET>>
  IF STATUS <> 0 THEN GO TO DBFAIL;
  SALES :=\SALES\;          <<MAKE SURE DSET# IS POSITIVE>>
  DARG :=0D;
15  DBGET(SBASE,SALES,MODE4,STATUS,SALESLIST,OUTBUF,DARG);
    <<SET UP LIST FOR FUTURE DBGET CALLS ON SALES DATA>>
    <<SET. THIS DIRECTED READ OF ENTRY #0 SHOULD FAIL,>>
    <<BUT ONLY AFTER INTERNALLY RECORDING SPECIFIED >>
    <<LIST. NO DATA WILL BE TRANSFERRED. >>
    IF STATUS <> 12 <<DIRECTED EOF>> THEN GO TO DBFAIL;
  DBINFO(SBASE,PRODNAME,MODE201,STATUS,PRODUCT);      16
  IF STATUS <> 0 THEN GO TO DBFAIL;
  PRODUCT :=\PRODUCT\;
  DBGET(SBASE,PRODUCT,MODE4,STATUS,PRODLIST,OUTBUF,DARG); 17
  IF STATUS <> 12 THEN GO TO DBFAIL;
    <<ALSO SET UP FOR DBGETS FROM PRODUCT DATA SET>>

NEXT:  GRANDTOTAL :=0D;
  PRINT(SPROMPT,-15,%320);          <<ASK FOR SALES SELECT TYPE>>
  I := READ(INBUF,-15);             <<READ IT>>
  IF > THEN GO TO OUT;               <<EOF - MIGHT AS WELL STOP>>
  IF I = 0 THEN GO TO NEXT;          <<NO INPUT OR I/O ERROR>>
  IF SELECT = "/E" THEN GO TO OUT;  <<SPECIAL STOP INPUT>>
  IF SELECT = "/C" THEN GO TO ALL;  <<SPECIAL ALL SALES RQST>>

  IF SELECT = "A" THEN
    BEGIN
      MOVE ITEM := "ACCOUNT;";
      ARGLGTH := -10;
    END
  ELSE IF SELECT = "S" THEN
    BEGIN
      MOVE ITEM := "STOCK#;";
      ARGLGTH := -8;
    END
  ELSE IF SELECT = "P" THEN
    BEGIN
      MOVE ITEM := "PURCH-DATE;";
      ARGLGTH := -6;
    END

```

Figure 6-5. Purchase Transaction Display Program (Continued)

```

ELSE IF SELECT = "D" THEN
  BEGIN
    MOVE ITEM := "DELIV-DATE;";
    ARGLGTH := -6;
    END
ELSE GO TO NEXT;          <<UNRECOGNIZED SELECT TYPE>>

<<AT THIS POINT, THE SELECT TYPE (SEARCH ITEM) HAS BEEN >>
<<SPECIFIED. THAT IS, THE USER HAS REQUESTED TO SEE ALL >>
<<SALES TRANSACTIONS FOR AN ACCOUNT OR A STOCK NUMBER OR>>
<<A PURCHASE DATE OR A DELIVERY DATE. NOW, ASK FOR THE >>
<<VALUE OF THE SEARCH ITEM. >>

SIVALUE: ARG := " ";
MOVE ARG(1) := ARG,(4);
PRINT(WPROMPT,-11,%320);    <<REQUEST AND READ>>
I := READ(ARG,ARGLGTH);    <<SEARCH ITEM VALUE>>
IF > THEN GO TO OUT;      <<EOF - MIGHT AS WELL STOP>>
IF < THEN GO TO SIVALUE;  <<I/O ERROR - ASK AGAIN>>
IF SELECT = "A" THEN
  BEGIN                    <<ACCOUNT NUMBER:TRANSLATE>>
    DARG := DBINARY(BARG,I); <<TO INTERNAL BINARY FORM>>
    IF <> THEN GO TO SIVALUE;
    END;

    <<SEARCH ITEM NAME IS NOW IN ITEM AND SEARCH>>
    <<ITEM VALUE IS IN ARG. >>
    DBFIND(SBASE,SALES,MODE1,STATUS,ITEM,ARG);
    <<GET TO HEAD OF CHAIN OF INTEREST>>
18 IF STATUS <> 0 THEN
  IF STATUS = 17 THEN GO TO WRAPUP <<NO CHAIN FOR THIS VALUE>>
  ELSE GO TO DBFAIL;
  MODE := MODE5;          <<PREPARE FOR CHAINED DBGETS>>
  GO TO GETNEXT;          <<GO RETRIEVE AND REPORT>>

ALL: <<COME HERE TO REPORT ALL SALES TRANSACTIONS, RATHER>>
<<THAN A SELECTED SUBSET. >>
MODE := MODE2;            <<PREPARE FOR SERIAL DBGETS>>
DBCLOSE(SBASE,SALES,MODE3,STATUS);    19
  <<REWIND SALES DATA SET>>
  IF STATUS <> 0 THEN GO TO DBFAIL;

```

Figure 6-5. Purchase Transaction Display Program (Continued)

```

GETNEXT:  DBGET(SBASE,SALES,MODE,STATUS,SAMELIST,SALESBUF,ARG);
20      <<GET NEXT SALES TRANSACTION. THIS IS EITHER A >>
      <<SERIAL (MODE2) OR CHAINED (MODE5) DBGET. IN >>
      <<EITHER CASE, ARG IS IGNORED. >>
      IF STATUS <> 0 THEN
        IF STATUS = 11 OR STATUS = 15 THEN GO TO WRAPUP <<NO MORE>>
        ELSE GO TO DBFAIL;

      <<WE HAVE A SALES TRANSACTION; FORMAT IT FOR PRINTING>>
      OUTBUF := " "; <<BLANK OUTPUT >>
      MOVE OUTBUF(1) := OUTBUF,(35); <<BUFFER >>
      DASCII(ACCOUNT,10,BOUTBUF); <<ACCOUNT NUMBER>>
      ASCII(SALESBUF(2),-10,BOUTBUF(13)); <<QUANTITY >>
      MOVE OUTBUF(8) := SALESBUF(3),(4); <<STOCK# >>
21      BGET(SBASE,PRODUCT,MODE7,STATUS,SAMELIST,OUTBUF(13),
      SALESBUF(3)); <<GET DESCRIPTION FROM PRODUCT>>
      IF STATUS <> 0 THEN GO TO DBFAIL; <<DATA SET >>
      I := DASCII(TOTALCOST,10,WORKBUF); <<TOTAL COST >>
      MOVE BOUTBUF(55-I) := WORKBUF,(I); <<RIGHT JUSTIFY >>
      MOVE BOUTBUF(57) := BSALESBUF(18),(6); <<PURCHASE DATE>>
      MOVE BOUTBUF(65) := BSALESBUF(24),(6); <<DELIVERY DATE>>
      PRINT(OUTBUF,-71,0); <<OUTPUT SALES TRANSACTION>>
      GRANDTOTAL := GRANDTOTAL + TOTALCOST; <<ACCUMULATE TOTAL>>
      GO TO GETNEXT; <<GO GET NEXT SALES>>

WRAPUP:  OUTBUF := " ";
      MOVE OUTBUF(1) := OUTBUF,(15);
      MOVE OUTBUF(16) := " GRAND TOTAL: ";
      I := DASCII(GRANDTOTAL,10,WORKBUF); <<GRAND TOTAL >>
      MOVE BOUTBUF(55-I) := WORKBUF,(I); <<RIGHT JUSTIFY>>
      PRINT(OUTBUF,-55,%202); <<OUTPUT GRAND TOTAL & SKIP LINE>>
      GO TO NEXT; <<GO ASK FOR NEXT REQUEST>>

DBFAIL:  <<COME HERE ON UNEXPECTED OR IRRECOVERABLE ERROR>>
      <<RETURNED BY ANY TurboIMAGE PROCEDURE. THERE IS >>
      <<NOTHING TO DO BUT TERMINATE, SO PRINT ALL >>
      <<INFORMATION ABOUT THE ERROR ON $STDLIST. >>
      DBEXPLAIN(STATUS);
      QUIT(1); <<IRRECOVERABLE: GET OUT>>
OUT:     DBCLOSE(SBASE,SALES,MODE1,STATUS);
      IF STATUS <> 0 THEN GO TO DBFAIL;
      END.

```

Figure 6-5. Purchase Transaction Display Program (Continued)

- 13 OPEN DATA BASE. The ORDERS data base is opened in mode 6 with password CLERK.
- 14 REQUEST DATA SET INFORMATION. The data set number for SALES is requested. Note that SALENAME is an array containing "SALES;" and the data set number is stored in the SALES variable.
- 15 READ ENTRY (DIRECTLY). This call requests a directed read of the entry in record 0. Although this read fails and returns condition word 12, the internal list of items is set up to include ACCOUNT, QUANTITY, STOCK#, and TOTAL. No data is transferred. Subsequent calls to read an entry from the SALES data set can use the special list construct \*; indicating the list is the same as the one used in this call. This technique saves processing time since the list is set up only once during program execution.
- 16 REQUEST DATA SET INFORMATION. The data set number for PRODUCT is requested.
- 17 READ ENTRY (DIRECTLY). This call is the same as (15) except the data set name is PRODUCT and the list includes only the DESCRIPTION item.
- 18 READ ENTRY (CHAINED). A call to DBFIND locates the pointers for a chain in the SALES data set. In the preceding code, the user is prompted for the search item (ACCOUNT, STOCK#, PURCH-DATE, or DELIV-DATE) and its value. ITEM contains the search item name and ARG contains the search item value. If the condition word is 17, there is no chain with the requested value. The read is performed with the call described below in (20).
- 19 REWIND DATA SET. A call to DBCLOSE with mode 3 rewinds the SALES data set to prepare for serial reads of all entries in the set. If the rewind fails, the condition code is CCL or CCG.
- 20 READ ENTRY (SERIAL OR CHAINED). This call is coded so that it performs either a forward chained (mode 5) or forward serial (mode 2) read of the SALES data set. The data is read into SALESBUF and the list is \*; indicating it is the same list that was set up in calls (15) and (17). ARG is ignored in both modes 2 and 5. If the end of the data set is reached while doing a serial read, the condition word is 11. If the end of chain is reached while doing a chained read, the condition word is 15. Since access mode 6 and password CLERK allow the user to read all items in the SALES and PRODUCT data sets, it is not necessary to check for condition word -21.
- 21 READ ENTRY (CALCULATED). A calculated read (mode 7) is performed using the search item value for STOCK# that is in SALESBUF(3) and (4). The data set is PRODUCT and the list parameter is \*. The description is read into OUTBUF(13) through OUTBUF(22).

:RUN SHOWSALE

ALL SALES FOR? /C

24536173	4	5405T14F	BAR STOOL	10300	840318	840320
24536173	1	3586T14Y	BIRDHOUSE	630	840319	CARRY
24536173	2	4397D13P	DRAIN OPENER	189	840320	CARRY
24536173	1	7391Z22F	PORTABLE WB KIT	24273	840321	840322
54283545	27	6650D22S	BASEBALL BAT	12567	840321	840322
10293847	1	3739A14F	CONVERTIBLE KIT	41722	840319	840320
54283545	1	4397D13P	DRAIN COVER	90	840322	CARRY
82463761	1	3586T14Y	BIRDHOUSE	630	840319	CARRY
82463761	1	2457A11C	NEHRU JACKET	271	840319	840322
10293847	1	4397D13P	DRAIN OPENER	90	840322	CARRY
90542176	1	6650D22S	BASEBALL BAT	517	840320	CARRY
44556677	2	5404T14F	BAR STOOL	5150	840319	840320
GRAND TOTAL:				96375		

ALL SALES FOR? ACCOUNT

WHICH ONE? 10293847

10293847	1	3739A14F	CONVERTIBLE KIT	41722	840319	840320
10293847	1	4397D13P	DRAIN OPENER	90	840322	CARRY
GRAND TOTAL:				41812		

ALL SALES FOR? STOCK#

WHICH ONE? 4397D13P

24536173	2	4397D13P	DRAIN OPENER	139	840321	CARRY
54283545	1	4397D13P	DRAIN OPENER	90	840322	CARRY
10293847	1	4397D13P	DRAIN OPENER	90	840322	CARRY
GRAND TOTAL:				369		

ALL SALES FOR? PUR

WHICH ONE? 740320

90542176	1	6650D22S	BASEBALL BAT	517	840320	CARRY
GRAND TOTAL:				517		

ALL SALES FOR? D

WHICH ONE? 740320

10293847	1	3739A14F	CONVERTIBLE KIT	41722	840319	840320
24536173	4	5404T14F	BAR STOOL	10300	840318	840320
44556677	2	5405T14F	BAR STOOL	5150	840319	840320
GRAND TOTAL:				57172		

ALL SALES FOR? ST

WHICH ONE? 9999F99F

GRAND TOTAL: 0

ALL SALES FOR? /E

END OF PROGRAM

Figure 6-6. Sample SHOWSALE Execution

To simplify your access to a TurboIMAGE data base through BASIC language programs, it is recommended that you use the BIMAGE interface procedures provided with the TurboIMAGE software. These routines convert all parameter byte addresses to word addresses as required by TurboIMAGE. In addition to calling the necessary TurboIMAGE procedure, the BIMAGE procedures perform the following functions for your convenience:

- Automatically pack, into a buffer, a list of expressions before calling the DBPUT or DBUPDATE procedures.
- Automatically unpack, from a buffer to a list of BASIC variables, the values of items returned by DBGET or the values returned by DBINFO.
- Automatically update the logical length of string variables to which data is transferred from the data base to reflect the length of the string actually transferred.

Table 6-2 lists the BIMAGE interface procedures with the TurboIMAGE procedures to which they correspond. The parameters are described in Table 6-3. The corresponding TurboIMAGE procedure parameter is listed next to the BIMAGE parameter.



# BASIC

Table 6-2. BIMAGE Procedure Calls

BIMAGE	CORRESPONDS TO:
XDBOPEN ( <i>B</i> \$, <i>W</i> \$, <i>mode</i> , <i>status</i> (*))	DBOPEN
XDBPUT ( <i>B</i> \$,{ <i>D</i> \$ <sub><i>d</i></sub> }, <i>mode</i> , <i>status</i> (*),{ <i>L</i> \$ <sub><i>l</i></sub> }, <i>writel</i> <i>ist</i> )	DBPUT
XDBFIND ( <i>B</i> \$,{ <i>D</i> \$ <sub><i>d</i></sub> }, <i>mode</i> , <i>status</i> (*),{ <i>I</i> \$ <sub><i>i</i></sub> },{ <i>A</i> \$ <sub><i>a</i></sub> })	DBFIND
XDBGET ( <i>B</i> \$,{ <i>D</i> \$ <sub><i>d</i></sub> }, <i>mode</i> , <i>status</i> (*),{ <i>L</i> \$ <sub><i>l</i></sub> }, <i>readl</i> <i>ist</i> ,{ <i>A</i> \$ <sub><i>a</i></sub> })	DBGET
XDBUPDATE ( <i>B</i> \$,{ <i>D</i> \$ <sub><i>d</i></sub> }, <i>mode</i> , <i>status</i> (*),{ <i>L</i> \$ <sub><i>l</i></sub> }, <i>writel</i> <i>ist</i> )	DBUPDATE
XDBDELETE ( <i>B</i> \$,{ <i>D</i> \$ <sub><i>d</i></sub> }, <i>mode</i> , <i>status</i> (*))	DBDELETE
XDBLOCK ( <i>B</i> \$,{ <i>descripl</i> <i>ist</i> }, <i>mode</i> , <i>status</i> (*))	DBLOCK
XDBUNLOCK ( <i>B</i> \$,{ <i>D</i> \$ <sub><i>d</i></sub> }, <i>mode</i> , <i>status</i> (*))	DBUNLOCK
XDBCLOSE ( <i>B</i> \$,{ <i>D</i> \$ <sub><i>d</i></sub> }, <i>mode</i> , <i>status</i> (*))	DBCLOSE
XDBBEGIN ( <i>B</i> \$, <i>T</i> \$, <i>mode</i> , <i>status</i> (*))	DBBEGIN
XDBMEMO ( <i>B</i> \$, <i>T</i> \$, <i>mode</i> , <i>status</i> (*))	DBMEMO
XDBEND ( <i>B</i> \$, <i>T</i> \$, <i>mode</i> , <i>status</i> (*))	DBEND
XDBINFO ( <i>B</i> \$,{ <i>Q</i> \$ <sub><i>q</i></sub> }, <i>mode</i> , <i>status</i> (*), <i>readl</i> <i>ist</i> )	DBINFO
XDBEXPLAIN ( <i>status</i> (*))	DBEXPLAIN
XDBERROR ( <i>status</i> (*), <i>M</i> \$[, <i>length</i> ])	DBERROR

Table 6-2. BIMAGE Procedure Calls (Continued)

BIMAGE	CORRESPONDS TO:
XDBEND ( $B\$,T\$,mode,status(*)$ )	DBEND
XDBCNTROL ( $B\$, \left\{ \begin{smallmatrix} D\$ \\ d \end{smallmatrix} \right\}, mode, status(*)$ )	DBCNTROL

# BASIC

Table 6-3. BIMAGE Procedure Parameters

BIMAGE**	IMAGE	
<i>A\$</i>	<i>argument</i>	May be any string expression.
<i>a</i>	<i>argument</i>	May be a numeric expression or numeric array of any data-type.
<i>B\$</i>	<i>base</i>	Must be a simple string variable. Value should not be altered between calls to XDBOPEN and XDBCLOSE.
<i>D\$</i>	<i>dset</i>	May be any string expression.
<i>d</i>	<i>dset</i>	May be a type-INTEGER expression.*
<i>descriplist</i>	<i>qualifier</i>	Has same form as <i>writelists</i> . You should ensure that once BASIC has concatenated the component variables, the result is a valid lock descriptor list (or set name) as defined for DBLOCK. (Parameter ignored for DBLOCK modes 1 and 2).
<i>l\$</i>	<i>item</i>	May be any string expression.
<i>i</i>	<i>item</i>	May be a type-INTEGER expression.*
<i>L\$</i>	<i>list</i>	May be any string expression or a string array. If it is a string array, all of the string elements are concatenated to form one string whose length may not exceed 255 characters. The concatenated string must form a syntactically correct <i>list</i> parameter. Commas must be placed appropriately.
<i>l</i>	<i>list</i>	May be an array of type INTEGER.
<i>length</i>	<i>length</i>	Must be a simple or subscripted type-INTEGER variable (if not, parameter is ignored.) Parameter is optional but if present, total length of TurboIMAGE message is returned. Value may exceed length of message by BIMAGE procedure if M\$ is too small and message is truncated. Not needed when M\$ is a string variable.
<i>M\$</i>	<i>buffer</i>	Should be a simple or subscripted string variable without substring designators. If message is larger than M\$, message is truncated on the right. Logical length of M\$ is set to length of message returned by BIMAGE and may not be equal to <i>length</i> if message is truncated.
<i>mode</i>	<i>mode</i>	Must be type-INTEGER expression.*
<i>Q\$</i>	<i>qualifier</i>	May be any string expression.
<i>q</i>	<i>qualifier</i>	May be a type-INTEGER expression.

Table 6-3. BIMAGE Procedure Parameters (Continued)

BIMAGE**	IMAGE	
<i>readlist</i>	<i>buffer</i>	Has form similar to <i>item list</i> of BASIC READ or MAT READ statement. May consist of one or more string or numeric simple or subscripted variables or arrays separated by commas. String variables with substring designators and the "FOR-loop" construct are not permitted.
<i>status(*)</i>	<i>status</i>	Must be a type-INTEGER array containing at least ten active elements.
<i>T\$</i>	<i>text</i>	Must be a simple string variable.
<i>W\$</i>	<i>password</i>	May be any string expression.
<i>writelist</i>	<i>buffer</i>	Has form similar to <i>item list</i> of BASIC PRINT or MAT PRINT statement. May consist of one or more string or numeric expressions or arrays separated by commas. "FOR-loop" not permitted. Substring designators are permitted.

\* See discussion of type-INTEGER expressions as parameters.

\*\* Note that if you specify an array as a parameter you must obey BASIC syntax rules and append parenthesis and asterisks, for example, L\$(\*,\*) or A(\*) .

Refer to the TurboIMAGE procedure descriptions in Section 4 for details regarding the purpose of a procedure and its parameters as well as available options.

BIMAGE provides some extensions to the TurboIMAGE procedure calling sequences to simplify your access to the data base:

- BIMAGE allows you to enter a list of expressions in place of the buffer parameter. The list is automatically packed into or unpacked from a temporary buffer constructed by the BIMAGE procedures. This facility is also available to construct lock descriptor lists.
- String or numeric expressions are accepted for many parameters. For example, the *dset* parameter may be a string expression when specifying the data set by name or a numeric expression when specifying the data set by number.

# BASIC

## String Variables

The physical length of a string variable determines the number of characters (bytes) read by the XDBGET procedure and the logical length of a string variable determines the number of characters written by the XDBPUT and XDBUPDATE procedures. Thus, you should ensure that the physical length of a string variable specified in a DIM or COM statement exactly matches the size of the item to be read by a call to XDBGET.

On the other hand, the same string variable can be used to write items of varying sizes. Substring designators should be used to ensure that the actual string passed to XDBPUT or XDBUPDATE fills the item to be written. For example, if the item is 8 characters long, and substring S\$(3) is 2 characters long, S\$(3,10) or S\$(3;8) fills the item with the S\$(3) substring and appends 6 blanks.

If the string variable is an array, the length of each string element or of the concatenated string elements should correspond to the length of the item or sub-item to be written. You can ensure this by specifying substring designators when assigning values to elements of the string array in your BASIC program.

## Type-Integer Expressions as Parameters

Since BASIC treats integral numeric constants as type-REAL, expressions involving constants cannot be passed directly to a type-INTEGER parameter of a BIMAGE procedure. You can define a function such as the following to ensure that a type-INTEGER expression is passed:

```
10 DEF INTEGER FNI(X)=X
```

When a procedure call is made, the function is used in this way:

```
50 CALL XDBLOCK(B$,D$,FNI(expression),S(*))
```

The function FNI converts expression to type-INTEGER.

## Doubleword Integer Parameters

In order to specify a doubleword integer in BASIC, define a two-word array in which the first word contains the high-order digits of values greater than 32767, or zero. The second word must contain low-order digits of values greater than or equal to 32767, or the entire value if it is less than or equal to 32767.

## Readlist, Writelist, Descriplist Parameters

When specifying string expressions in a readlist, writelist, or descriplist, each string expression should correspond to a data item or sub-item, or groups of items or sub-items in the case of string arrays. You should not specify several string expressions as the source or destination of one item or sub-item. The transfer of strings to or from the data base always begins on a word boundary of the buffer. Therefore, writing from or reading into two odd-length strings is not the same as writing or reading into one even-length string.

## The Status Parameter

If the *status* parameter is a type-INTEGER variable, a condition word is returned in the first word and the second word is set to zero if *status* is at least a two-element array. The condition word will have a value equal to those listed for the corresponding TurboIMAGE procedure and, in addition, may contain one of the conditions listed in Table 6-4.

If the *status* parameter is not type-INTEGER, the BIMAGE procedures cannot return a condition word for the common error: failure to declare the *status* variable type-INTEGER. This error will usually result in the BASIC message UNDEFINED VALUE the first time the *status* array contents are examined.

**Table 6-4. Additional BIMAGE Condition Word Values**

EXCEPTIONAL CONDITIONS	PROCEDURES
51 Insufficient stack for temporary buffer.	XDBGET,XDBPUT, XDBINFO,XDUPDATE
52 Invalid number of parameters.	All procedures except XDBERROR and XDBEXPLAIN
53 Invalid parameter.	
54 <i>status array</i> has less than 10 elements.	

## BASIC

### Open Data Base

```
10 DIM B$(10),P$(8)
20 INTEGER S(10),M
30 B$="  ORDERS;"
40 INPUT "ENTER PASSWORD: ",P$(1;8)
50 INPUT "ENTER ACCESS MODE (1-8): ",M
60 CALL XDBOPEN(B$,P$,M,S[*])
70 IF S[1]<>0 THEN 9300
.
.
.
(code to use data base)
.
.
.
9300 PRINT "DBOPEN FAILURE"
9310 CALL XDBEXPLAIN(S[*])
9320 STOP
.
.
.
```

In this example, the ORDERS data base is opened in the access mode entered by the user and with a user class number corresponding to the password entered by the user and stored in the P\$ string. If the password is less than 8 characters the P\$ string is padded with blanks. The first word of the status array, S, is tested to determine whether the procedure executed successfully. If not, an error message is printed.

## Add Entry

```

10 DIM B$(10),P$(8),A$(8),C$(20)
20 INTEGER S(10),M,M1
30 B$=" ORDERS;"
40 INPUT "ENTER PASSWORD: ",P$(1;8)
50 INPUT "ENTER ACCESS MODE (3,4): ",M
60 GOTO M OF 70,70,90,90
70 PRINT "CANNOT ADD ENTRIES IN THIS ACCESS MODE"
80 GOTO 50
90 CALL XDBOPEN(B$,P$,M,S[*])
100 IF S[1]<>0 THEN 9300
110 INPUT "ENTER STOCK# OR / TO TERMINATE: ",A$(1;8)
120 IF A$[1,1]="/" THEN GOTO 9900
130 INPUT "ENTER DESCRIPTION: ",C$[1;20]
140 M1=1
150 CALL XDBPUT(B$,"PRODUCT;",M1,S[*],">@;",A$[1;8],C$[1;20])
160 IF S[1]<>43 THEN 190
170 PRINT "DUPLICATE STOCK NUMBER"
180 GOTO 110
190 IF S[1]<>16 THEN 220
200 PRINT "DATA SET FULL"
210 GOTO 9900
220 IF S[1]<>0 THEN 250
230 PRINT "NEW PRODUCT HAS BEEN ENTERED"
240 GOTO 110
250 IF S[1]=-23 THEN 290
260 PRINT "DBPUT FAILURE"
270 CALL XDBEXPLAIN(S[*])
280 GOTO 9900
290 PRINT "YOUR PASSWORD DOES NOT ALLOW YOU TO ADD ENTRIES"
300 GOTO 9900
.
.
.
9300 (code same as example above)
9900 (close data base)

```

This sample code adds an entry to the **PRODUCT** manual master data set. Note that the **B\$** string used to open the data base is the base parameter in this call. It should not be changed after the call to **XDBOPEN** since this call saves a data base identifier in the first word of **B\$**. This list of items to be added is specified as **@;** which indicates that values are specified for all items in the entry. The values for the **STOCK#** and **DESCRIPTION** data items are stored in **A\$** and **C\$**. Sample values are "7474Z74Z" and "ORANGE CRATEΔΔΔΔΔΔΔΔ".

In the example, the condition word of the status array is tested for a value of 43, indicating that an entry with the specified **STOCK#** search item value already exists in the data set, or 16, indicating that the data set is full, or -23, indicating that the user's password does not grant write access to the data set.

If an entry is to be added to a detail set, the program may first check to see if the required entries exist in the manual masters linked to the detail set. Values must be provided for all search items and the sort item, if one is defined, of a detail data set entry.



## BASIC

### Read Entry (Serially)

```
10 DIM B$(10),P$(8),D1$(14),L1$(20),S1$(16),S2$(2)
20 INTEGER S(10),M,M1,M2
30 B$=" ORDERS;"
40 M1=1
50 INPUT "ENTER PASSWORD: ",P$(1;8)
60 INPUT "ENTER ACCESS MODE (1-8): ",M
70 CALL XDBOPEN(B$,P$,M,S[*])
80 IF S(1)<>0 THEN 9300
200 M2=2
210 D1$="SUP-MASTER;"
220 L1$="SUPPLIER,STATE;" ←—————readlist
230 CALL XDBGET(B$,D1$,M2,S[*],L1$,S1$,S2$,)
240 IF S(1)<>11 THEN 270
250 GOSUB 900
260 GOTO 230
270 IF S(1)<>0 THEN 320
280 PRINT "STOCK#= ",S1$,"DESCRIPTION= ",S2$
290 INPUT "CONTINUE (Y OR N)? ",X$
300 IF X$(1,1)="Y" THEN GOTO 230
310 GOTO 9900
320 IF S(1)=-21 THEN 360
330 PRINT "DBGET FAILURE"
340 CALL XDBEXPLAIN(S[*])
350 GOTO 9900
360 PRINT "YOU DO NOT HAVE ACCESS TO THIS DATA"
370 GOTO 9900

.
900 (routine to rewind data set)
.
.
.
9300 (same as XDBOPEN example)
.
.
.
9900 (close data base)
```

To read the next entry of the SUP-MASTER data set, a mode of 2 is used. This directs the XDBGET (and DBGET) procedure to perform a forward serial read. In the example, the list in the L1\$ string specifies two data items to be read. After returning to the calling program, the S1\$ string contains the STOCK# data item value and S2\$ contains the DESCRIPTION data item value. The argument parameter is ignored if mode equals 2, therefore, a null string may be used for this parameter.

If an end-of-file is encountered, the condition word is set to 11. In this case, if the user wants to continue, the routine rewinds the data set and tries the read again. A rewind routine is shown later in the examples of the XDBCLOSE procedure. The rewind reinitializes the current record pointer so that the next request for a forward serial read will read the first entry in the data set.

If the user's password does not allow read access to the data, a condition word of -21 is returned.

## Read Entry (Calculated)

```

10 DIM B$[10],P$[8],C$[20],S0$[8]
20 INTEGER S[10],M1,M
30 B$=" ORDERS;"
40 M1=1
50 DEF INTEGER FNI(X)=X
60 INPUT "ENTER PASSWORD: ",P$[1;8]
70 INPUT "ENTER ACCESS MODE (1-8): ",M
80 CALL XDBOPEN(B$,P$,M,S[*])
90 IF S[1]<>0 THEN 9300
300 INPUT "ENTER STOCK# OR / TO TERMINATE: ",S0$[1;8]
310 IF S0$[1,1]="/" THEN GOTO 9900
320 CALL XDBGET(B$,"PRODUCT ",FNI(7),S[*],"DESCRIPTION;",C$,S0$)
330 IF S[1]<>17 THEN GOTO 360
340 PRINT "NO SUCH STOCK NUMBER"
350 GOTO 300
360 IF S[1]=0 THEN GOTO 410
370 IF S[1]=-21 THEN 430
380 PRINT "DBGET FAILURE"
390 CALL XDBEXPLAIN(S[*])
400 GOTO 9900
410 PRINT S0$,C$
420 GOTO 300
430 PRINT "YOUR PASSWORD DOES NOT GRANT ACCESS TO DATA REQUESTED"
440 GOTO 9900
.
.
.
9300 (same code as XDBOPEN example)
.
.
.
9900 (close data base)
.
.
.

```

To locate the **PRODUCT** data set entry which has a **STOCK#** search item value equal to the one entered in **S0\$** by user, a calculated read is used. The mode is 7 and the item to be read is **DESCRIPTION**. After **XDBGET** returns control to the calling program, the description is in **C\$**. If no entry exists with the specified **STOCK#** value, the condition word is 17. If the user does not have read access to the requested data, a condition word of -21 is returned.

## BASIC

### Read Entry (Backward Chain)

```
10 DIM B$(10),P$(8),I1$(6),A$(8),A1$(16)
20 INTEGER S(10),M1,M,M6
30 B$=" ORDERS;"
40 M1=1
50 M6=6
60 INPUT "ENTER PASSWORD: ",P$(1;8)
70 INPUT "ENTER ACCESS MODE (1-8): ",M
80 CALL XDBOPEN(B$,P$,M,S[*])
90 IF S[1]<>0 THEN GOTO 9300
300 INPUT "ENTER LASTSHIPDATE (YYMMDD) OR E TO EXIT: ",I1$(1;6)
310 IF I1$(1,1)="E" THEN GOTO 9900
320 CALL XDBFIND(B$,"INVENTORY ",M1,S[*],"LASTSHIPDATE;",I1$)
330 IF S[1]<>17 THEN GOTO 360
340 PRINT "NO SHIPMENTS ON THAT DATE"
350 GOTO 300
360 IF S[1]=0 THEN GOTO 410
370 IF S[1]=-21 OR S[1]=-52 THEN GOTO 480
380 PRINT "DBFIND FAILURE"
390 CALL XDBEXPLAIN(S[*])
400 GOTO 9900
410 CALL XDBGET(B$,"INVENTORY;",M6,S[*],"STOCK#,SUPPLIER;",A$,A1$)
420 IF S[1]<>14 THEN GOTO 450
430 PRINT "NO MORE SHIPMENTS ON THIS DATE"
440 GOTO 300
450 IF S[1]<>0 THEN GOTO 500
460 PRINT A$,A1$
470 GOTO 410
480 PRINT "YOUR PASSWORD OR ACCESS MODE DOES NOT GRANT ACCESS TO DATA"
490 GOTO 9900
500 PRINT "DBGET FAILURE"
510 GOTO 390
.
9300 (same as XDBOPEN example)
.
9900 (close data base)
```

First the XDBFIND procedure is called to determine the location of the first and last entries in the chain. The call parameters include the detail data set name, the name of the detail search item used to define a path with the DATE-MASTER data set, and the search item value of both the master entry containing the chain head and the detail entries making up the chain. The search item value is requested from the user and stored in I1\$, for example, the user may enter 841214.

If no entry in the DATE-MASTER has a search item value entered, the condition word will be 17. If the user does not have read access to the data, a condition word of -21 or -52 is returned.

If the XDBFIND procedure executes successfully, a call to the XDBGET procedure with a mode parameter of 6 reads the last entry in the chain. Subsequent calls to XDBGET with the same mode read backward through the chain until the first entry has been read. If the condition word is 14, the beginning of the chain has been reached and no more entries are available, or there are no entries in the chain.

If an entry is successfully read, the program uses the STOCK# value stored in A\$ and the SUPPLIER value stored in A1\$ and then returns to statement 350 to read another entry in the chain.

## Update Entry

```

10 DIM B$(10),P$(8),D1$(12),I2$(16),A5$(26),S9$(16)
20 INTEGER S(10),M
30 B$=" ORDERS;"
40 DEF INTEGER FNI(X)=X
50 D1$="SUP-MASTER "
60 I2$="STREET-ADDRESS;"
70 INPUT "ENTER PASSWORD: ",P$(1;8)
80 M=3
90 CALL XDBOPEN(B$,P$,M,S[*])
100 IF S(1)<>0 THEN 9300
200 INPUT "ENTER SUPPLIER OR / TO TERMINATE: ",S9$(1;16)
210 IF S9$(1,1)="/" THEN GOTO 290
220 CALL XDBGET(B$,D1$,FNI(7),S[*],I2$,A5$,S9$)
230 IF S(1)=-21 THEN GOTO 290
240 IF S(1)=0 THEN GOTO 310
250 IF S(1)=17 THEN GOTO 430
260 PRINT "DBGET FAILURE"
270 CALL XDBEXPLAIN(S[*])
280 GOTO 9900
290 PRINT &
    "YOUR PASSWORD OR ACCESS MODE DOES NOT ALLOW ACCESS TO THIS DATA"
300 GOTO 9900
310 PRINT "CURRENT ADDRESS: ",A5$
320 INPUT "ENTER NEW ADDRESS: ",A5$(1;26)
330 CALL XDBUPDATE(B$,D1$,FNI(1),S[*],I2$,A5$)
340 IF S(1)<>42 THEN GOTO 370
350 PRINT "YOU ARE NOT ALLOWED TO ALTER THIS ITEM"
360 GOTO 200
370 IF S(1)=0 THEN 410
380 PRINT "DBUPDATE FAILURE"
390 CALL XDBEXPLAIN(S[*])
400 GOTO 9900
410 PRINT "ADDRESS CHANGED"
420 GOTO 200
430 PRINT "NO SUCH SUPPLIER"
440 GOTO 200
.
.
9300 (same as XDBOPEN example)
.
9900 (close data base)

```

Before an entry can be updated it must be located. In this example, the entry is located with a calculated XDBGET that reads the STREET-ADDRESS item in the SUP-MASTER data set. The entry is located by using the SUPPLIER search item with a value supplied by the user. If the read is successful, the current address is printed and the application program user is prompted for the new address which is moved into A5\$. The XDBUPDATE procedure is then called to alter the STREET-ADDRESS data item in the entry.

## BASIC

If the current user class number does not allow this item to be altered or the access mode does not allow updates to take place, the condition word 42 is returned.

A null list can be used with DBGET to locate an entry to be updated.

### Delete Entry (with Locking and Unlocking)

```
10 DIM B$[10],P$[8],D1$[12],S9$[16],A5$[16]
20 INTEGER S[10],M2,M1,M4
30 B$=" ORDERS;"
40 DEF INTEGER FNI(X)=X
50 D1$="SUP-MASTER "
60 INPUT "ENTER PASSWORD: ",P$[1;8]
70 M1=1
80 M2=2
85 M4=4
90 CALL XDBOPEN(B$,P$,M1,S[*])
100 IF S[1]<>0 THEN 9300
110 INPUT "ENTER SUPPLIER OR / TO TERMINATE: ",S9$[1;16]
120 IF S9$[1,1]="/" THEN GOTO 9900
130 CALL XDBLOCK(B$,D1$,M4,S[*])
140 IF S[1]<=0 THEN 170
150 PRINT "DATA SET IS BUSY. TRY AGAIN LATER."
160 GOTO 9900
170 IF S[1]=0 THEN 210
180 PRINT "DBLOCK FAILURE"
190 CALL XDBEXPLAIN (S[*])
200 GOTO 9900
210 CALL XDBGET(B$,D1$,FNI(7),S[*],"SUPPLIER;",A5$,S9$)
220 IF S[1]=0 THEN 330
230 IF S[1]=-21 THEN 280
240 IF S[1]=17 THEN 310
250 PRINT "DBGET FAILURE"
260 CALL XDBEXPLAIN(S[*])
270 GOTO 290
280 PRINT "YOUR PASSWORD DOES NOT GRANT ACCESS TO DATA SET"
290 GOSUB 9000
300 GOTO 9900
310 PRINT "NO SUCH SUPPLIER"
320 GOTO 430
330 CALL XDBDELETE(B$,D1$,FNI(1),S[*])
340 IF S[1]<>44 THEN GOTO 370
350 PRINT "INVENTORY ENTRIES EXIST, SUPPLIER CANNOT BE DELETED"
360 GOTO 430
370 IF S[1]=0 THEN GOTO 420
380 IF S[1]=-23 THEN 280
390 PRINT "DBDELETE FAILURE"
400 CALL XDBEXPLAIN(S[*])
410 GOTO 9900
420 PRINT "SUPPLIER DELETED"
430 GOSUB 9000
```

```

440  GOTO 110
      .
      .
      .
9000  CALL XDBUNLOCK(B$, "", M1, S[*])
9010  IF S[1]=0 THEN RETURN
9020  PRINT "DBUNLOCK FAILURE"
9030  CALL XDBEXPLAIN(S[*])
9040  GOTO 9900
9300  PRINT "DBOPEN FAILURE"
9310  CALL XDBEXPLAIN(S[*])
9320  STOP
9900  CALL XDBCLOSE(B$, "", FNI(1), S[*])
9910  IF S[1]=0 THEN STOP
9920  PRINT "DBCLOSE FAILURE"
9930  GOTO 9310
9999  END

```

In the example above, the program calls XDBLOCK to lock the SUP-MASTER data set. Since mode 4 is used, the program must check the condition word when DBLOCK returns control to verify that the data set is locked and the calling program has exclusive access to it. If this is so, the condition word is 0.

If the data is successfully locked, the program performs the necessary data base operations. In this case, it deletes an entry. Before the entry can be deleted, the current record of the data set must be that of the entry to be deleted. This record may be located by calling XDBGGET. The program may request the name of the supplier whose record is to be deleted and use XDBGGET in calculated mode to locate the appropriate entry. If entries in the INVENTORY data set exist that have the same SUPPLIER value as the entry to be deleted, the condition word is set to 44 and the entry is not deleted.

After the entry is deleted, the data set is unlocked by XDBUNLOCK.

A null list can be used with DBGET to locate an entry to be deleted.

## BASIC

### Request Data Set Information

```
10 DIM B$(10),P$(8)
20 INTEGER S(10),D2(7),M
30 B$=" ORDERS;"
40 INPUT "ENTER PASSWORD: ",P$(1;8)
50 INPUT "ENTER ACCESS MODE (1-8): ",M
60 CALL XDBOPEN(B$,P$,M,S[*])
70 IF S[1]<>0 THEN 9300
300 M=203
310 CALL XDBINFO(B$,"",M,S[*],D2[*])
320 IF S[1]=0 THEN 350
330 CALL DBEXPLAIN(S[*])
340 GOTO 9900
350 PRINT "YOU HAVE ACCESS TO ";D2[1];"DATA SETS AS FOLLOWS:"
360 FOR I=2 TO D2[1]+1
370 PRINT D2[I]
380 NEXT I
390 GOTO 9900
.
.
.
9300 (same as XDBOPEN example)
9900 (close data base)
```

The procedure call in this example obtains the numbers of data sets that are available to the current user class by specifying mode 203. If the user class number is 12 and the procedure executes successfully, the D2 array contains:

D2(1)	4	Access to 4 data sets.
D2(2)	2	Read access to data set 2.
D2(3)	-3	Modify access to data set 3
D2(4)	-5	and data set 5.
D2(5)	6	Read and possibly update access to data set 6.

## Rewind Data Set

```

10  DIM B$[10],P$[8],D1$[14],L1$[20],S1$[16],S2$[2]
20  INTEGER S[10],M,M1,M2
30  B$="  ORDERS;"
40  M1=1
    .
    .
    .
    (open data base)
    .
    .
    .
210  D1$="SUP-MASTER;"
    .
    .
    .
    (read data set serially)
    .
    .
    .
900  INTEGER M3
910  M3=3
920  CALL XDBCLOSE(B$,D1$,M3,S[*])
930  IF S[1]=0 THEN RETURN
940  PRINT "DBCLOSE FAILURE"
950  CALL XDBEXPLAIN(S[*])
960  GOTO 9900
    .
    .
    .
9900 (close data base)

```

To rewind the SUP-MASTER data set, a call to DBCLOSE is made with mode equal to 3. The dynamic status information in the DBU for SUP-MASTER is reset, including the current record number. If a serial read request encounters an end-of-file, this call resets the current record to the beginning of the data set and another serial read request reads the first entry in the data set.



## BASIC

### Close Data Base

```
10 DIM B$(10),P$(8)
20 INTEGER S(10),M
30 B$=" ORDERS;"
40 DEF INTEGER FNI(X)=X
.
.
.
9900 CALL XDBCLOSE(B$,"",FNI(1),S[*])
9910 IF S[1]=0 THEN STOP
9920 PRINT "DBCLOSE FAILURE"
9930 GOTO 9310
9999 END
```

This call closes the data base. It is issued after the program has completed all data base operations and before program termination.

### Print Error

```
10 DIM B$(10)
20 INTEGER S(10)
.
.
.
9310 CALL XDBEXPLAIN(S[*])
9320 STOP
```

A call to DBEXPLAIN prints a message on the \$STDLIST device which interprets the contents of the status array, S. This is the routine which is called to display the status in the preceding examples.

## Move Error to Buffer

```

10 DIM B$(10),P$(8),M$(72)
20 INTEGER S(10),M,M1
30 B$=" ORDERS;"
40 M1=1
50 INPUT "ENTER PASSWORD: ",P$(1;8)
60 INPUT "ENTER ACCESS MODE (1-8): ",M
70 CALL XDBOPEN(B$,P$,M,S[*])
80 IF S[1]<>0 THEN 9300
90 PRINT "DATA BASE OPENED"
100 GOTO 9900
.
.
.
9300 PRINT "DBOPEN FAILURE"
9310 CALL XDBERRORS(S[*],M$)
9320 PRINT M$
9330 STOP

```

In this example, a call to DBERROR returns one of the messages appropriate to the current condition word. For example, if the condition word is equal to 16, the message returned in M\$ is THE DATA SET IS FULL. Note that the length parameter need not be included since the logical length of M\$ is set by XDBERROR.

# RPG

The following restrictions apply to TurboIMAGE data bases used with RPG:

1. Data is added and retrieved as complete entries, in other words, you cannot read or modify single items through RPG. Therefore, if the RPG program is to read an entry from a data set, the specified password must correspond to a user class number allowing read access to all data items in the entry. If the RPG program is to write an entry to a data set, the password must correspond to a user class number allowing write access to all data items in the entry.
2. Since entries are handled in this way, data sets to be used with RPG programs are sometimes defined with one-item entries. However, if you intend to use QUERY with the data set you may need to define more items.
3. Only one search item can be used to reference a data set in a program unless the data set is defined as more than one file, or you are doing an ISAM simulation and processing between limits. (Consult the *RPG/3000 Reference Manual* for more information.)
4. RPG supports all the DBGET procedure input modes except reread. It provides two additional modes:
  - Simulated indexed sequential read, forward and backward.
  - Read down chain until key changes.
5. Since RPG file names cannot exceed 8 characters and can contain no special characters, a file specification for the SUP-MASTER data set or DATE-MASTER data set should have file names such as SUP and DATE with the full names given in a data set name record.

## CAUTION

RPG versions prior to A.06.04 allow only 1 to 15 characters for data set and data item name, not the 1 to 16 characters as allowed by TurboIMAGE.

## RPG Programs and TurboIMAGE

To use a TurboIMAGE data base through RPG application programs you must describe the data base with File Description specifications. A data set may be described by more than one File Description specification to allow you to access it in more than one way, for example, performing both serial and chained reads or using two different search items (keys). The File Description specification and its continuation records specify:

- a TurboIMAGE file by naming both the data base and a data set within it.
- a search item name.
- an access mode.
- a password.
- an input/output mode for the file.

In addition, you can add and delete entries with special RPG Output specifications.

Complete instructions for using a TurboIMAGE data base through RPG programs are given in the *RPG/3000 Reference Manual*.

Note that an RPG program can use any of the three modes of locking allowed for a TurboIMAGE data base: data base locking, data set locking, and data entry locking. For a discussion of how to select and implement the correct locking mode, refer to the RPG reference manual. (Note that the data entry locking is called "record level locking" in the RPG manual.)

Figure 6-7 contains a sample RPG program which reads the SALES entries associated with a particular stock number and prints the contents in a report. The File Description specifications include:

- line 0003--a description of the SALES file as a chained input file with fixed length records 38 bytes long. The processing mode used for the data set is random. The key field is 8 bytes long and contains alphanumeric data. The file organization code M signifies a TurboIMAGE file. The SALES file name is a logical data set name, in other words, it can be a reminder of the actual SALES data set name (see discussion of line 0007 below).
- line 0004--a data base name record specifying the ORDERS data base, an access (open) mode of 3 (exclusive access), and input/output mode C (chained sequential read).
- line 0005--an ITEM name record specifying the STOCK# search item as the key.
- line 0006--a LEVEL identification record specifying the DO-ALL password.
- line 0007--a DSNAMES data set name record specifying the SALES data set. This is the actual data set name and overrides the file name (SALES), which is a logical name identifying the data set.
- line 0008--a description of the INPUT file as a demand file with fixed length records 8 bytes long. The file's device is designated as \$STDIN. A file equation may also be used to alter the INPUT and PRINT files.
- line 0009--a description of the PRINT file as an output file of variable length records which are at most 80 characters long. The file's device is designated as \$STDLIST. Printing will be done at the user's terminal screen, if the program is run interactively. A file equation may also be used to alter the device designation.

The input specifications describe:

- lines 0010 through 0018--a SALES data entry with five binary, two numeric (ASCII), and one character (ASCII) data items.
- lines 0019 through 0020--an INPUT record of 8 bytes with a field named ISTOCK.
- lines 0021 through 0032--Calculation specifications, request input from the INPUT file which was specified \$STDIN in line 0008. The INPUT file may be specified outside the program, using the MPE :FILE command before executing the program. They also read the SALES entries with values equal to the stock number entered, print the information, and, when the end of chain is encountered, request another stock number. Note that in line 0030, indicator 12 (in columns 54-55) is set on whenever no record is returned by the CHAIN operation, and indicator 11 (in columns 56-57) is set on when the end of chain is encountered.

```

0001  $CONTROL USLINIT,NAME=SALES1
0002  H              L              X
0003  FSALESDS IC  F      38R 8AM      DISC
0004  F              KIMAGE ORDERS3C
0005  F              KITEM  STOCK#
0006  F              KLEVEL DO-ALL
0007  F              KDSNAMESALES
0008  FINPUT  ID  F      8      $STDIN
0009  FPRINT  O  V      80      $STDLIST
0010  ISALESDS AA
0011  I              B  1  40ACCT
0012  I              5  12 STOCK#
0013  I              B  13 140QTY
0014  I              B  15 182PRICE
0015  I              B  19 222TAX
0016  I              B  23 262TOTAL
0017  I              27 320PDATE
0018  I              33 380DDATE
0019  IINPUT  BB
0020  I              1  8 Istock
0021  C              SETOF 1211
0022  C              SETON 15
0023  C              EXCPT
0024  C              SETOF 15
0025  C              READ INPUT LR
0026  C              SETON 16
0027  C  NLR          EXCPT
0028  C              SETOF 16
0029  C              LOOP TAG
0030  C  NLR          Istock CHAINSALESDS 1211
0031  C  NLR          EXCPT
0032  C  NLRN11N12    GOTO LOOP
0033  OPRINT  E 2      16
0034  O              10 "ACCOUNT"
0035  O              19 "STOCK#"
0036  O              28 "QUANTITY"
0037  O              36 "PRICE"
0038  O              43 "TAX"
0039  O              52 "TOTAL"
0040  O              62 "PURCHASED"
0041  O              72 "DELIVERED"

```

Figure 6-7. Sales Transaction Display Program

0042	O	E 1	N11N12		
0043	O			ACCT Z	10
0044	O			STOCK#	19
0045	O			TOTAL J	52
0046	O			TAX J	44
0047	O			PRICE J	36
0048	O			QTY J	26
0049	O			PDATE Y	62
0050	O			DDATE Y	72
0051	O	E 22	12		
0052	O			15	"NO SUCH STOCK#"
0053	O	E 21	15		
0054	O			20	"ENTER STOCK# OR :EOD"

**Figure 6-7. Sales Transaction Display Program (Continued)**

- lines 0033 through 0050--Output specifications, describe a report with column headings for each item and one-line records for each entry. The ACCT item is edited with a Z edit specification, the TOTAL, TAX, PRICE, and QTY items with a J edit specification, and the PDATE and DDATE items with a Y edit specification.
- lines 0051 through 0054--last Output specifications, describe the message to be printed if there is no entry with the requested stock number value and the message which prompts for the stock number or the end of program..

The following figure shows SALES1 program execution. The user is prompted to enter stock number and is given a report heading and data pertaining to the SALES data set records which contain stock information.

# RPG

:RUN SALES1

Program: SALES1 = SALES.TEST.IMAGE

MON, DEC 31, 1984, 8:32 AM

ENTER STOCK# OR :EOD

11

ACCOUNT	STOCK#	QUANTITY	PRICE	TAX	TOTAL	PURCHASED	DELIVERED
11 11		30	100.00	6.05	106.05	12/06/84	12/17/84
12 11		2	110.00	6.50	116.50	10/11/84	12/10/84
12 11		35	115.00	6.70	121.70	9/11/84	9/15/84
11 11		12	117.00	6.90	123.90	12/03/84	12/05/84
11 11		13	130.00	7.50	137.50	10/30/84	11/02/84

ENTER STOCK# OR :EOD

16

ACCOUNT	STOCK#	QUANTITY	PRICE	TAX	TOTAL	PURCHASED	DELIVERED
12 16		20	120.00	6.86	126.86	12/10/84	12/24/84

ENTER STOCK# OR :EOD

15

ACCOUNT	STOCK#	QUANTITY	PRICE	TAX	TOTAL	PURCHASED	DELIVERED
10 15		32	15.00	1.29	16.29	12/04/84	12/17/84
11 15		10	10.00	.90	10.90	12/04/84	12/06/84
10 15		150	17.00	1.42	18.42	10/31/84	11/05/84

ENTER STOCK# OR :EOD

19

NO SUCH STOCK#

ENTER STOCK# OR :EOD

17

ACCOUNT	STOCK#	QUANTITY	PRICE	TAX	TOTAL	PURCHASED	DELIVERED
10 17		100	16.50	1.36	17.86	10/11/84	11/01/84
12 17		10	16.50	1.36	17.86	10/11/84	10/27/84
13 17		10	17.50	1.48	18.98	12/10/84	12/17/84
13 17		10	17.70	1.52	19.22	11/29/84	12/01/84

ENTER STOCK# OR :EOD

:EOD

Pgm-End: SALES1 = SALES.TEST.IMAGE

MON, DEC 31, 1984, 8:40 AM

Figure 6-8. Sample SALES1 Execution

# MAINTAINING THE DATA BASE

## SECTION

## 7

The TurboIMAGE data base is initialized and maintained through various TurboIMAGE utility programs. The utility programs include the following:

DBUTIL	A utility program used to create and maintain the data base.
DBUNLOAD	Copies data to specially formatted tape or serial disc volumes.
DBLOAD	Loads data from backup volumes (DBUNLOAD tape or serial disc) into the data base.
DBSTORE	Stores data base to tape or serial disc.
DBRESTOR	Copies data base from backup volumes (DBSTORE tape or serial disc) to disc.
DBRECOV	Recovers data base from a log file.

This section contains a discussion of the procedures to be followed in performing tasks such as restructuring the data base, logging transactions, and recovering the data base in the event of a system failure. Use this section together with Section 8 which gives the syntax of the various utility programs and commands.

Utility programs may be run in either job or session mode. DBUTIL, DBSTORE, DBRESTOR, DBUNLOAD, and DBLOAD all require the user to be logged on in the group and account which contains the data base root file. Consequently, these programs may not be used with a remote data base unless you initiate a remote session and run the utility as part of that session. These programs do not allow you to use the :FILE command to equate a data base or data-base-access file. DBRECOV is an exception, since :FILE commands are permissible, and since you need not be logged on under the same group and account as the log file. However, DBRECOV has the same remote session requirement for remote data base access as the other utility programs.

You may operate the utility programs as long as you are the data base creator, or know the maintenance word. If no maintenance word is defined, only the data base creator can execute the other utility programs and the DBUTIL commands that require a maintenance word.



## RESTRUCTURING THE DATA BASE

It is possible to make certain changes to the structure of an existing data base without having to write special programs to transfer data from the old data base to the new one. The general sequence of operations which you use to do this is:

1. Run DBUNLOAD on the old data base, copying all the data entries to tape or serial disc.
2. Purge the old data base using DBUTIL >>PURGE.
3. Redefine the data base using the same data base name and create a new root file with the Schema Processor.
4. Use the DBUTIL >>CREATE command to create and initialize the data sets of the new data base.
5. Run DBLOAD on the new data base using the tape or serial disc created in step 1 to put the old data into the new data base.

The above procedure provides only limited structural changes to the schema. DBLOAD does not prohibit other changes, however there is no guarantee the data will be consistent. Schema changes that yield correctly transformed data bases and always result in a good transformation follow (Allowed Structural Changes).

### Allowed Structural Changes

Any of the following schema changes, alone or combined, which are acceptable to the Schema Processor will always result in a successfully transformed data base:

- Adding, changing, or deleting passwords and user class numbers.
- Changing a data item or data set name and all references to it.
- Changing data item or data set read and write class lists.
- Adding new data item definitions.
- Removing or changing definitions of unreferenced data items.
- Increasing data set capacities.
- Adding, deleting, or changing sort item designators.

## Conditional or Unsupported Structural Changes

The following structural changes are legitimate only in some circumstances and may result in data set discrepancies or lost data:

- Changing primary paths.
- Adding new data items to the original end of a data entry definition.
- Removing data items from the original end of a data entry definition.
- Changing an automatic master to a manual master or vice versa.
- Changing the native language definition for the data base.
- Adding or deleting a data set at the end of the schema.

These are the unsupported schema changes. DBLOAD does not prohibit other changes, however there is no guarantee that the data will be consistent. A change must be judged in light of the particular data base and the functioning of DBUNLOAD and DBLOAD, described later in this section. Basically, all entries from an old data set are put into the corresponding data set, except that no entries are directly put into automatic masters. The entries are truncated or padded with binary zeros as necessary to fit the new data set's entry length. DBUNLOAD and DBLOAD always handle full entries, without regard to item positions or lengths. If the new data set's entry is defined with the items in a different order than the old data set, DBLOAD may not fail but the data set content will nevertheless be invalid. For example, data of type real may now occupy the position of a character type item.

In some circumstances, the load completes, but data is lost, for example, if a data set's capacity has been reduced in the new data base to a number less than the number of that data set's entries on the tape or serial disc.

An unsupported or conditional schema change is adding or deleting data sets. As data sets are loaded by number, additions and deletions should be made to the end of the schema. The number of the data set is determined by the sequence of order the data sets were entered in the schema file. Data set one would correspond with the first data set appearing in the schema (or root file). DBLOAD will always return a warning if it detects a discrepancy between the number of data sets defined in the schema and the number of data sets on the DBLOAD media, but you can allow the DBLOAD to continue after the warning if you are confident that the data base is not corrupted.

## MAKING A DATA BASE BACKUP COPY

A backup copy of the data base should be made prior to using the data base whenever there is a possible need for recovering the data base following a system failure. The data base administrator uses the DBSTORE TurboIMAGE utility to store a copy of the data base with flags (access disabled, recovery enabled, logging enabled) set as specified in Logging Installation, later in this section. In addition, since the correspondence between log files and backup data bases is crucial, DBSTORE sets a DBSTORE flag in the data base root file before storing the data base, along with a time stamp designating the date and time of the DBSTORE operation.

The DBSTORE flag is cleared by the first modification to the data base (DBPUT, DBDELETE, or DBUPDATE) indicating that the data base no longer corresponds to the stored copy. Before logging is enabled, DBUTIL checks the DBSTORE flag to ensure that the working data base is the same as the backup copy data base. For example, suppose a data base is stored and some modifications to the data base are made before logging is enabled. If the administrator then tries to enable logging, DBUTIL, seeing that the DBSTORE flag has been cleared, prints a message warning that the present state of the (modified) data base does not correspond to the stored version. If the message is ignored, the resulting log file will not contain all of the transactions that actually occurred against the working data base. Consequently, a recovery using the stored copy and the incomplete log file may fail or yield erroneous results. The following is an example of how to run DBSTORE.

```
:RUN DBSTORE.PUB.SYS
WHICH DATA BASE?  ORDERS
DATA BASE STORED
END OF PROGRAM
```

When multiple data bases and files are involved, you can use the MPE command STORE to collectively copy them to tape or serial disc and, if necessary, collectively restore them by using the MPE command RESTORE. However, all data bases and files must reside in one group or account and you must have account manager and privileged mode (PM) capability to use this method. Note that when using the :STORE command no time stamp (signifying the date and time the backup copy was made) will be set in the data base. For additional MPE command information refer to the *MPE Commands Reference Manual*.

## DATA BASE RECOVERY OPTIONS

Two levels of data base recovery following a system failure are provided within TurboIMAGE: Intrinsic Level Recovery (ILR), and Roll-Back Recovery or Roll-Forward Recovery at the transaction level. These recovery options ensure the physical and logical integrity of the data base following a system interruption or possible system failure. The level of recovery to be used is determined by the data base administrator, and is based upon available data base backup and logging resources in addition to user performance requirements.

In addition to the recovery methods mentioned above, a system can be set up for constant access or "high availability" and have a controlled maintenance using a new feature of DBRECOV called STOP-RESTART. Backups and down-time can be regulated with a maintenance method called the mirror data base. This method consists of two identical data bases on two separate computer systems. The mirror data base resides on the secondary system and is maintained with user logging, DBRECOV, and periodic DBSTORE's.

Intrinsic Level Recovery provides recovery of intrinsics that were interrupted during execution. Execution of ILR is automatic and transparent to the user. Roll-back and roll-forward recovery require the use of the user logging facilities and the TurboIMAGE utility DBRECOV. Execution of the roll-back and roll-forward recovery options are directed by the data base administrator. Appendix G "Recovery and Logging Quick Reference" offers a brief description of the recovery options. This appendix also lists benefits and disadvantages of logging to disc and logging to tape, and gives sample job streams for recovery and logging cycles. Appendix G may be used, along with information given in this section, to determine which type of recovery to use.

## INTRINSIC LEVEL RECOVERY

When Intrinsic Level Recovery (ILR) is enabled, TurboIMAGE automatically logs each DBPUT and DBDELETE to an internal ILR file. Since ILR is only concerned with data base structure, only the most recent (or last) DBPUT or DBDELETE is noted in the ILR file.

The ILR file is created when ILR is enabled. TurboIMAGE opens the ILR file the first time a user program opens the data base for access. When the last user process closes the data base, TurboIMAGE automatically closes the associated ILR file. If recovery is needed the next time the data base is opened, TurboIMAGE recovers the data base automatically.

A single extra data segment called the Intrinsic Level Control Block (ILCB) is allocated upon the first DBOPEN of the data base. The ILCB is used as an intermediate staging area for the buffers that will be modified by each DBPUT or DBDELETE. If the data base is enabled for ILR, the first access of the data base by DBOPEN will examine the ILR file. If recovery is necessary, the buffers logged in the ILR file are posted to the data sets and the interrupted intrinsic is redone at the end of the DBOPEN time.

If a DBPUT or DBDELETE intrinsic is interrupted by a system failure or abnormal termination of TurboIMAGE and fails to complete execute, TurboIMAGE completes the intrinsic so that the newly opened data base appears as if the interrupted intrinsic had completed normally. Chains are reconstituted automatically so that the internal structure of the data base remains consistent.

As TurboIMAGE does not allow a program abort to interrupt a DBPUT or DBDELETE, ILR is not needed following an abnormal program termination (user program abort). An interrupted DBUPDATE, however, does not require the use of ILR, as DBUPDATE completes execution in one autonomous write.

The following is an example of enabling a data base for Intrinsic Level Recovery.

```
:RUN DBUTIL.PUB.SYS
>>ENABLE dbname FOR ILR
    ILR is enabled
```

<b>NOTE</b>
-------------

ILR is enabled automatically when the roll-back feature is enabled, but must be manually disabled (using DBUTIL) when roll-back is disabled. Roll-forward recovery does not require ILR to be enabled, but it is recommended in order to eliminate the possibility of broken chains since ILR ensures physical consistency of the data base. If using roll-forward recovery do not restart logging following a system failure until after the roll-forward recovery process has completed. Use of ILR alone may cause inconsistency in the log file if logging is restarted without first running roll-forward recovery. ILR must be both manually enabled and disabled when used with the roll-forward feature.

## Using ILR

To enable a data base for ILR, run DBUTIL and use the >>ENABLE command. This causes TurboIMAGE to build and initialize an ILR file for the specified data base. At this time, TurboIMAGE sets a flag in the data base root file, and also sets a date and time stamp in both the ILR file and in the root file. This ensures a method of matching information logged to the ILR file with the appropriate data base root file.

The ILR file is a privileged file used only by TurboIMAGE. Its name is derived by adding two ASCII zeros to the root file name (root file name is "ORDERS", ILR file name is "ORDERS00").

To determine if ILR has been enabled for a data base, either use the DBUTIL command >>SHOW or programmatically call DBINFO in mode 402.

When ILR is enabled, the ILR file is stored or restored along with the data base by the the DBSTORE and DBRESTOR utilities. This happens automatically to ensure that the ILR file is retained during data base backups.

To discontinue using ILR on a data base, use the DBUTIL command >>DISABLE. When ILR is disabled by the user, TurboIMAGE first checks the ILR file to see if the data base needs recovery. If ILR recovery is pending and required, TurboIMAGE uses the existing ILR file to recover the data base, then purges the ILR file and clears the flag in the data base root file.

## Special Considerations

Before using ILR, consider the advantages and limitations of this type of recovery:

- Recovery after a system failure requires no more overhead than a single DBPUT or DBDELETE. However, logging the intrinsics increases the overhead on each DBPUT and DBDELETE. Logging DBPUT and DBDELETE intrinsics requires additional memory to move and write buffers to the ILCB.
- Output Deferred can not be used with ILR to defer writing modifications to the data base. However, Output Deferred should not be used when the primary objective is to assure the structural integrity of the data base. If you have enabled the AUTODEFER option in DBUTIL, ILR can not be used as the recovery method on the data base. The following message will be printed at the terminal if the user is attempting to enable ILR when AUTODEFER is already enabled on the data base:

AUTODEFER MUST BE DISABLED BEFORE ILR CAN BE ENABLED

The user should disable AUTODEFER and enable ILR using DBUTIL >>ENABLE command. (For more information on AUTODEFER refer to Section 8.)

## LOGICAL TRANSACTIONS AND LOCKING

Both the roll-back and roll-forward features operate at the transaction level and are designed to restore data bases to a consistent state, both structurally and logically following a system failure. The concept of a "logical transaction" is central to this process. A logical transaction is defined as a sequence of one or more procedure calls begun with a **DBBEGIN** and concluded with a **DBEND**. If **DBBEGIN** and **DBEND** are not used, TurboIMAGE considers each **DBPUT**, **DBDELETE**, and **DBUPDATE** as a single logical transaction. While a transaction is executing, the data base is considered to be in an inconsistent state.

For example, consider the manual master data set **CUSTOMER** in the **ORDERS** data base, with the addition of a new field, **YTDSALES**, indicating the total value of the year-to-date sales for each customer. A one-step transaction might involve updating a particular customer's address. However, adding a new sales item for a customer, which involves addition of an entry to the **SALES** detail data set as well as updating the **YTDSALES** item in the **CUSTOMER** master set, is a two-step transaction. The data base is consistent before the transaction begins, because the **YTDSALES** value corresponds exactly with the sum of the **TOTAL** values in the **SALES** detail set that are chained to that particular customer's account number. However, after the first modification, which might be adding the new **SALES** entry, this correspondence no longer holds, so the data base is said to be inconsistent with itself. After the second step, modifying the **YTDSALES** item in the **CUSTOMER** data set, the data base is returned to a consistent state.

If the system fails while the data base is being modified, two forms of damage to the data base could result. A logical inconsistency might result if the failure occurs between modifications of a multiple-step transaction, as the above example illustrates. Secondly, structural damage (such as broken chains) can result if the failure occurs during the execution of a TurboIMAGE procedure.

Since the recovery system is designed to restore the data base to a consistent state, those modifications belonging to transactions that failed to complete due to a system failure are suppressed by the recovery system. Consequently, although one or more data base modifications may be lost upon recovery, the resulting data base will be consistent. To this end, each user application should indicate the beginning and end of each transaction in the log file by appropriate use of the TurboIMAGE intrinsics, **DBBEGIN** and **DBEND**. (See Section 4 for more information.)

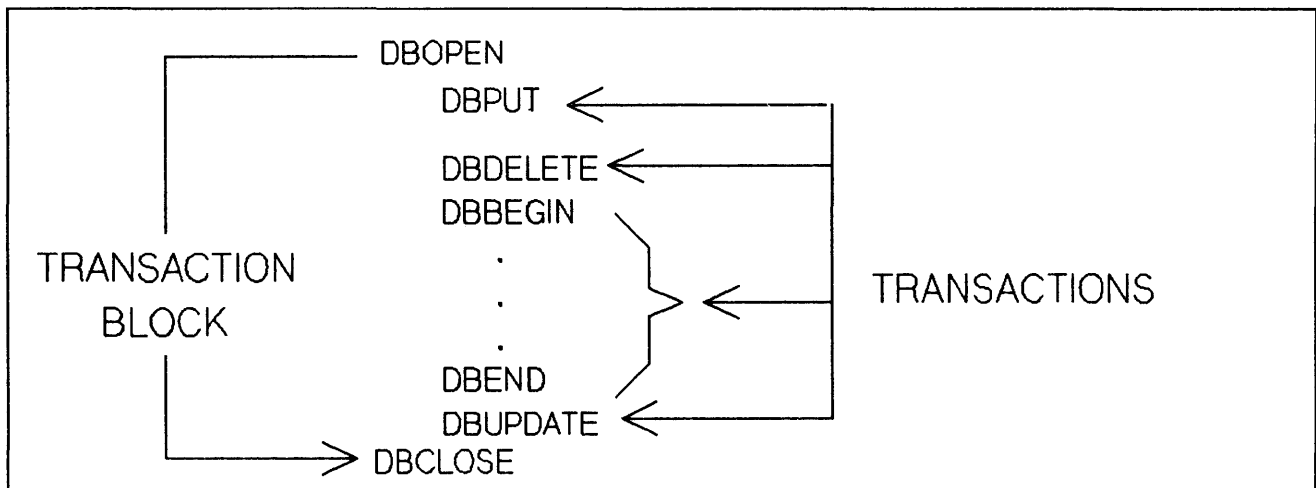


Figure 7-1. Transactions and Transaction Blocks

## Locking Requirements

DBRECOV requires that all multiple-intrinsic data base transactions execute independent of all other transactions. Transaction independence within the data base may be insured in a user program by releasing locks on data after a DBEND is called, thus eliminating the possibility of another user modifying the same data at the same time. An example may clarify the need for locking data to be modified.

Suppose transaction A consists of adding two records to the data base which are later modified by transaction B. Transaction B is dependent upon transaction A, as records must exist before they can be modified. Recall that a transaction is defined as a sequence of one or more modifications that transfer the data base from one consistent state to another. A data base may be in an inconsistent state during a transaction. Therefore, if transaction A and B are executing concurrently, transaction B may be viewing the data base in an inconsistent state and consequently could be generating invalid results. If transaction A is completed properly, this problem is avoided since transaction B cannot access the data until transaction A has released its locks.

A second problem due to inadequate locking affects suppression of transactions by the recovery system (see Figure 7-2). Suppose transaction A intends to add six records to the data base, and after adding three records, transaction B is executed by another process. Transaction B concurrently modifies one of the records added by transaction A and then completes. Suppose that at this time, the system fails and recovery is executed. Since transaction A failed to complete, all of its record additions will be suppressed. Since transaction B is dependent upon the suppressed transaction A, it cannot be recovered. DBRECOV is forced to suppress transaction B, even though it successfully completed during real-time processing. This potential problem could be avoided if transactions modifying the data base employ locking correctly. Transactions attempting to access the same data concurrently are serialized by the locking mechanism.



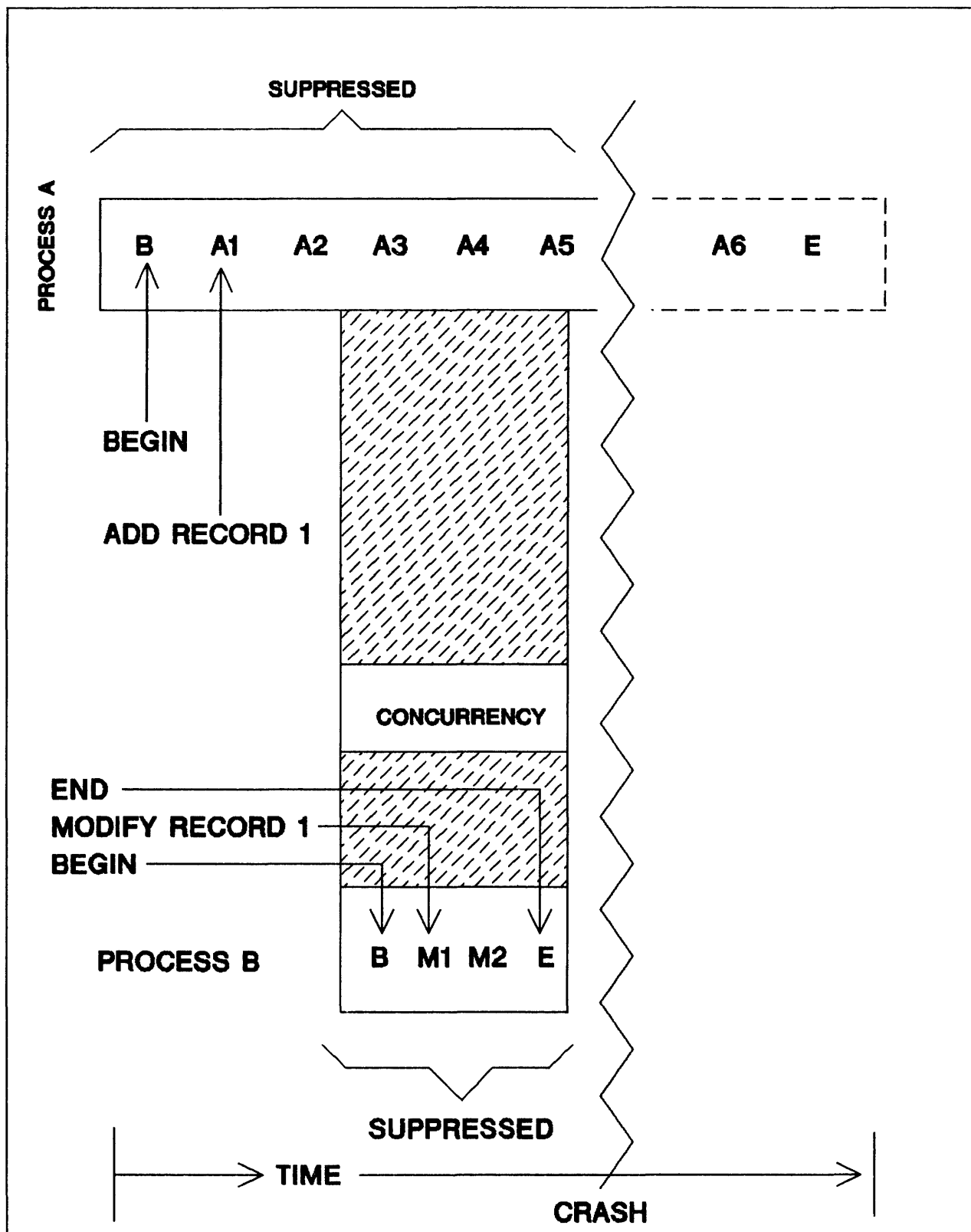


Figure 7-2. Suppression of Transactions Due to Inadequate Locking

The following provides examples of two recommended schemes for locking.

### Single Lock Strategy

```

DBLOCK for account 2,18,34
DBBEGIN                                << begin may precede DBLOCK call >>
DBGET data for account 2
DBUPDATE data for account 2
DBPUT data for account 34
DBGET data for account 18
DBDELETE data for account 18
DBEND
DBUNLOCK for all accounts              << DBUNLOCK must be last call >>

```

### Multiple Lock Strategy

```

DBLOCK account 2,34
DBBEGIN
DBGET data for account 2
DBUPDATE data for account 2
DBPUT data for account 34
DBLOCK for account 34
DBGET data account 18
DBDELETE data account 18
DBEND
DBUNLOCK for all accounts              << DBUNLOCK must be last call >>

```

### CAUTION

Use extreme caution when employing a multiple lock strategy requiring Multiple RIN (MR) capability (refer to Appendix D). Hewlett-Packard does not accept responsibility for possible deadlocks or system lockouts that could result from improper use of the MR capability.

In the first example (above) calling DBLOCK before DBBEGIN makes the transaction shorter in duration. It is recommended to call DBLOCK first since there is no way of knowing how long DBLOCK will have to wait to acquire the lock after the transaction is begun. For additional locking information refer to Section 4, "USING THE LOCKING FACILITY".

## Program Abort and Recovery Considerations

The TurboIMAGE logging and recovery system is not intended to be a solution for transactions which fail to complete in real time due to a program abort. Since subsequent transactions may be dependent on a transaction interrupted by a program abort, the recovery system will not suppress transactions that fail for this reason. Instead, TurboIMAGE will log a special DBEND to the log file so that the transaction can be recovered. This mechanism can be overridden with the NOABORTS control option in DBRECOV as long as all processes are stopped immediately after a program abort and the data base is restored and recovered. Any delay in executing recovery with the NOABORTS option could result in erroneous data or recovery failure due to transaction interdependence. Alternatively, when using roll-forward recovery the STOPTIME option could be used to restore transactions that logged up to a time preceding the program abort. (See Section 8, "DBRECOV".)

The utility DBRECOV can also recover transactions interrupted by an abnormal program termination if the NOABORTS option is used. This utility also allows you to create individual user recovery files. The information from these files then enables you to inform each user where to resume transactions within the data base.

Overhead required by the logging process depends on the number and type of modifications that are logged and the data base structure. The time needed for recovery depends on the number of transactions that were written to the log file following the last backup of the data base. Overhead and recovery time also depends on the type of recovery being used.

As a secondary function, the transaction logging system can be a useful tool for auditing. The log file is actually a programmatically accessible journal of all modifications to items in the data base, providing information about previous entries as well as the current state of the data base. The logging intrinsic DBMEMO, containing user text, provides a method of accessing and interpreting the log files for future reference.

The data base administrator is responsible for enabling or disabling the logging and recovery processes, generating backup data base copies, and for making logging a global function controlled at the data base level rather than at the individual user level.

The TurboIMAGE logging and recovery system is based upon the MPE user logging system. For further details of operation, and for the data format of log files on tape or disc, refer to *MPE System Manager/System Supervisor Reference Manual*, *Console Operator's Guide*, *MPE Commands Reference Manual*, and the *MPE Intrinsics Reference Manual*.

- **RECOVERY FROM A STREAM FILE.** A stream file may specify all of the data bases logging to one log file for recovery. If one of the data bases has not been restored at the time the stream file is run, recovery for that data base is prevented because recovery for that data base is disabled if the recommended procedures have been followed. (Refer to "Roll-Forward Recovery" later in this section for more information.) Recovery can be completed for all of the other specified data bases that have been restored from a backup copy with recovery enabled, as long as >CONTROL ERRORS is set appropriately. This means that ERRORS must be increased by one for each data base disabled for recovery, since an error message occurs each time a data base specified in the >RECOVER command is not enabled for recovery.

- **LOCKING AND TRANSACTION INTERDEPENDENCE.** In order to maximize the extent of recovery, locking should be employed while also logging in order to eliminate concurrent transaction interdependence. Locking by logical transaction (DBBEGIN, intrinsics, DBEND) guarantees the logical consistency of the data base. Locking by logical transaction is required for roll-back recovery in order to ensure that all incomplete transactions are backed out of the data base. Transaction locking is recommended for roll-forward recovery. Intrinsic Level Recovery logs the most recent (or last) intrinsic to a log file, and therefore does not utilize transaction interdependence.
- **QUIET PERIODS, RECOVERY BLOCKS AND STAGING DISC FILE.** A log file quiet period occurs at a point in time when no transactions are in progress on the log file. The log records between one quiet period and the next is called a recovery block (see Figure 7-3).

The recovery system reads recovery blocks into a temporary staging disc file before actually re-executing the transactions. The recovery process then applies all transactions which were not active (or in progress) at the time of the system failure. Only transactions within the same recovery block can possibly be suppressed due to concurrent transaction interdependence.

It is desirable to shorten the length of the recovery blocks and maximize the number of quiet periods. This may prove helpful in minimizing the number of transactions lost during recovery and in preventing record table overflow (see "DBRECOV >CONTROL" command in Section 8 for more information).

The average size of a recovery block will be a function of the number of concurrent processes running, the transaction rate, and the average elapsed time of each transaction.

In the event that the recovery system fails due to inadequate disc space for the staging disc size, a file equation specifying the formal designator TEMPLOG can be used to change the default staging disc size. The TEMPLOG is created with a default of 160,000 records. If recovery fails due to inadequate staging disc space then create TEMPLOG with 80,000 records (or half the size of the file). Then DBRECOV can be run again.

#### EXAMPLE

:FILE TEMPLOG;DISC=40000 ← *number of records*

- **MULTIPLE DATA BASE TRANSACTIONS.** Although the concept of a transaction has been specified only in terms of a single data base, certain applications will undoubtedly execute transactions that 'span' two or more data bases. There is currently no mechanism within TurboIMAGE to provide for the declaration of multi-data base transactions. Programmers may be tempted to call DBBEGIN twice (once for each data base), update both data bases, and then call DBEND twice in an attempt to implement this capability. However, a system failure during the 'window' between the two final calls to DBEND will result in the recovery of the transaction for one of the data bases, and its suppression on the other. Consequently, an application which uses this strategy should also have the capability to examine the recovery files to determine if this problem occurred, and if so, back out one of the data bases as needed, using >CONTROL STOPTIME or >CONTROL EOF.

- MPE CLEANUP MODE.** In the event of a system failure and subsequent "warmstart", MPE will attempt to clean-up any log files that were open at the time of the failure. The cleanup procedure involves writing any records left in the log system disc buffer file to the end of the log file. (When using roll-forward recovery records left in the memory buffer will still be lost.) The console operator has the option to cancel this cleanup procedure if the log file is on tape. The advantage of the procedure is that fewer log records written just prior to the failure are lost. The disadvantage for tape files is the time it takes for the tape to be rewound and sequentially scanned until the end-of-file is detected so that the remaining records can be appended to the end. In addition, a dedicated tape drive is required when logging to tape. The TurboIMAGE recovery program DBRECOV does not require the clean-up to be performed. If it is not performed, however, DBRECOV will most likely report a sequence or checksum error when the discrepancy caused by the failure is encountered. This will cause DBRECOV to assume the end-of-file has been reached.

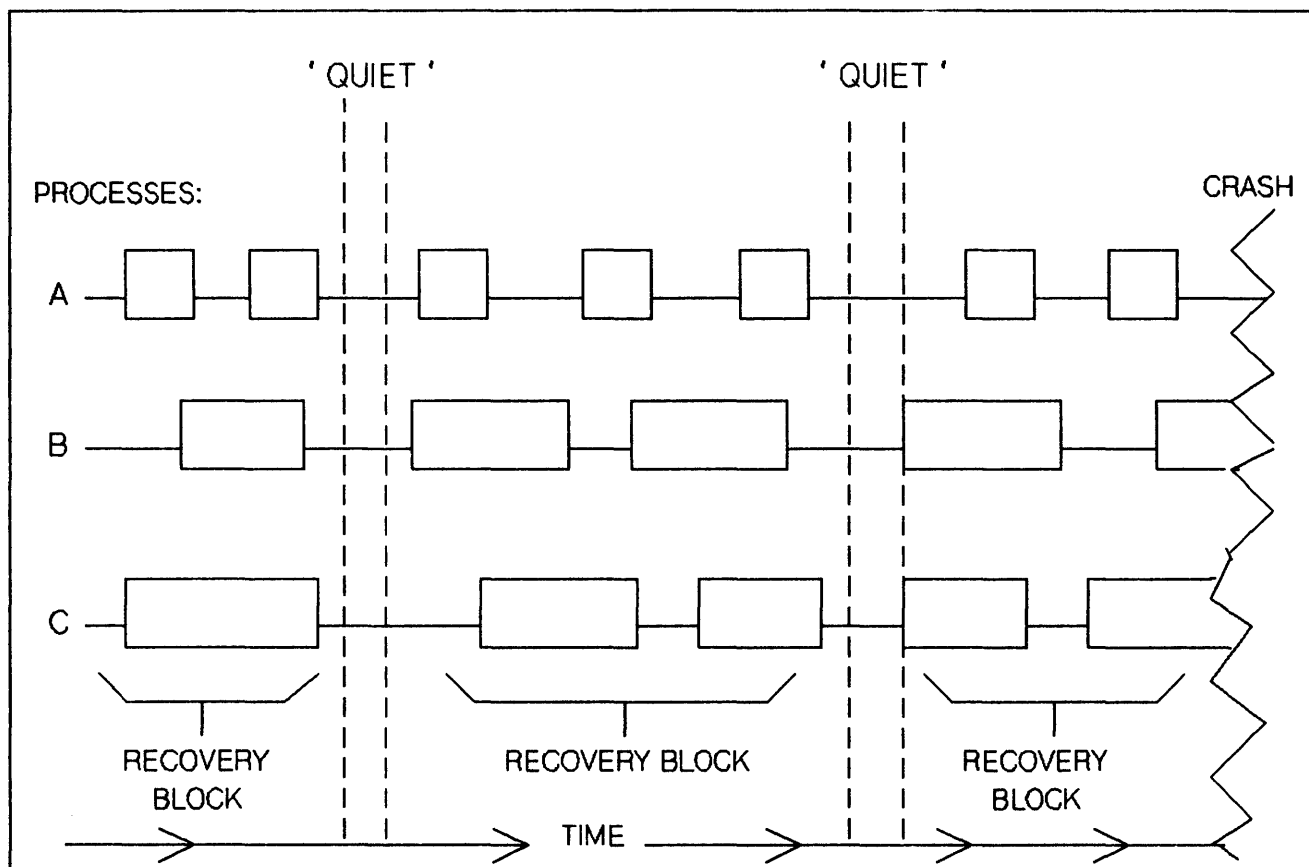


Figure 7-3. Quiet Periods and Recovery Blocks

## Recovery Tables

The first three of the following four tables are returned by every execution of the recovery system. The last table is returned only if the user recovery file feature is used.

```
*****
* 1                                PROCESS STATISTICS                                *
*                                                                              *
*LOG#  TIME  NAME  ACCOUNT  PROGRAM  DATABASE  TRANS  PUTS  DELS  UPS *
*-----*
* 1   15.45  TST   MKTG   INVENTRY  ORDERS    145   145    0    0 *
* 2   15.47  TST   MKTG   ORDENTRY  ORDERS    431   431    0    0 *
*****
```

```
*****
* 2                                DATABASE STATISTICS                                *
*                                                                              *
*  NAME  GROUP  ACCOUNT  OPENS  TRANS  PUTS  DELETES  UPDATES  *
*-----*
* ORDERS  TST   MKTG      2    576   576      0      0 *
*****
```

```
*****
* 3                                LOGGING SYSTEM                                *
*                                                                              *
* ...CREATOR...  RECORDS  DEV  .....LOGFILE..... *
* LOGID  NAME  ACCOUNT  PROCESSED  TYPE  NAME  GROUP  ACCOUNT *
*-----*
* ORDERLOG  TST   MKTG      640  DISC  ORDER001  TST   MKTG *
*****
```

```
*****
* 4                                RECOVERY SYSTEM                                *
*                                                                              *
*  FILE REFERENCE  USER  IDENT  RMODE  FMODE  *
*-----*
* PART1  SYS  MKTG  TST  MKTG  P1D1  1  1 *
* PART2  SYS  MKTG  TST  MKTG  P1D2  1  1 *
* PART3  SYS  MKTG  TST  MKTG  P1D3  1  1 *
*****
```

END OF PROGRAM

The tables provide the following information:

- 1 The **PROCESS STATISTICS** table lists the log number assigned to each process by the **OPENLOG** intrinsic, the logon name and account, program name, and transaction statistics. In this table there is one entry for each process that logged transactions to the log file. An asterisk will appear for any process that issued a **DBOPEN** without a corresponding **DBCLOSE** before the system failure. In roll-forward recovery the columns "TRANS, PUTS, DELS, UPS" indicate the number of transactions recovered. In roll-back recovery these columns and numbers indicate the number of transactions rolled-out.
- 2 In the table of **DATABASE STATISTICS**, the total number of transactions are given for each data base recovered. The columns "TRANS, PUTS, DELS, UPS" indicate the number of transactions recovered in roll-forward recovery, or rolled-back if using roll-back recovery.
- 3 The **LOGGING SYSTEM** table should have only one entry. It lists the log identifier for the log file that was accessed by the recovery system. The creator is the user who created the log identifier with the **:GETLOG** command. The number of records processed is usually greater than the number of transactions given in the other tables because some transactions require more than one log record, and there are header records and trailer records in each log file.
- 4 The **RECOVERY SYSTEM** table references the file to which the records were returned, the user name and identifier, and the *rmode* and *fmode* parameters specified in the **:FILE** commands. Note that all of these tables can be returned without recovering a data base by using the **>CONTROL STATS** option when running the recovery program. Roll-back recovery ignores the *rmode* parameter.

## LOGGING INSTALLATION

In order to prepare a data base for transaction logging, you must set a log identifier (*logid*) into the data base root file. The log identifier could be one associated with an existing log file, in which case you can go directly to Step 3 (below) if you know the log identifier and password. Note that, in order to recover a data base from the log file, you must either be the creator of the *logid* or have system manager capability. Assuming you intend to create a new log identifier, you should take the following steps:

- Acquire logging capability.
- Acquire log identifier.
- Set log identifier into data base, along with appropriate flags.
- Store a backup copy of the data base. (This step is required when using roll-forward recovery, recommended when using roll-back recovery.)
- Build log file if logging to disc.

### NOTE

This is a one-time procedure. The logging maintenance operations are performed on regular basis, perhaps daily (refer to "Maintaining Logging" below).

## 1. Acquiring Logging Capability

A user must have logging capability in order to use the following MPE commands: :GETLOG, :RELLOG, :ALTLOG, :CHANGELOG, :LISTLOG, and :SHOWLOGSTATUS. Logging capability is acquired through the MPE system manager and account manager commands. First the system manager provides the account logging capability by using the command :NEWACCT, or the command :ALTACCT if a new capability is being assigned to an established account:

```
:NEWACCT acctname,mgrname;CAP=capability list (include LG)
:ALTACCT acctname;CAP=capability list (include LG)
```

The account manager then can provide logging capability to individual users through the command :NEWUSER, or the command :ALTUSER if a new capability is being assigned to an established user:

```
:NEWUSER username;CAP=capability list (include LG)
:ALTUSER username;CAP=capability list (include LG)
```

For example:

```
:NEWACCT CAPE,RICK;CAP=LG,AM,AL,GL,SF,ND,IA,BA
:NEWUSER ILSA;CAP=LG,AL,GL,SF,ND,IA,BA
```

(CAP=LG must be included to provide logging capability)

Refer to the *MPE Commands Reference Manual* for other MPE user logging commands, including :RELLOG (removes a log identifier), :ALTLOG (alters an existing log identifier), and :LISTLOG (lists the current log identifiers).

Any errors or messages that are followed by (ULOGERR#) or (ULOGMSG#) are returned by MPE. Refer to the *MPE Console Operator's Guide* for more information.



## 2. Acquiring Log Identifier

A log identifier (*logid*) is an eight-character logical name that identifies a system logging process to which log records are passed. Acquire the log identifier from MPE by using the command :GETLOG. Other users can be allowed access to the log file by notifying them of the *logid* and its password. Users accessing the logging system directly through MPE must supply the identifier and the password on the OPENLOG intrinsic, in addition to having logging (LG) capability.

### Syntax

```
:GETLOG logid;LOG=logfile,{DISC/TAPE/SDISC/CTAPE}
      [;PASS=password] [;{AUTO/NOAUTO}]
```

### Parameters

<i>logid</i>	is the logging identifier to be established on the system. A string of up to eight characters that are meaningful to the user application.
<i>password</i>	is the password to be associated with the logging identifier. This parameter protects the log file from unauthorized access. Up to eight characters are allowed.
<i>logfile</i>	is an MPE file reference that identifies the actual file to which the log records are written. If the AUTO option is specified, the last three digits are numeric (from 001-999). The first log file created with :GETLOG must end with the last three digits equal to 001 if the AUTO option is used. (A warning message will be issued if the logfile does not end in 001.)
DISC/SDISC	is the device on which the log file is to reside. If the log file specified for the <i>logid</i> is a serial file then the AUTO/NOAUTO option will be ignored.
TAPE/CTAPE	is the device on which the log file is to reside.
AUTO	performs an automatic CHANGELOG when the disc log file becomes full.
NOAUTO	is the default. No CHANGELOG will be performed when the disc log file becomes full.

### Example

```
:GETLOG ORDERLOG;LOG=ORDER001,DISC;PASS=PASSLOG
```

When the disc log file becomes full the file is closed and logging is shut down. If the AUTO option is used user logging will initiate a CHANGELOG when the current log file becomes full. A new log file will be created with the same logfile name incremented by one in the last digit (ORDER002). This enables logging to continue uninterrupted, also creating a sequence of log files or a *log file set*.

### 3. Setting Log Identifier and Flags

The two previous commands were executed through the MPE user logging system. At this point, the data base administrator must interface TurboIMAGE to the MPE user logging system by storing the log identifier and password into the data base root file, using the DBUTIL program >>SET command, as shown in the example below:

```
:RUN DBUTIL.PUB.SYS
>>SET ORDERS LOGID = ORDERLOG
PASSWORD? *****
```

DBUTIL checks the validity of the *logid* with MPE, and reports a warning if the log identifier is not valid or the password is incorrect:

```
WARNING: non-existent LOGID
```

Once the log identifier has been set into the data base, the log identifier parameters should not be altered in order for the logging and recovery system to function correctly.

Next, the data base administrator uses DBUTIL to set three flags in the root file indicating that the data base is enabled for logging, the data base is disabled for user access, and the data base is enabled for recovery. Each of these flags is discussed below.

**ENABLE LOGGING FLAG.** Enabling logging ensures that a journal of all data base modifications will be logged and available for later use by the recovery system, if necessary. When the data base administrator enables the data base for logging, DBUTIL checks whether a backup data base copy has been stored with DBSTORE; if not, it issues a message warning the data base administrator to store the backup copy. When logging is first installed, since the data base is stored after logging has been enabled, the DBUTIL warning message can be interpreted as a prompt to store the data base. The command to enable logging, if using roll-forward recovery:

```
>>ENABLE dbname FOR LOGGING
WARNING: data base modified and not DBSTORED
```

If using roll-back recovery, >>ENABLE *dbname* FOR ROLLBACK. Once the logid is set and the log file built, the >>ENABLE command for roll-back automatically enables logging and ILR for the data base.

**DISABLE ACCESS FLAG.** By disabling the data base for user access, the administrator ensures that modifications can not be made to the data base. Any attempt to open the data base with an otherwise valid call to DBOPEN will return an error message. Access to the data base should be disabled before storing the backup copy, so that in the event of a system failure the data base will be restored with access disabled. This will prevent users from opening the data base and making modifications before recovery is executed.

Disabling access to the data base is also useful as a general security measure to prevent data base access at unauthorized times. The DBUTIL command for disabling access is shown below:

```
>>DISABLE dbname FOR ACCESS
```

**ENABLE RECOVERY FLAG.** Enabling the data base for recovery gives the TurboIMAGE recovery system access to the data base. The data base is stored with recovery enabled so that when it is restored, it will be ready for recovery. The DBUTIL command for enabling recovery is:

```
>>ENABLE dbname FOR RECOVERY
```

A DBSTORE can be done after the preceding flags have been set in the data base. A backup copy of the data base is highly recommended. Logging status can be checked by referring to the procedure in "DISPLAYING LOGGING STATUS", later in this section.

## 4. Building a Logfile for Logging to Disc

If the log file is to reside on disc rather than tape, the data base administrator must build the new file and allocate space for it on disc by using the MPE command :BUILD.

### Syntax

```
:BUILD logfile;CODE=LOG;DISC=[numrec][,numextents][,initialloc]]
```

### Parameters

<i>logfile</i>	is the name of the log file being built, as specified in the :GETLOG command.
<i>numrec</i>	is the maximum number of logical records. Maximum value allowed is 2,147,483,647. Default is 1023.
<i>numextents</i>	is the maximum number of disc extents; a value of from 1 to 32. Default is 8.
<i>initialloc</i>	is the number of extents to be initially allocated to the file at the time it is opened; a value of from 1 to 32. Default is 1.

### Example

```
:BUILD ORDER001;CODE=LOG;DISC=200000,20,7
```

(This step is required only when logging to disc)

If the default NOAUTO option is specified in the :GETLOG command care should be taken to ensure that disc log files are of sufficient size to prevent the end-of-file from ever being reached. This is because MPE does not automatically switch to a new disc file, but instead causes the associated log process to terminate when the log file is filled to capacity. Subsequent calls to TurboIMAGE intrinsics that require log records to be written will therefore fail. If this event occurs in the middle of a transaction, the data bases will be left in an inconsistent state. It will then be necessary to recover transactions with roll-forward or roll-back recovery. Reaching the end of a disc log file is therefore similar in effect to a system failure and should be carefully avoided. Consequently, disc log files should be built with a total capacity far exceeding their required size and consisting of many extents (up to 32) of which only enough to satisfy the expected capacity are initially allocated. The command :SHOWLOGSTATUS can be used to determine when to perform a :CHANGELOG to open a new log file.

## DISPLAYING LOGGING STATUS

The DBUTIL >>SHOW command can be used to display the log identifier and the status of the flags for access, recovery, and logging. The following example illustrates roll-forward recovery and the commands used to set the *logid* and flags into the data base, as presented in this section. If the steps regarding Logging Installation have been followed the data base can be stored. (Note that passwords will not appear on the terminal screen).

```
:RUN DBUTIL.PUB.SYS
HP 32215C.00.00 TurboIMAGE:  DBUTIL (C) COPYRIGHT HEWLETT-PACKARD CO. 1984
```

```
>>SET ORDERS LOGID=ORDERLOG
PASSWORD *****
```

```
LOGID: ORDERLOG IS VALID
      PASSWORD IS CORRECT
```

```
>>DISABLE ORDERS FOR ACCESS
      Access is disabled
>>ENABLE ORDERS FOR RECOVERY, LOGGING
      WARNING: Data base modified and not DBRESTORed
      Recovery is enabled
      Logging is enabled
```

```
>>SHOW ORDERS ALL
      For data base ORDERS
```

```
Maintenance word is not present.
```

```
Access      is disabled.
Autodefer   is disabled.
Dumping     is disabled.
Rollback recovery is disabled.
Recovery    is enabled.
ILR         is disabled.
Logging     is enabled.
Data base last stored on MON, DEC 10, 1984, 1:09 PM
Data base has not been modified since last store date.
Restart     is disabled.
Subsystem access is READ/WRITE.
```

```
LOGID:  ORDERLOG is valid
        password is correct
The language is 0:NATIVE-3000.
```

```
BUFFER SPECIFICATIONS:
30(1/120)
```

```
No other users are accessing the data base.
>>
```

## MAINTAINING LOGGING

Each data base administrator should determine a log maintenance cycle for the data base. For example, suppose the data base is maintained on a daily cycle. This means that at the beginning of each day, the log process is initiated from the console with the :LOG command and flags are set by the data base administrator (see below). At the end of the day, the console operator stops the log process and the administrator resets the flags for storage of the backup data base. Note that the duration of this maintenance cycle depends on at least two considerations: the amount of time needed to store the data base periodically, and the amount of time required to recover the data base from the log file using DBRECOV if the system fails. The more often the data base backup copy is stored, the smaller the log file and recovery time will be. Regular backup of the data base is recommended, however a data base backup copy is not needed when using roll-back recovery. Refer to Appendix G for a brief overview of the disadvantages and benefits of logging to disc and logging to tape. This appendix includes sample job streams for the logging cycle.

### Starting the Logging Process

After a data base backup copy has been stored as described earlier in "LOGGING INSTALLATION", a logging process must be allocated to the log identifier so that it can be activated. A log process is an MPE system process responsible for buffering log records in memory. If the log file is on tape, the log process also buffers the log records on disc before writing them to the log file. The operator initiates this process from the console by using the command :LOG.

### Syntax

```
:LOG logid, {START  
              RESTART  
              STOP}
```

### Parameters

<i>logid</i>	is the name of the <i>logid</i> to be activated; the <i>logid</i> has been set into the data base root file previously.
START	initiates a log process for the first time.
RESTART	initiates a log process when appending new log records to an old log file.
STOP	terminates a log process. Termination does not take effect until all current users have closed the log file by calling the CLOSELOG intrinsic.

### Example

```
:LOG ORDERLOG, START
```

Note that if the log process is stopped using the :LOG command, but a backup data base copy is not generated at that time, the console operator should use the RESTART option in order to resume logging to the same log file.

To determine whether or not a log process is running, use the MPE command :SHOWLOGSTATUS to determine the log identifiers of active log processes. :SHOWLOGSTATUS will display the percentage of records in the log file if the *logid* output is to disc. This additional information may prove helpful in gaining a better idea when to perform a :CHANGELOG.

## Example

```
:SHOWLOGSTATUS
```

LOGID	CHANGE	AUTO	USERS	STATE	CUR-RECS	MAX-REC	%USED	CUR-SET
DUMMYLOG	NO	NO	4	INACTIVE	100	1000	10%	1
TAPELOG	YES		1	INACTIVE	5738			1
ORDERLOG	YES	YES	2	INACTIVE	500	1000	50%	2

## Changelog Capability

The MPE CHANGELOG feature provides a continuous MPE user logging process, with the ability to change log file tape or disc files when they reach capacity without stopping the user logging process. User logging will also keep track of the order of the files in the *log file set*. Parts of the changelog record contain the file set number (001-999) and device type of the file names in the record. In addition, there are records for the previous file in a set, first file in a set, and current file in a set. This format will allow recovery to always start at the beginning of the file set, or at any point within the file set if the sequence number is used, and reopen the log files on the same device type that they were created. The user issuing the :CHANGELOG command must be the creator of the *logid*. If the user issuing a :CHANGELOG is not the creator of the *logid* either LG or OP capability is required. If the mirror data base method (DBRECOV STOP/RESTART) is being used, CHANGELOG makes logging without interruption on the primary system possible.

## Syntax

```
:CHANGELOG logid[;DEV=device]
```

## Parameters

*logid* is the name of the currently active logging process.

*device* is the device name of the new log file (DISC, TAPE, SDISC, CTAPE). If the device specified is DISC, the file will be created in the *logid* creators logon group and account.

## Example

```
:CHANGELOG ORDERLOG; DEV=DISC
```

Note that the *logid* specified must be that of the currently active logging process. If the *logfile* is changed using :ALTLOG no linkage of the log file set is provided. CHANGELOG can only be performed on a *logid* set up with the :GETLOG command. The CHANGELOG command will terminate if the logging process state is RECOVERING, STOP, INITIALIZING, or CHANGELOG is already pending. The following message will be displayed on \$STDLIST:

```
INVALID STATE OF PROCESS
```

After issuing the :CHANGELOG command, if the *logid* is valid, changelog records are posted to the current log file. The current log file is closed and the new log file is opened. A message similar to the following message is displayed on the \$STDLIST and the console to confirm the change:

```
Log file for logid ORDERLOG has been changed from ORDER001 to ORDER002
```

If the new log file is a serial file the following message will appear on the console requesting the mounting of a new log file, in this case the *logid* is ORDERLOG:

```
Mount new {tape/cartridge tape/serial disc} for logid ORDERLOG
```

If a :LISTLOG command is executed while the logging process is performing a CHANGELOG, the file name displayed will be that of the current log file. The log file name will not be updated until the CHANGELOG sequence successfully completes. The :SHOWLOGSTATUS command may be used to display the current status of a logging process to determine if a CHANGELOG is taking place.

The following example shows the display of :LISTLOG. A CHANGELOG is currently taking place on log file ORDER001, since the CHANGELOG to ORDER002 has not yet successfully completed, ORDER001 is displayed:

## Example

```
:LISTLOG
```

LOGID	CREATOR	CHANGE	AUTO	CURRENT LOG FILE
DUMMYLOG	DATA.SYS	NO	NO	DUMMY.PUB.SYS
TAPELOG	DATA.SYS	YES		TAPE001
ORDERLOG	TST.MKTG	YES	YES	ORDER001.MKTG.SYS

## Setting Data Base Enable/Disable Flags

The data base administrator now can allow users to modify the data base by running DBUTIL and enabling data base for access. The administrator should also disable recovery at this time. This provides a safeguard against unintended recovery if DBRECOV is executed from a stream file against several data bases simultaneously.

### Example

```
:RUN DBUTIL.PUB.SYS
>>ENABLE ORDERS FOR ACCESS
  Access is Enabled
>>DISABLE ORDERS FOR RECOVERY
  Recovery is Disabled
```

## Ending the Logging Maintenance Cycle

At the end of the specified maintenance cycle (e.g., the end of the day) the above maintenance steps are reversed; that is, the console operator stops the logging process, and the data base administrator disables access, enables recovery, and stores a backup data base copy (required for roll-forward recovery).

### Example

```
:LOG ORDERLOG,STOP
:RUN DBUTIL.PUB.SYS
>>DISABLE ORDERS FOR ACCESS
  Access is Disabled
>>ENABLE ORDERS FOR RECOVERY
  Recovery is Enabled
>>EXIT
END OF PROGRAM

:RUN DBSTORE.PUB.SYS
WHICH DATA BASE? ORDERS
DATA BASE STORED
END OF PROGRAM
```



## Notes on Logging

- **LOG RECORDS.** The result of installing logging as described above is that all data base modifications (DBPUT, DBUPDATE, DBDELETE) are logged, and in modes 1 through 4 calls to DBOPEN, DBCLOSE, DBBEGIN, DBEND, and DBMEMO are logged to the log file. Each DBBEGIN and DBEND cause a log record to be written to the log file which includes such information as time, date, and user buffer. These log records are used by DBRECOV to identify logical transactions. All TurboIMAGE log records are contained within MPE WRITELOG records.

DBOPEN log records contain a time stamp in the data base root file, indicating the date and the time of the last DBSTORE (this time stamp is referenced by DBRECOV roll-forward recovery). DBOPEN log records also include the user identifier, log identifier, and the name, group, and account of the user, data base, and program.

DBUPDATE log records include both the new and the old data (before and after images); DBDELETE includes a copy of the deleted data (before image); DBPUT includes the record being added (after image).

- **LOG FILE TIME STAMPS.** There are two different log file time stamps; the DBSTORE time stamp set at the time the last data base backup copy was made (used by roll-forward recovery), and the roll-back time stamp created at the time the first DBOPEN is executed against the data base. The DBSTORE time stamp is fixed and does not change once the data base backup copy has been made. The roll-back time stamp is updated to the real time of the first DBOPEN following each close of the data base, providing a roll-back termination point should a roll-back recovery be required.
- **LOGGING TO TAPE OR DISC.** It is the choice of the data base administrator whether to log to tape or disc. The overhead required by the logging operation is comparable on disc or tape. However there are other factors that should be considered. Logging to tape is the more secure option, since a log file residing on tape is less susceptible to damage from possible system failure than a disc log file. (Refer to Appendix G for considerations when logging to disc and tape.)

In terms of allocating resources, logging to tape requires that the system be able to make a tape drive available as long as the data base is accessible for modification. If the decision is made to log to disc, you must use the MPE command :BUILD to create a new file and allocate space on disc. This allocation must be generous enough to avoid any possibility of filling the log file to capacity, as described earlier in "Building A Logfile For Logging To Disc".

- **DISABLING LOGGING.** While the transaction logging and recovery system is being used, logging is constantly enabled. However, in the event that logging is disabled and it is to be re-enabled, storing the data base with DBSTORE before re-enabling logging is recommended. This ensures that the DBSTORE flag and time stamp set when logging was first enabled are not reset when logging is re-enabled. This applies to roll-forward recovery only.
- **ERASING THE DATA BASE.** The execution of utility programs is not logged. If DBUTIL is used to erase the data base, the >>ERASE command automatically disables logging, ILR and roll-back recovery. Therefore, if the data base is erased, store the erased data base with DBSTORE before enabling the data base for logging once again.

## ROLL-BACK RECOVERY

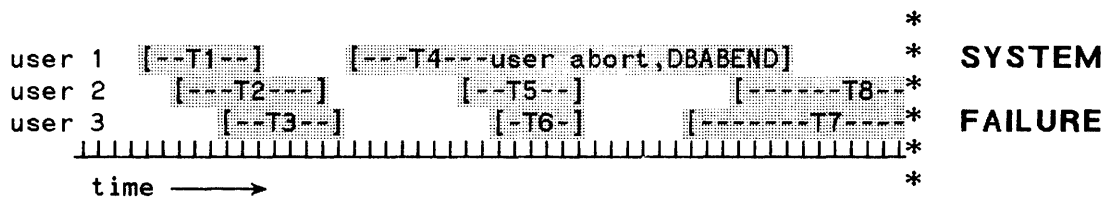
Roll-back recovery provides rapid recovery of data base data integrity following a "soft" system crash (e.g., system failure or loss of working memory). The roll-back feature is invoked through the DBRECOV utility and requires only the current data base log files in order to restore data integrity. A data base backup copy is not required for roll-back recovery. Regular backup of the data base is recommended, however, and is always required for roll-forward recovery in the event of a more serious problem (e.g., a disc head crash or problems occurred while roll-back recovery was in progress).

When invoked, the roll-back recovery feature will "roll-back", or undo, any incomplete data base transaction as shown in the log file following a soft system crash. Intrinsic Level Recovery (ILR) must also be enabled when using the roll-back feature to prevent the possibility of broken chains within the data structure.

In the event of a system failure, any multiple-intrinsic data modifying transaction that was incomplete could cause the data base contents to be logically inconsistent.

With roll-back enabled prior to the time of system failure, a record of each user transaction, in the sequence of occurrence, is available to determine which transactions were incomplete at the time of failure. Following a system failure all incomplete transactions as shown in the log file must then be undone, or rolled-back.

The following diagram illustrates the transactions of three different users at the time of a system failure:



In the above illustration the first user has completed one transaction (T1) and aborted another (T4) prior to the system failure. Both users two and three have completed two transactions each and each has one incomplete transaction at the time of failure. Individual data base transactions T1, T2, T3, T5, and T6 were completed and are properly reflected in the data base following system failure. Transactions T7 and T8, however, were incomplete at the time of system failure, causing an incomplete modification of data to be reflected in the data base. These incomplete transactions (T7 and T8) will then be rolled-back to their beginning, returning all affected data in the data base to their state before T7 and T8 began execution.

When transaction T4 is aborted, TurboIMAGE completes the transaction by issuing an abnormal end (DBABEND). This transaction is then seen as completed by the roll-back feature and is not normally rolled-back. If the aborted transaction is also to be rolled-back, the following DBRECOV command string must be issued before issuing the DBRECOV command >RUN (refer to Section 8, "DBRECOV"):

```
>CONTROL NOABORTS
```

The above command string causes the aborted transaction to be treated as an incomplete transaction during transaction roll-back. When >CONTROL NOABORTS is issued, TurboIMAGE only rolls-back aborted transactions which occurred during the last recovery block before the system failure. Refer to Section 8, DBRECOV >CONTROL, "RECORD NUMBERS AND TABLE OVERFLOW" for considerations when using >CONTROL command.

## Intrinsic Level Recovery (ILR) Requirements

To ensure correctness of the data base physical data links, ILR must be enabled when using the roll-back feature. With ILR enabled when a system failure occurs TurboIMAGE will automatically reconstitute chains. Without ILR enabled, a system failure may cause a loss of physical integrity and, as the roll-back feature does not repair broken chains, recovery of the data base would then be difficult. Intrinsic Level Recovery is enabled automatically when roll-back is enabled, but ILR must be manually disabled when disabling roll-back. Refer to "Using ILR", covered earlier in this section. Since roll-back recovery enables ILR, output deferred mode may not currently be enabled on the data base. If the data base is enabled for AUTODEFER (output deferred mode), the following message will be printed after the user attempts to enable ROLLBACK:

AUTODEFER MUST BE DISABLED BEFORE ILR CAN BE ENABLED

(Refer to DBUTIL >>ENABLE in Section 8 for more information).

## Enabling the Roll-Back Feature

To enable the roll-back feature complete the following sequence:

1. Set the logid and build a log file (if logging to disc) as shown in "Logging Installation", steps 1 through 5, earlier in this section.
2. Enable the roll-back feature for each particular data base by entering the DBUTIL command string:

>>ENABLE *dbname*[/*maintenance word*] FOR ROLLBACK.

3. Start the logging process and enable user access to the appropriate data bases as shown in "Maintaining Logging", earlier in this section. Once the logid has been set and log file built, DBUTIL will automatically enable logging and ILR when roll-back is enabled. Note that logging and ILR must be disabled manually.

If the logid was not set and/or the log file not built before issuing the >>ENABLE command for roll-back, a WARNING message that the logid is non-existent will be printed on the screen.

When roll-back is enabled, DBUTIL sets a roll-back flag to indicate that roll-back is enabled for the data base. DBUTIL also reserves six words in the root file for the roll-back time stamp (three words for the previous time-stamp and three words for the current time stamp). The roll-back time stamp is updated and logged in the log file and in the root file when the data base is first opened. Roll-back recovery then uses the time stamp during recovery to verify the correct log file for each data base.

### CAUTION

In the event of a system failure, do not restart logging before running the recovery system. Log records may have been lost due to the system failure. If logging is resumed without a recovery, the resulting discontinuous log file would cause invalid results in the event of a subsequent recovery.

## Disabling the Roll-Back Feature

In order to disable the roll-back feature, each involved data base must be in a quiet state with no user accesses in progress. Disable the roll-back feature by entering the DBUTIL command string:

```
>>DISABLE dbname FOR ROLLBACK
```

When roll-back is disabled DBUTIL resets the roll-back flag and roll-back time stamp.

### CAUTION

DO NOT DISABLE ROLLBACK IF ROLLBACK RECOVERY MUST BE USED LATER. This action will reset the logging time stamp. Once the logging time stamp is reset (when roll-back is disabled). Roll-back recovery cannot be performed with the current log file on the named data base. The data in the data base is considered correct and therefore cannot be rolled-back.

When the roll-back >>DISABLE command is issued, DBUTIL prompts a warning to remind you that the time stamp will be erased and prompts for a response as follows:

```
WARNING: ROLLBACK time stamp will be erased.
Please type Y to confirm your disable command>>
```

If you enter Y, DBUTIL will then continue the roll-back disable process. If Y is not entered the >>DISABLE command is not performed.

## Performing Roll-Back

To complete the transaction roll-back process following a system failure, perform the following steps:

1. Following a restart to bring-up the operating system, locate the applicable log file media to be used for transaction roll-back. If logging to tape, the correct tape will need to be applied. If using the CHANGELOG feature of MPE and there are multiple log file tapes, this will be the first tape of the series. If logging to disc, TurboIMAGE will automatically locate the applicable log file by checking the beginning of the root file for the logid.
2. Backup of the data base is recommended, in case a system failure occurs during the recovery process.
3. Enter the MPE command string:

```
:RUN DBRECOV.PUB.SYS
```

4. Enter the following DBRECOV command ( where *dbname* is the name of individual data bases to be rolled-back):

```
>ROLLBACK dbname [,dbname2,...,dbnameN]
```

5. Enter all other desired DBRECOV commands (>FILE, >CONTROL, and >PRINT). Note that the >FILE command optional parameter *rmode* is not used with the roll-back feature. (Refer to Section 8 for more information.)
6. Enter the DBRECOV command:

>RUN

Following entry of the >RUN command, DBRECOV will ask you to mount the log tape (if the log file media is tape). Continue the roll-back process as directed by messages returned to both the console and the terminal screen. If using the :CHANGELOG command or :GETLOG AUTO option the following message will be returned to both console and terminal screen:

Reply CONTINUE on console when *logfile* is ready

The response CON would be given at the console.

### RUN COMMAND

After the >RUN command is given, the DBRECOV program recovers the specified data bases, creates specified user recovery files, and terminates. The DBRECOV program could be terminated alternatively without any recovery taking place with an >EXIT command.

For recovery to succeed, the person running DBRECOV (usually the data base administrator) must have access to the log file. This implies having system manager capability or being the creator of the log identifier with read access to the log file if it resides on disc in a different logon group and account. If the log file is on tape, the user must be able to provide the volume identifier to the operator mounting the tape.

### Other DBRECOV Commands

Other DBRECOV commands available include:

- >CONTROL
- >FILE
- >PRINT

**CONTROL COMMAND**

The >CONTROL command is used to specify the conditions for recovery. If the >CONTROL command is not issued, the following conditions must be met for recovery to succeed:

- The data base time stamp must correspond with the time stamp in each DBOPEN log file record.
- No errors are allowed in job (batch) execution.
- Transactions which are incomplete due to program aborts are rolled-out.

The >CONTROL command can be used to override these conditions. Each override option can be negated by specifying its default option, and vice versa:

OPTION	DEFAULT OPTION
NOABORTS	ABORTS
MODE4	MODEX
STATS	NOSTATS
ERRORS=nnnn	ERRORS=0
EOF=nnnn	*EOF=nnnn

\*The initial default condition is that no end-of-file is imposed on recovery. Once a particular record number has been specified by EOF, it can be changed by specifying a new record number.

The following provides an example of the override:

```
>CONTROL MODE4
```

MODE4 allows users read access to the data base while recovery is in process.

For discussion of options and form of the >CONTROL command, refer to the >CONTROL command under the utility program DBRECOV.

Note that the >CONTROL command does not specify a data base. Therefore, all CONTROL options apply to all data bases being recovered.

## FILE COMMAND

The recovery file facility is an interface between the recovery system and the application program. With the >FILE command, DBRECOV sorts the log file by individual users and/or user identifiers, and designates an MPE file as the destination for the log records for each user.

The recovery file facility is based on the concept of transactions within transaction blocks. A transaction block consists of all transactions between a call to DBOPEN and DBCLOSE. Within each transaction block, a transaction is defined as either:

1. A single call to DBPUT, DBUPDATE, or DBDELETE if not preceded by a call to DBBEGIN, or
2. A sequence of calls beginning with a call to DBBEGIN, followed by any number of calls to DBPUT, DBUPDATE, or DBDELETE and ending with a call to DBEND.

For each transaction block, the >FILE command returns the initial DBOPEN log record to the user recovery file. The DBCLOSE record is returned as well, unless either:

1. Not all of the transactions within the block could be recovered, or
2. There was no DBCLOSE log record for this block on the log file. This happens when the system fails while the data base is open.

Consequently, an application can determine the outcome of recovery to some extent by examining the number of DBOPEN and DBCLOSE or pairs of DBBEGIN and DBEND log records returned to the user recovery file. If there are as many calls to DBCLOSE as to DBOPEN, it is likely that all transactions were successfully recovered. However, there is a possibility that an entire transaction block was lost due to the system failure if the block was very short. Fewer calls to DBCLOSE indicate the possibility that some transactions were lost and need to be re-entered. More information about recovery can be inferred from the recovery file by using the optional *fmode* parameter to return transactions to the user recovery files in addition to the intrinsic DBOPEN and DBCLOSE. *Fmode* refers to transactions that were rolled-out. Refer to the DBRECOV command >FILE for details of operation.

## PRINT COMMAND

The >PRINT command is an option used to display information before actually initiating recovery with the >RUN command. If DBTABLE is specified in the command, the names of data bases specified for recovery by >RECOVER commands are returned. If FILETABLE is specified, file references, user references, and *fmodes* specified by >FILE commands are returned. These tables, along with other statistics, are also printed when recovery is complete.

## ROLL-FORWARD RECOVERY

The roll-forward recovery system can be executed to bring data bases back to a likeness of their state at the time of a hard system failure (e.g., disc head crash). This requires that a backup copy of the data base has been stored and the log file is available. (Refer to "Logging Installation", earlier in this section, for roll-forward logging information.)

When executing roll-forward recovery following a hard system failure, the TurboIMAGE utility program DBRECOV recovers the data base physical and logical integrity by suppressing any incomplete transactions. A backup copy of the data base is updated with the completed transactions that were written to the log file.

Recovery of the data base requires restoring the backup copy and running the recovery system to re-execute the data base modifications from the log file. In addition, a transaction-oriented file facility can be used by the data base administrator to route log records back to individual user recovery files and to return information regarding the successful recovery or suppression of transactions. Following recovery, an application program can use these files to inform each user where to resume transactions. (Refer to the "DBRECOV >CONTROL" command in Section 8.)

Although the logging and recovery system is designed to successfully re-execute transactions that completed before the system failure, there is a possibility that some transactions will not be recovered. The possible causes of this situation include the following:

- One or more records could be lost in the log system buffers if the system fails before they are written to the log file.
- A transaction may have originally failed to complete due to the failure, and is therefore suppressed on purpose.
- A transaction may depend upon some data base modification that was suppressed. This condition indicates inadequate locking between processes.
- An incorrect data base was restored. Recovery will yield invalid and erroneous results or a record table overflow if this occurs.

If any transaction fails to be recovered, all subsequent calls within the same transaction block are suppressed as well. (For information about transaction blocks refer to "FILE Command", later in this section.)

<b>CAUTION</b>
----------------

In the event of a system failure, do not restart logging before running the recovery system. Log records may have been lost due to the system failure. If logging is resumed without a recovery, the resulting discontinuous log file would cause invalid results in the event of a subsequent recovery. The same is true for making modifications to the data base. The data base should be disabled for user access until recovery has completed. Follow the recommended steps when performing a DBSTORE. (Refer to "Making a Data Base Backup Copy".)



## Intrinsic Level Recovery (ILR) Requirements

When using roll-forward logging for the purpose of recovery following a system failure, it is not necessary to enable the Intrinsic Level Recovery (ILR) feature as roll-forward logging provides recovery of both intrinsics and transactions following a system failure.

If roll-forward logging is enabled and being used to provide a log file for audit purposes only, ILR should be enabled to ensure correctness of the data base physical data links. Without ILR enabled (when using roll-forward logging for audit only), a system failure may cause a loss of physical integrity within the log file due to broken chains. If logging is restarted before running roll-forward recovery on the data base, an inconsistent log file will result. ILR should not be used as a recovery method when the data base is enabled for roll-forward recovery.

## Enabling the Roll-Forward Feature

To enable the roll-forward feature complete the following sequence:

1. Set the logid and build a log file (if logging to disc) as shown in "Logging Installation", steps 1 through 5, earlier in this section.
2. Enable the roll-forward feature for each particular data base by entering the DBUTIL command string:

```
>>ENABLE dbname FOR LOGGING
```

3. Start the logging process and enable user access to the appropriate data bases as shown in "Maintaining Logging", earlier in this section. The Intrinsic Level Recovery (ILR) feature must be manually enabled at this point if desired. Use of ILR is recommended as it eliminates possible problems with broken chains within the data base. (Refer to "Using ILR", earlier in this section.) If used, ILR must be manually disabled.

## Restoring the Backup Data Base Copy

Before roll-forward recovery can begin, the data base administrator must restore the data base to the state at which logging was enabled. This is done by running the DBRESTOR program after purging the damaged data base or by using the MPE facility :RESTORE. Keep in mind that to use RESTORE all data bases and files must reside in the same group and account, and you must have account manager capability. You should ensure that recovery is enabled and access disabled to prevent user modifications before the recovery system executes (refer to the DBUTIL command >>SHOW). If the flags were set as recommended prior to storage of the backup copy, no changes will be needed.

Several data bases can log to the same log file simultaneously since each call to DBOPEN specifies the fully qualified name of the data base. If all data bases that logged to the same log file are to be recovered simultaneously, then all backup copies must be restored prior to running the recovery system. However, if the recovery system begins execution before a data base has been restored, accidental recovery is prevented if recovery has been disabled on the working data base, as specified earlier in "Maintaining Logging".

### Example

```
:RUN DBRESTOR.PUB.SYS
WHICH DATA BASE?  ORDERS
DATA BASE RESTORED
END OF PROGRAM
```

```
:RUN DBUTIL.PUB.SYS
>>SHOW ORDERS FLAGS
For Data Base ORDERS
Access is Disabled
Dumping is Disabled
Logging is Disabled
Recovery is Disabled
>>EXIT
END OF PROGRAM
```

### CORRESPONDENCE BETWEEN BACKUP COPY AND LOGFILE

The TurboIMAGE logging and recovery systems depend upon the exact correspondence between the stored backup data base copy and the working data base on disc at the time logging was initiated. The DBSTORE flag and timestamp, properly used, will enforce this condition. Therefore, it is strongly recommended that you always use DBSTORE to generate backup copies.

For flexibility, in the event that you might use :STORE or :SYSDUMP to store the backup, the capability exists to defeat the timestamp and DBSTORE flag mechanism, using the NOSTAMP and NOSTORE options of the >CONTROL command. In this case, you must assume responsibility for maintaining the correspondence between backup copy and the log file. Note that a data base recovered with the wrong log file causes DBRECOV to generate erroneous data in the data base and that this condition cannot always be detected. Modifications to the data base with logging disabled will also cause the the recovered data base to be incorrect. DBRECOV may also abort with a record table overflow due to the necessity of moving transactions from the old record number to a new record number when modifications have been made.

## Recovering Data Without a Backup Copy

If a data base structure is damaged, and no backup copies are available, it may be possible to salvage most or all of the data by serially reading the data entries, writing them to a tape or a disc file, recreating the data base, and reloading the data. If structure damage is detected by an abnormal termination of the DBUNLOAD program running in CHAINED mode, or by a discrepancy between the number of entries unloaded and the number expected from one or more data sets, it may be possible to unload the data base by running DBUNLOAD in SERIAL mode, which does not depend on internal linkages. These DBUNLOAD modes are discussed in the next section.

If all necessary existing manual master data entries are written to tape or serial disc, reloading the data base using the DBLOAD program, after erasing the data base using DBUTIL, results in a structurally intact approximation of the original data base.

## Performing Roll-Forward

To complete the transaction roll-forward process following a hard system failure, perform the following steps:

1. Following a restart to bring-up the operating system, locate the applicable log file media to be used for roll-forward recovery. If logging to tape, the correct tape will need to be applied. If using the changelog feature of MPE and there are multiple log file tapes, this will be the first tape in the series. If logging to disc, TurboIMAGE will automatically locate the applicable log file by checking the beginning of the root file for the logid.

2. Enter the following MPE command string:

```
:RUN DBRECOV.PUB.SYS
```

3. Enter the DBRECOV command below (where dbname is the name of the individual data bases to be recovered):

```
>RECOVER dbname [,dbname2,...,dbnameN]
```

4. Enter all other desired DBRECOV commands (>FILE, >CONTROL, and >PRINT). Of the available DBRECOV commands, only the >RECOVER and >RUN commands are necessary for recovering a data base.
5. Enter the DBRECOV command >RUN. Following the entry of the >RUN command, DBRECOV will ask you to mount the log tape (if the log file media is tape). Continue the roll-forward process as directed by messages returned to both the console and the terminal screen.
6. If using the :CHANGELOG command or :GETLOG AUTO option and logging file media is tape, the following message will appear on the terminal screen and the console:

```
Reply CONTINUE on console when logfile is ready
```

The response CON would be replied to at the console.

## Post-Recovery Procedures

After a recovery has been completed, the data base administrator and system manager have three procedural options. The option chosen determines the recovery procedure in the event of a second system failure. Together, the data base administrator and system manager or console operator should agree upon the best post-recovery procedure in order to avoid confusion at recovery time. The options available after recovery include:

1. The data base administrator stores a new backup data base copy, and the system manager or operator starts a new log file from the console. In the event of a subsequent system failure, the new backup data base is restored and recovered against the new log file. This option allows for a straightforward recovery procedure but delays users from accessing the data base until the new backup copy has been generated.
2. A new backup data base is not generated; the system manager or operator resumes transaction logging to the same log file using the **RESTART** option. In the event of a subsequent system failure, the old data base copy is restored and recovered against the log file.

This procedure is the same as the original recovery, but takes longer due to the additional log file records. Users can access the data base after the first system failure without waiting for it to be stored.

Do not restart a log file before the data base has been recovered after a system failure. Otherwise, since some log records could have been lost in the system failure, the log file may not be consistent with the true state of the data base. A recovery is necessary to bring the data base and log file into agreement before restarting the log process.

3. A new backup data base is not generated; the system manager or operator initiates logging to a new log file. In the event of a system failure, the old data base copy is restored and two recoveries are executed: the first against the old log file, the second against the new log file.

Until a new data base backup copy is generated, if the system manager or operator consistently starts logging to a new log file after a system failure, a total recovery preceded by  $n$  failures requires  $n$  executions of the recovery system.

Note that the second and subsequent recoveries of a data base against more than one log file will be refused unless the **DBSTORE** flag is disabled. This is because the first modification re-executed from the first log file clears the **DBSTORE** flag from the data base rootfile. Subsequent calls to **DBRECOV** can only succeed by specifying the **>CONTROL NOSTORE** option. Furthermore, the data base administrator must ensure that the log files are recovered in the proper order.

This procedure is not recommended if option #2, above, is available.

### RECOVER COMMAND

The >RECOVER command designates the name of a data base to be recovered. If more than one data base has logged to the same log file, they can be recovered concurrently by typing the data base names separated by a comma.

If the data base copy was stored with a procedure other than DBSTORE (for example, MPE RESTORE), the DBSTORE flag will not have been set in the data base root file. If you are sure you have restored the correct, unmodified version of the data base, and wish to use it for recovery, the >CONTROL NOSTORE option must be entered before the >RECOVER command can succeed (refer to >CONTROL).

Other conditions necessary for success of the >RECOVER command include:

- The data base must be accessible to you from your logon group and account.
- The log identifier must not have been altered since the log file was generated (see Setting Log Identifier and Flags In Data Base).
- The data base must be enabled for recovery.
- All data bases specified for recovery must contain the same log identifier.
- You must be the creator of the log identifier, unless you have system manager capability.
- No other users are accessing the data base.

The last condition, above, can be overridden if users are given read access during recovery (refer to the >CONTROL MODE4 command in "DBRECOV").

If the >RECOVER command succeeds, recovery can be initiated by typing the >RUN command.

### RUN COMMAND

After the >RUN command is given, the DBRECOV program recovers the specified data bases, creates specified user recovery files, and terminates. The DBRECOV program could be terminated alternatively without any recovery taking place with an >EXIT command.

For recovery to succeed, the person running DBRECOV (usually the data base administrator) must have access to the log file. This implies having system manager capability or being the creator of the log identifier with read access to the log file if it resides on disc in a different logon group and account. If the log file is on tape, the user must be able to provide the volume identifier to the operator mounting the tape.

## Other DBRECOV Commands

Other DBRECOV commands available include:

>CONTROL

>FILE

>PRINT

### CONTROL COMMAND

The >CONTROL command is used to specify the conditions for recovery. If the >CONTROL command is not issued, the following conditions must be met for recovery to succeed:

- The data base time stamp must correspond with the time stamp in each DBOPEN log file record.
- The DBSTORE flag must be set in the data base root file.
- No errors are allowed in job (batch) execution.
- Transactions which are incomplete due to program aborts are recovered.

The >CONTROL command can be used to override these conditions. Each override option can be negated by specifying its default option, and vice versa:

OPTION	DEFAULT OPTION
NOSTAMP	STAMP
NOSTORE	STORE
NOABORTS	ABORTS
MODE4	MODEX
STATS	NOSTATS
ERRORS=nnnn	ERRORS=0
STOPTIME=dateX timeX	*STOPTIME=dateY timeY
EOF=pppp	*EOF=qqqq

\* The initial default condition is that no stoptime or end-of-file is imposed on recovery. Once a particular date or record number has been specified by STOPTIME or EOF, it can be changed by specifying a new date or record number.

The following provides an example of the override:

>CONTROL NOSTAMP,STAMP

Since STAMP was entered after NOSTAMP, STAMP negates NOSTAMP, so that recovery proceeds with the timestamp check intact.

For discussion of options and form of the >CONTROL command, refer to the >CONTROL command under the utility program DBRECOV. Note that the >CONTROL command does not specify a data base. Therefore, all CONTROL options apply to all data bases being recovered.

## FILE COMMAND

The recovery file facility is an interface between the recovery system and the application program. With the >FILE command, you sort the log file by individual users and/or user identifiers, and designate an MPE file as the destination for the log records for each user.

The recovery file facility is based on the concept of transactions within transaction blocks. A transaction block consists of all transactions between a call to DBOPEN and DBCLOSE. Within each transaction block, a transaction is defined as either:

1. A single call to DBPUT, DBUPDATE, or DBDELETE if not preceded by a call to DBBEGIN, or
2. A sequence of calls beginning with a call to DBBEGIN, followed by any number of calls to DBPUT, DBUPDATE, or DBDELETE and ending with a call to DBEND.

For each transaction block, the >FILE command returns the initial DBOPEN log record to the user recovery file. The DBCLOSE record is returned as well, unless either:

1. All of the transactions within the block could not be recovered (refer to the first page of Roll-Forward Recovery), or
2. There was no DBCLOSE log record for this block on the log file. This happens when the system fails while the data base is open.

Consequently, an application can determine the outcome of recovery to some extent by examining the number of DBOPEN and DBCLOSE or pairs of DBBEGIN and DBEND log records returned to the user recovery file. If there are as many calls to DBCLOSE as to DBOPEN, it is likely that all transactions were successfully recovered. However, there is a possibility that an entire transaction block was lost due to the system failure if the block was very short. Fewer calls to DBCLOSE indicate the possibility that some transactions were lost and need to be re-entered. More information about recovery can be inferred from the recovery file by using the optional *rmode* and *fmode* parameters. These parameters return transaction information to the user recovery files in addition to the intrinsic DBOPEN and DBCLOSE. *Rmode* and *fmode* refer respectively to transactions that succeeded and failed to be recovered. Refer to the DBRECOV command >FILE for details of operation.

## PRINT COMMAND

The >PRINT command is an option used to display information before actually initiating recovery with the >RUN command. If DBTABLE is specified in the command, the names of data bases specified for recovery by >RECOVER commands are returned. If FILETABLE is specified, file references, user references, *fmodes* and *rmodes* specified by >FILE commands are returned. These tables, along with other statistics, are also printed when recovery is complete.

## THE MIRROR DATA BASE

Transaction logging and regular backups are good maintenance. However, if data bases must be accessible at all times, and cannot be down, even for maintenance, then a new maintenance method is needed. A system can be set up for constant access or "high availability", and still have controlled maintenance.

The mirror data base is the fundamental element in creating a high availability data base system. This system consists of two identical data bases on two separate computer systems. One data base is housed on a primary system and is constantly accessible to users and application programs. The other "mirror" data base resides on the secondary system and is used for maintenance.

To establish a mirror data base, the following requirements are necessary:

- Two identical copies of the data base(s) is needed, one copy on the primary system, one on the secondary system.
- All transactions on the primary system must be logged to a permanent file.
- Move (or copy) the file containing the transactions to the secondary system, and update the data base(s) on the secondary system using the transactions files.

Once the secondary system is established, it can be used to make backups of the data base. The primary system never has to be brought down for maintenance.



## DBRECOV STOP-RESTART FEATURE

MPE CHANGELOG and GETLOG AUTO option make logging without interruption on the primary system possible, thus increasing the availability of the data bases.

After the log files are transferred to the secondary system of the mirror data base system, they are applied to the mirror data bases using the DBRECOV roll-forward recovery process. DBRECOV has been modified to make the mirror data base a workable maintenance method. The STOP-RESTART feature of DBRECOV adds the capability to CONTINUE or STOP the recovery process on the secondary system if DBRECOV cannot find the next log file in the log set. This STOP-RESTART feature is the key to the mirror data base system. Whenever DBRECOV cannot find the next log file in a log set, the recovery process on the secondary system can be stopped, the data bases can be backed up, and then recovery can be restarted from the point it was stopped. The primary system never has to be brought down for backups.

DBRECOV applies the chained log files starting with the first log file created when logging was enabled. It continues to process each log file in the log set consecutively until it cannot find the next log file in the set. It then prompts the user to CONTINUE or STOP the recovery process.

If the reply is CONTINUE, DBRECOV will keep searching for the next log file. When the next log file is found DBRECOV resumes roll-forward recovery on the mirror data base. The CONTINUE or STOP prompt will appear as long as DBRECOV cannot find the next log file in the log set. DBRECOV is stopped if the reply STOP is entered and a RESTART file containing all the necessary information to restart recovery is created.

Once the DBRECOV process is stopped, backup of the data base in a consistent state can be done and limited data base maintenance on the secondary system can be performed. Some DBUTIL functions can not be performed while the DBRECOV process is stopped. If the data base is in RESTART mode then the following DBUTIL processes cannot be performed:

- Access is not allowed in order to keep the data base logically consistent.
- Resetting the maintenance word is not allowed. If the maintenance word were reset then RESTART would be impossible.
- Purging or erasing the data base is not allowed. If either of these options were used in DBUTIL then the recovery process would be invalidated. (The user must run DBRECOV,ABORT or DBRECOV,PURGE before purging or erasing the data base.)

DBRECOV,RESTART will restart the roll-forward recovery process from the point it was stopped. DBRECOV uses the information in the RESTART file to restart recovery. DBRECOV will continue until, once again, it cannot find the next log file in the log set. The prompt to CONTINUE or STOP will be displayed and backup of the data base can again be done.

If RESTART recovery from the current STOP point cannot be done, DBRECOV,ABORT can be used. Recovery can no longer be restarted from the same point that it was stopped once ABORTed because the RESTART file is purged. The data base flags are returned to the same settings as before the recovery process was started.

If ABORT fails to abort recovery because of an inconsistent RESTART file, DBRECOV,PURGE can be used to delete the current RESTART file before beginning the mirror data base process again.

## Notes on Logging

Backups on the secondary system are made more efficient by controlling the logging processes on the primary system. There are some important factors to consider before enabling logging on the primary system.

- Logging to tape eliminates the step of storing log files with the data bases once they are rolled forward on the secondary system. However, keeping track of which log file tapes go with which data base and RESTART file backup tapes is required. Logging to tape requires a dedicated tape drive.
- Logging to disc enables storing log files and the data bases on a single tape using an MPE STORE rather than a DBSTORE command. When logging to disc, the user must remember to backup all log files that were processed after the last DBRECOV,RESTART along with the data bases and RESTART file. Private volume may also be used and may be faster since you can transfer to last log file without waiting for the primary system to be warmstarted.
- Naming conventions make storing the data bases much easier and eliminate the use of several different tapes for the log files. If naming conventions are followed (refer to "Maintaining Logging") an MPE STORE using the "@" can be used to store the log files, data base and RESTART file.
- You can either let the :GETLOG AUTO option switch to the next log file automatically and/or manually issue a :CHANGELOG command to close the current log file and open the next file in the log set.
- When using the STOP-RESTART option, the log file name and the logid must be different.

The data base administrator must determine how big to make the log files, based on how far behind the secondary system will be, and how often backups will be done. To keep the secondary system as close to a mirror image of the primary data base as possible, log files should be made small so that they will be filled quickly and can be sent to the secondary system frequently. Of course, making the log files small means spending more time transferring log files from the primary to the secondary system.

However, there is a disadvantage to having several small log files in the application of STOP-RESTART. DBRECOV will only prompt to CONTINUE or STOP recovery if it is between log files in a log set, and it cannot find the next log file. Therefore, if there are several small log files, the prompts to CONTINUE or STOP are more frequent.

Another logging option would be to set the log file size very large and just manually change to the next log file by issuing the :CHANGELOG command. The idea is to continually fill the log file with transactions, and when the user is ready to copy the log file over to the secondary system, change to the next log file and copy the current one over. This method requires someone at the system console to monitor the logging and data base maintenance processes. If the user wants to schedule backups on the secondary system around certain times of the day, say at the beginning and end of a work day, use this logging procedure on the mirror data base. The user can log a full shift's transactions and then manually issue a :CHANGELOG at the system console to create a new log file in the log set. (If the :GETLOG AUTO option was specified when logging was enabled, a manual :CHANGELOG command can also be issued at any time.) Then the closed log file is transferred to the secondary system and the DBRECOV roll-forward recovery process continues on the secondary data bases.

Once the log file has been processed, it will look for the next log file in the log set. If it is still logging to that next log file on the primary system, the user is prompted by DBRECOV to CONTINUE or STOP. At this point, recovery can be stopped and the secondary data base can be stored and await the arrival of the next log file at the end of the shift. Remember to store the RESTART file and the current, unprocessed log files with the data bases.

## Transferring Log Files

The :GETLOG AUTO option and :CHANGELOG provide the capability to schedule secondary system backups through various methods of logging. The data base administrator can control the way the log files are transferred from the primary system to the secondary system. The method chosen should depend on the maintenance needs. Four ways of copying log files from the primary to the secondary system follow:

1. Copying files over a direct DSLINE from the primary to the secondary system.
2. Logging to a serial disc and physically transporting the disc to the secondary system.
3. Logging to (private or system) disc and copying disc to tape and transporting the tape over to the secondary system.
4. Logging directly to tape and mounting the tape on the secondary system.

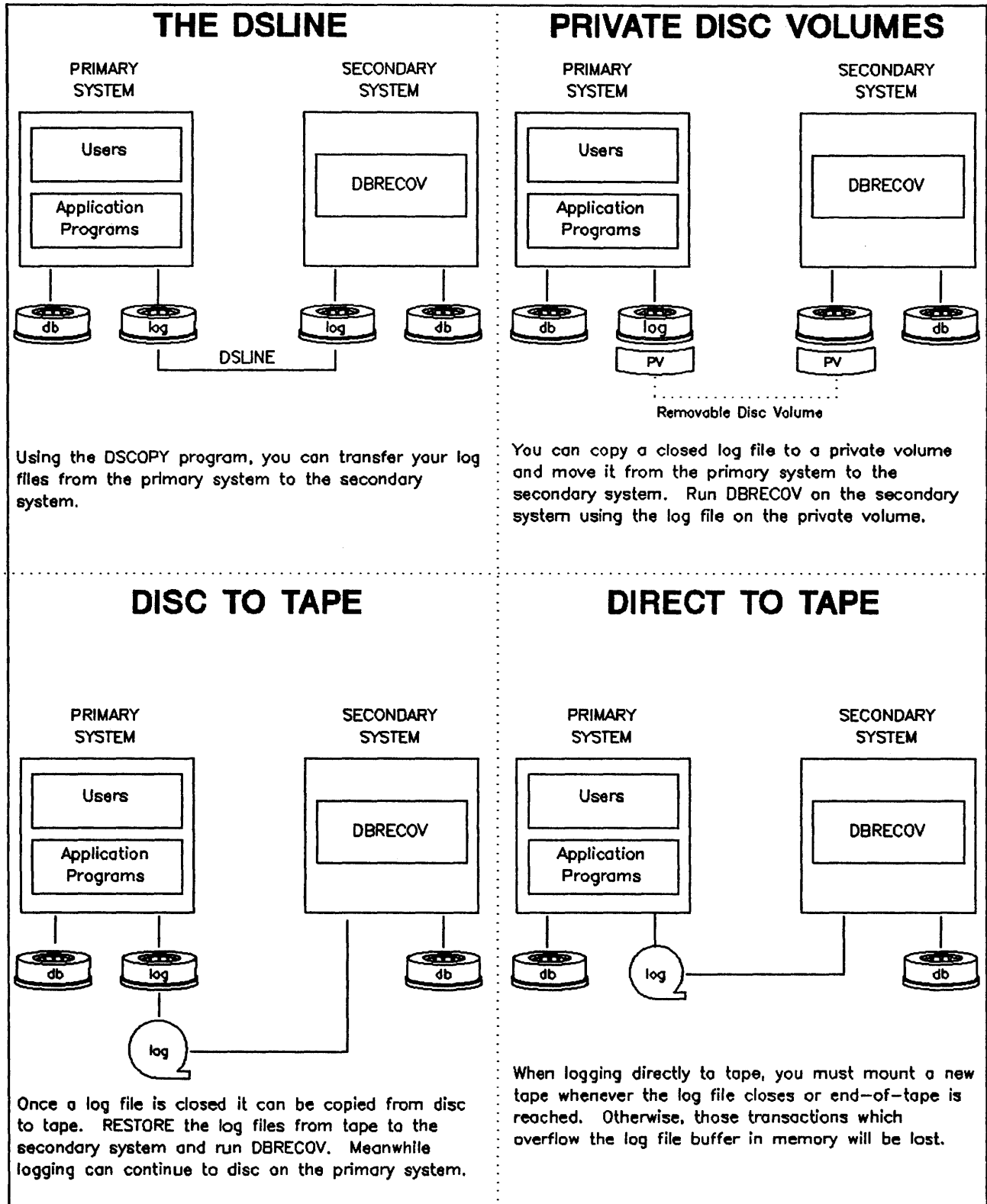


Figure 7-4. Transferring Log Files to the Secondary System

## Performing DBRECOV STOP-RESTART

After the mirror data base system is set up the DBRECOV STOP-RESTART feature is used to maintain the secondary data base. To start the initial DBRECOV procedure, the user must make sure logging is enabled on the primary system and that either the MPE GETLOG AUTO option or CHANGELOG is being used. For more information on logging options refer to "Installing Logging" and "Maintaining Logging" earlier in this section. Appendix G provides a brief outline of logging to disc and logging to tape.

### STOPPING DBRECOV

DBRECOV will roll forward all log files in the log set on the secondary system, one at a time. When DBRECOV cannot find the next log file in a log set, it will print this message on the console:

```
DBRECOV - Reply CON or STOP when filexxx is ready.
```

A message for the user is displayed in the \$STDLIST file:

```
UNABLE TO OPEN LOG FILE filexxx  
REPLY 'CONTINUE' OR 'STOP' ON CONSOLE.
```

The filexxx is the log file that DBRECOV is trying to find. If that log file has been closed on the primary system and is ready to be moved over to the secondary system, transfer it to the secondary system and reply CON or CONTINUE on the console. DBRECOV will look for filexxx again. The roll-forward process will continue as long as the next log file has been copied over correctly and is available to DBRECOV.

The next log file may not be ready yet. For example, the primary system might still be logging transactions, the log file might have been renamed, or it might be on a tape that was not mounted. This provides an opportunity to STOP recovery and perform maintenance on the data base. (Refer to "Storing The Data Bases" on the next page.) To stop recovery simply reply STOP at the console. A list of the data bases involved in recovery are displayed in the \$STDLIST file. At this point, DBRECOV creates a RESTART file containing all the necessary information to continue the recovery process when the RESTART option is requested.

```
DATA BASE(S) WITH RECOVERY SUSPENDED:
```

```
    base1.group.acct  
    base2.group.acct
```

```
    .  
    .
```

This is a list of the data bases that are in the RESTART file. These data base names are specified later on when either the RESTART or ABORT options are used. The RESTART file name is the same as the logid name entered when logging was enabled on the primary system.

DBRECOV will then print the name of the log file it will need to restart recovery, the record number at which the quiet block begins in the current log file, the number of records currently in the staging file, and the actual file name of the RESTART file for that recovery process:

```
RESTART RECOVERY WITH LOG FILE: filexxx
QUIET BLOCK BEGINS AT RECORD recordnumber
NUMBER OF RECORDS IN STAGING FILE numrecs
RESTART FILE NAME: filename
```

The user is returned to MPE. DBUTIL >>SHOW *data base name* FLAGS displays the recovery state, whether the data base in recovery has been set for RESTART.

When running multiple recovery processes from the same log file the user needs to equate the logid, which is the formal file designator for the RESTART file, to a unique file name for each recovery process. The new file name is the RESTART FILE NAME for that specific recovery process.

## STORING THE DATA BASES

The data bases can be backed up at this time. It is important to store all files involved in recovery since the last successful RESTART. In other words the data base administrator should store the data bases, the current RESTART file, and all log files that were processed since the last successful DBRECOV,RESTART. If the RESTART file is not stored with the data base backups, it will be modified once recovery is restarted. Without the previous RESTART file or the log files, the backup data bases cannot be used to RESTART recovery in case the current RESTART fails.

The RESTART file and the data bases have time stamps inside them which tell DBRECOV which RESTART file goes with which data bases. Once DBRECOV is restarted, the time stamp in the RESTART file is changed. If the RESTART file is not stored, the time stamps will not match, and the RESTART will not be successful.

The method used to store the data bases, RESTART file, and log files depends on the medium that the user is logging to.

If logging to tape, the log files are already stored on a transportable medium and backing them up is not necessary. However, the user must keep the log files grouped with the data base and RESTART file backups. If recovery must be restarted from a backup, the user will need to restore the tapes containing:

- the data bases
- the RESTART file
- all log files processed since the last successful RESTART

## Maintaining The Data Base

If the user does not keep track of which log files go with which data bases, the RESTART of recovery from the backups is not possible.

If logging to disc, remember to store the log files that were rolled forward since the last successful restart along with the RESTART file and the data bases. Logging to disc makes it easier to keep the log files grouped with the data bases and RESTART file because all the log files can be stored at the same time when recovery is stopped. Use an MPE STORE with the "@" option (rather than a DBSTORE) to backup all the files on a minimum number of tapes. If it is necessary to restart from a backup all the necessary files will be together.

Using naming conventions makes storing the files to tape much easier. The logging naming conventions should be used. For example, if the data base is ORDERS, name the logid ORDERRS (RS for RESTART), and the log file ORDER001. The user can MPE STORE all the files with one command: ":STORE ORDER@".

### NOTE

To avoid incompatible time stamps it is important to store the RESTART file at the same time that the data bases are stored. If logging to disc, also make sure to store all log files processed since the last successful restart.

## RESTARTING DBRECOV

To RESTART the recovery process after the next log file in the set is transferred, or the data base maintenance is completed, type the run command:

```
:RUN DBRECOV.PUB.SYS,RESTART
```

DBRECOV will request the name of one of the data bases in the RESTART file:

WHICH DATA BASE?

If the user types in the name of a nonexistent data base, another prompt for the data base in the RESTART file will appear. Once again enter the name of a data base in the RESTART file. From the data base name that is entered, DBRECOV determines the name of the RESTART file, tries to open it, and restart the recovery process. If the RESTART file is successfully opened, but is not a RESTART file the following error message is printed:

filename is not a DBRECOV RESTART file.

and returns the user to the MPE prompt. This error usually occurs when another file with the same name as the RESTART file has been created on the system. Make sure the file is a RESTART file, and try the RESTART again.

If the RESTART file cannot be located, go back to the previous tape, restore the data bases, which should have their own RESTART file and log files stored with them, and run DBRECOV,RESTART from that point. The log files between the previous and the current STOP point will be reprocessed and the roll-forward process will continue with the current log file.

When the correct RESTART file is opened, DBRECOV will look at the file to make sure that the version numbers are compatible with the version of DBRECOV being run. If the version numbers do not match, DBRECOV will print the error message:

RESTART FILE NOT COMPATIBLE WITH THIS VERSION OF DBRECOV

and the user is returned to the MPE prompt. This message means that another version of DBRECOV is running other than the version which created the current RESTART file. Install the correct version of TurboIMAGE and run DBRECOV,RESTART again.

If the user logon is not the same as the logon when DBRECOV was suspended, the following message is printed:

must be logged on as same user and account where DBRECOV was suspended.

Log on as the same user and account that was used when DBRECOV was originally suspended and run DBRECOV again.

Once the RESTART file is opened, DBRECOV will try to open all data bases identified in the RESTART file. For each data base that cannot be opened, DBRECOV will display the message:

Can't re-open DATABASE basename

RESTART will be terminated and the user is returned to the MPE prompt. Make sure the correct data bases are on the system. If the data bases are the correct ones, but they still cannot be opened, use the DBRECOV,ABORT command and RESTART recovery from the previous STOP point.

When all the data bases have been opened, DBRECOV then checks to make sure all the data bases in the RESTART file are set for RESTART. When DBRECOV encounters a data base not in RESTART mode it displays the message:

DATA BASE basename IS NOT IN RESTART MODE.  
RESTART TERMINATED.

RESTART is terminated and the user is returned to the MPE prompt. Make sure the correct data bases are loaded on the system. If the data bases are the correct, but RESTART is still unaccepted recovery, use the DBRECOV,ABORT command and RESTART from the previous STOP point.

The following options should be used to start the recovery process again:

- Find out why the data bases are not in RESTART mode and try to correct the problem. If the problem is irretrievable then take either of the following steps:
  1. Go to the previous STOP point and use the data bases and RESTART file stored to restart roll-forward recovery.
  2. ABORT the current RESTART process. Disable user access on the primary data bases and make a copy for the secondary system. Begin a new logging process on the primary system and a new recovery process on the secondary system.



## Maintaining The Data Base

If all data bases are found, and they are in **RESTART** mode, then the time stamps in the data base root file will be compared to the time stamp in the **RESTART** file. If they do not agree, the following **DBRECOV** error message is printed:

RESTART TIME STAMPS DON'T AGREE WITH DATA BASE TIME STAMPS

This indicates incompatibility of the **RESTART** file and the data bases. The user is returned to the **MPE** prompt. Use the same steps given above to recover from a time stamp error.

Once all the compatibility checks have passed, **DBRECOV** will print a table of the data bases to be recovered:

DATA BASE(S) TO BE RESTARTED:

base1.group.acct

base2.group.acct

.

The user is then be prompted to confirm the restart:

CONTINUE WITH RECOVERY (N/Y)?

Respond "Y" or "YES" to continue, or type "N" or "NO" (or carriage return) to return to the **STOP** point. If any of the data bases cannot be opened during recovery, an **MPE** file error is returned and **DBRECOV RESTART** is terminated. When this happens, go back to the previous **STOP** point and use the data bases, log files, and the **RESTART** file to **RESTART** recovery.

## ABORTING DBRECOV

If a log file in the log set has been damaged or the user cannot **RESTART** recovery for any reason, **ABORT** the current recovery process and begin the mirror data base process again. Once the recovery process terminates, the user is returned to the **MPE** prompt. There are two ways of continuing to mirror the data bases:

1. Go to the previous **STOP** point and use the data bases, log files, and **RESTART** file stored to restart roll-forward recovery. This option is not valid if there is a missing or damaged log file.
2. Disable user access on the primary data bases and make a copy for the secondary system. Begin a new logging process on the primary system and a new recovery process on the secondary system.

## WARNING

When recovery is aborted, the current **RESTART** file is purged and **RESTART** must be done from the previous **STOP** point.

To run the ABORT option:

```
:RUN DBRECOV.PUB.SYS,ABORT
```

Just like the RESTART option the prompt for a data base in the RESTART file appears:

```
WHICH DATA BASE?
```

If the name of a nonexistent data base is entered an error message is printed and the user is prompted once again to enter the name of a data base in the RESTART file. From the data base entered, DBRECOV determines the name of the RESTART file and tries to open it. If the file is opened and is not a RESTART file the following DBRECOV error message is printed:

```
filename is not a DBRECOV RESTART file.
```

and the user is returned to the MPE prompt. This error usually occurs when another file has the same name as the RESTART file has been created on the system. Make sure the file is a RESTART file and try running DBRECOV,ABORT again.

When the correct RESTART file is opened, DBRECOV will look at the file to make sure that it has the same version numbers as the version of DBRECOV being run. If the version numbers do not match then DBRECOV will print the error message:

```
RESTART FILE NOT COMPATIBLE WITH THIS VERSION OF DBRECOV
```

and the MPE prompt is returned. This message means another version DBRECOV is running other than the version that created the current RESTART file. Install the correct version of TurboIMAGE and run DBRECOV,ABORT again.

If the user logon is not the same as the logon when DBRECOV was suspended, the following message is printed:

```
must be logged on as same user and account where DBRECOV was suspended.
```

Log on as the same user and account that was used when DBRECOV was originally suspended and run DBRECOV again.

When the RESTART file is successfully opened, DBRECOV will identify all the data bases in the RESTART file, and verify that they are in restart mode. DBRECOV will then check the time stamps in the RESTART file and the data bases to make sure they match. If the time stamps do not match, the following message is printed:

```
RESTART TIME STAMPS DON'T AGREE WITH DATA BASE TIME STAMPS
```

This indicates incompatibility of the RESTART file with the data bases. The MPE prompt is returned. Locate the correct RESTART file, and run DBRECOV,ABORT again.

## Maintaining The Data Base

Once the RESTART file is opened, DBRECOV will try to open all data bases identified in the RESTART file. For each data base that cannot be opened, DBRECOV will display the message:

```
Can't re-open DATABASE basename
CONTINUE WITH ABORT (N/Y)?
```

DBRECOV then allows the user to double check to be sure that the ABORT is desired. If not all data bases are in the RESTART file, it may mean that this a different set of data bases. Respond "Y" or "YES" to continue the ABORT, and "N", "NO" (or a carriage return) to stop the ABORT.

DBRECOV then checks to make sure all the data bases in the RESTART file are set for RESTART. When DBRECOV encounters a data base not in RESTART mode it prompts:

```
DATA BASE basename IS NOT IN RESTART MODE

CONTINUE (N/Y)?
```

Respond "Y" or "YES" to continue the ABORT, and "N", "NO" (or a carriage return) to stop the ABORT.

Once all compatibility checks have passed, DBRECOV will display all data bases in the RESTART file:

```
DATA BASE(S) WITH RECOVERY TO BE ABORTED:
  base1.group.acct
  base2.group.acct
  .
  .
```

If not all of the data bases can be opened, DBRECOV prints an MPE file error and prompts the user to continue with the ABORT:

```
CONTINUE WITH ABORT (N/Y)?
```

Respond "Y" or "YES" to continue the ABORT, and "N", "NO" (or a carriage return) to stop the ABORT.

Once ABORT is successfully completed, the current RESTART file is purged and the MPE prompt is returned. The user can issue a DBUTIL >>SHOW command. The RESTART flag is disabled and the data base access flag has been reset to what it was before DBRECOV was run.

## PURGING A RESTART FILE

If the RESTART option fails at the current STOP point, the user can ABORT the current recovery process and RESTART the data bases from the previous STOP point. However, if the ABORT option fails the DBRECOV,PURGE command can be used as a last resort to delete the useless RESTART file before restarting with a backup of the data bases and RESTART file.

### WARNING

When using PURGE on a RESTART file, RESTART must be done from the previous STOP point.

```
:RUN DBRECOV.PUB.SYS,PURGE
```

DBRECOV will prompt you for the name of the RESTART file:

```
ENTER RESTART FILENAME?
```

Enter the filename displayed when DBRECOV was stopped. DBRECOV will open the file and verify that it is actually a RESTART file. If DBRECOV is unable to open the RESTART file, an error message is printed and DBRECOV is terminated. The user can either determine that the file is not a RESTART file and delete it, or can RESTART recovery from a previous STOP point. When a RESTART file is restored from a backup, the previous RESTART file writes over the current RESTART file.

If the RESTART file is successfully opened, DBRECOV will display the table of data bases in the RESTART file:

```
RESTART FILE CONTAINS FOLLOWING DATA BASE(S):
```

```
base1.group.acct  
base2.group.acct  
.  
.
```

All the data bases will be opened, and DBRECOV will check if they are all enabled for RESTART. If they are all in RESTART mode, the following message is printed and DBRECOV is terminated:

```
DATA BASE base1.group.acct IS IN RESTART MODE.  
DATA BASE base2.group.acct IS IN RESTART MODE.  
RECOVERY SUSPENDED - USE DBRECOV,ABORT TO ABORT RECOVERY.
```

Run DBRECOV,ABORT to purge the RESTART file.

## Maintaining The Data Base

If none of the data bases in the RESTART file are set for RESTART, the RESTART file will be purged with no further confirmation.

If some of the data bases are not found it will prompt for confirmation to purge the RESTART file:

```
Can't re-open DATABASE basename.  
CONTINUE WITH PURGE (N/Y)?
```

DBRECOV allows the user to double check to be sure that purging this recovery process is desired. If not all data bases are in the RESTART file, it may mean that this is a different set of data bases.

Respond "Y" or "YES" to purge the RESTART file, or "N", "NO", (or carriage return) to stop.

Occasionally, DBRECOV can terminate abnormally due to a bad log file in the log set or a system failure. If the user cannot RESTART recovery from the previous STOP point because of a damaged or missing log file, PURGE the current RESTART file and begin the mirror recovery process again. There are four basic steps to reestablishing the mirror data base system after an abnormal termination of DBRECOV:

- Disable user access on the primary system and store the data bases from the primary system.
- Purge the data bases on the secondary system.
- Restore the data bases from the primary system onto the secondary system.
- Start a new log set, enable user access on the primary system and start roll-forward recovery on the secondary system.

# USING THE DATA BASE UTILITIES

SECTION

8

The TurboIMAGE utility programs are used to create and initialize the data base files and perform various maintenance functions. This section discusses the various utility programs and details the syntax of each utility.

The data base creator is defined by the logon group and account that was used when the Schema Processor created the root file. To execute the DBUTIL >>CREATE command or to change or remove the maintenance word with the DBUTIL >>SET command, the user must be the data base creator. To operate the other utility programs or to enter other DBUTIL commands, the user need not be the data base creator provided they know the maintenance word. If no maintenance word is defined, only the data base creator can execute the other utility programs and the DBUTIL commands that require a maintenance word.

Here is a brief summary of the utility routines and their functions.

Table 8-1. IMAGE Utility Programs

PROGRAM	COMMANDS	FUNCTION
DBLOAD		Loads data entries copied by DBUNLOAD back into the data sets.
DBRECOV *	CONTROL EXIT FILE PRINT RECOVER ROLLBACK RUN	Controls various options which affect execution of DBRECOV. Terminates DBRECOV without re-executing any transactions. Routes log records to individual user files and returns information about recovery. Prints information about data bases or user files specified for recovery. Designates name of data base to be roll-forward recovered. Defines name of data base to be roll-back recovered. Initiates recovery process.
DBRESTOR		Copies the data base to disc from magnetic tape or serial disc volumes created by DBSTORE, or the MPE :STORE or :SYSDUMP commands.
DBSTORE		Copies entire data base including root file and ILR log file to magnetic tape or serial disc volumes.
DBUNLOAD		Copies data entries to specially formatted magnetic tape or serial disc volumes; arranges entries in each data set so that chained access along the primary path is more efficient.

**Table 8-1. IMAGE Utility Programs (continued)**

PROGRAM	COMMANDS	FUNCTION
DBUTIL	ACTIVATE	Prepares a data-base-access file used when accessing a remote data base.
	CREATE	Creates and initializes a data base file for each data set.
	DEACTIVATE	Deactivates a data-base-access file.
	DISABLE	Disables logging, roll-forward recovery, roll-back recovery, ILR, autodefer, access and dumping options.
	ENABLE	Enables logging, roll-forward recovery, roll-back recovery, ILR, autodefer, access and dumping options.
	ERASE	Erases existing data entries from all data sets. Used before loading stored data entries back into the data base.
	EXIT	Terminates DBUTIL program execution.
	HELP	Provides description of all other DBUTIL commands.
	MOVE	Moves TurboIMAGE files across devices.
	PURGE	Purges entire data base including root file, ILR log file, and data set files. Used before restoring a stored data base and before creating a new, restructured data base.
	RELEASE	Suspends security provisions for the root file and data set files.
	SECURE	Restores security provisions suspended by RELEASE command.
DBUTIL	SET	Changes or removes the maintenance word, specifies numbers of buffers to be used, stores log identifier and password into root file. Changes the Native Language of the data base.
	SHOW	Used to display information about data base maintenance.
DBUTIL	VERIFY	Used to determine whether a data-base-access file is activated or deactivated.

\*Refer to DBRECOV in this section for valid options when executing this program.

## UTILITY PROGRAM OPERATION

The utility programs may be run in either job or session mode. DBUTIL, DBSTORE, DBRESTOR, DBUNLOAD, and DBLOAD all require the user to be logged on under the group and account which contains the data base root file. Consequently, these programs may not be used with a remote data base unless you initiate a remote session and run the utility as part of that session. These programs do not allow you to use the :FILE command to equate a data base or data-base-access file to a different file.

DBRECOV is an exception, since :FILE commands are permissible, and since you need not be logged on under the same group and account as the data base root file. However, DBRECOV has the same remote session requirement for remote data base access as the other utility programs.

To execute the DBUTIL >>CREATE command or to change or remove the maintenance word with the DBUTIL >>SET command, you must log on with the same user name that was used when the Schema Processor created the root file; this verifies to TurboIMAGE that you are the data base creator. To operate the other utility programs or enter other DBUTIL commands, you need not be the data base creator provided you know the maintenance word. If no maintenance word is defined, only the data base creator can execute the other utility programs and the DBUTIL commands that require a maintenance word.

### NOTE

To maintain compatibility with earlier versions of DBUTIL, the >>CREATE, >>ERASE, and >>PURGE commands may also be executed by specifying them as DBUTIL entry points.

## Backup Files

The backup files created by DBSTORE and DBUNLOAD may be written to magnetic tape or serial disc volumes. In the discussion of the utility programs that follows, the term *volume* refers to either a magnetic reel or a serial disc pack.

## Error Messages

Some of the error messages are described with the operating instructions for the utility programs. Appendix A contains a complete summary of the error messages issued by these programs.



# DBLOAD

Loads data entries from the backup volume(s) created by the DBUNLOAD program into data sets of the data base.

## Operation

```
1  :FILE DBLOAD[=filename] [;DEV=device]]
2  :RUN DBLOAD.PUB.SYS
   .
   .
3  WHICH DATA BASE? data base name [/maintenance word]

   WARNING: The LANGUAGE of the data base is DIFFERENT from
             the language found on the DBLOAD MEDIA.
   Continue DBLOAD operation? (Y/N)

4  DATA SET n: x ENTRIES
   .
   .
   .
5  END OF VOLUME m,y READ ERRORS RECOVERED
6  DBLOAD OPERATION COMPLETED
   END OF PROGRAM
```

(Refer to the following page for Operation Discussion.)

The volume(s) must have been produced by the DBUNLOAD program, and the data base name on the volume must be exactly the same as the data base name, or root file name, in the current session or in the group and account of the job. DBLOAD issues an error message if the data base name or maintenance word specified is different from the UNLOAD file. In addition, DBLOAD checks that the group and account specified is the same as that in the DBUNLOAD file. To reload the identical data into the data base, the DBUTIL >>ERASE command must be used prior to DBLOAD unless the data base has been purged and recreated. Executing DBUTIL in this way reinitializes the data sets to an empty state while keeping the root file and data sets as catalogued MPE files on the disc.

DBLOAD reads each entry from the backup volume and puts it into the respective data set from which it was read by DBUNLOAD. If a data set in the receiving data base is an automatic master, no entries are directly put into it by DBLOAD, even though there are entries on the volume associated with the data set's number. Automatic master entries are created as needed in the normal fashion when entries are put into the detail data sets related to the automatic master.

DBLOAD calls the DBPUT procedure to put the entries read from the backup volume into the appropriate data sets. In every case, the DBPUT *dset* parameter is a data set number and the *list* parameter is @;. Prior to calling DBPUT, DBLOAD moves each entry from the backup volume into a buffer. The length of the entry is determined by the definition of entries in the target data set. When DBLOAD is calling DBPUT, this length is less than, equal to, or greater than the length of an entry on the backup volume. If the data set entry is larger than the backup entry, the data is left-justified and is padded out to the maximum entry length with binary zeros. If the data entry is smaller than the backup entry, the backup volume record is truncated on the right and the truncated data is lost.

The location of master set entries is based on their search item value which is hashed to an internal location. The detail data set entries are put into consecutive data set records with the appropriate new chain pointer information.

DBLOAD requires exclusive access to the data base. If the data base is already open to any other process, DBLOAD terminates and prints the message: DATA BASE IN USE.

## Parameters

<i>filename</i>	is the name (up to 8 characters) that replaces DBLOAD in the mount request at the operator's console.
<i>device</i>	is the device class name of the device from which the data entries are to be loaded.
<i>data base name</i>	is the name of a TurboIMAGE data base root file catalogued in the current session or job's account and log on group.
<i>maintenance word</i>	is the maintenance word defined by the data base creator. This word must be supplied by anyone other than the data base creator.
<i>n</i>	is the number of the last data set loaded from the backup volume.
<i>x</i>	is the number of entries loaded into the specified data set. <i>x</i> is zero if the data set is an automatic master. Note: this number may not represent the total number of records in the data set if entries existed prior to DBLOAD execution.
<i>m</i>	is the volume number.
<i>y</i>	is the number of read errors from which DBLOAD recovered.

## Operation Discussion

- 1 Is an optional file equation which specifies the device class name for the device from which the data entries are to be loaded. The default is device class TAPE.
- 2 Initiates execution of the DBLOAD program which is in the PUB group and SYS account.
- 3 In session mode, DBLOAD prompts for the data base name and maintenance word. In job mode, the data base name and maintenance word, if any, must be in the record immediately following the :RUN command.

The language ID of the data base is stored along with data when DBUNLOAD has been used to copy the data base to tape or disc. If the data base native language (on disc) is not consistent with the system level native language (on tape), the following message will appear (refer to Table A-8, Appendix A for more information):

WARNING: The LANGUAGE of the data base is DIFFERENT from  
the language found on the DBLOAD MEDIA.  
Continue DBLOAD operation? (Y/N):

## DBLOAD

- 4 After each data set is copied, DBLOAD prints a message on the listfile device which includes the data set number and the number of entries copied.
- 5 When the end of a volume is encountered, DBLOAD prints a message (where  $n$  is the logical device number of the unit, XXXX is the data base name, and  $y$  is the volume number). DBLOAD also instructs the operator to mount a new tape with the following message on the system console:

MOUNT DBLOAD VOLUME XXXX $y$  ON LOGICAL DEVICE  $n$

If the operator mounts the wrong volume, DBLOAD informs the operator with the following message (where  $n$  is the logical device number):

WRONG VOLUME MOUNTED ON LOGICAL DEVICE  $n$

DBLOAD then terminates and you must begin loading the data base again. This may require running DBUTIL, >>ERASE again if any entries have already been loaded.

- 6 After the data entries have been successfully loaded, DBLOAD prints a completion message.

### CONSOLE MESSAGES

After you supply the data base name and DBLOAD opens the input file, a message is displayed on the system console. A tape or serial disc must be mounted on the appropriate unit and identified through an operator reply. Refer to the *Console Operator's Guide* for instructions about console interaction.

### USING CONTROL Y

When executing DBLOAD in session mode, Control Y can be pressed to request the approximate number of entries in the current data set that have already been copied. DBLOAD prints the following message on \$STDLIST:

<CONTROL Y> DATA SET  $n$ : $x$  ENTRIES HAVE BEEN PROCESSED

### Example

```
:RUN DBLOAD.PUB.SYS
```

```
WHICH DATA BASE?  ORDERS/SELL
DATA SET 1:      19 ENTRIES
DATA SET 2:      AUTOMATIC MASTER
DATA SET 3:      25 ENTRIES
DATA SET 4:      12 ENTRIES
DATA SET 5:      32 ENTRIES
DATA SET 6:     258 ENTRIES
END OF VOLUME 1, 0 READ ERRORS RECOVERED
DATA BASE LOADED
```

```
END OF PROGRAM
```

Initiate execution of DBLOAD. Supply the data base name and maintenance word. DBLOAD indicates the number of entries copied. Note that data set 2 is an automatic master so 0 entries are copied; the entries are created as related detail entries are copied to the data base.

One volume was copied with no read errors.

## NOTE

For optimum performance, DBLOAD uses a special "output deferred" mode of operation when it adds entries to a data base. In this mode, data and structural information may not be written back to disc each time DBPUT returns to the DBLOAD program. As a result, the data base is not considered to be logically or structurally complete on disc until the DBLOAD is complete. During DBLOAD the data base being loaded is marked "bad" and only at the completion of a DBLOAD run is the data base marked "good" again.

If during a load, an MPE or hardware crash occurs, the data base is definitely not structurally intact, and it returns its "bad" flag. After the system is brought back up, TurboIMAGE does not allow the data base to be opened for normal access. If you get a "bad" data base error in such a situation, erase the data base with DBUTIL and then perform the load again. (For more information on the error message "Bad Data Base" refer to Table A-9, Appendix A). Alternatively, the data base may be purged with DBUTIL and then restored from a backup copy. A "bad" data base may also be stored, restored, or unloaded (in serial mode only).

## Text Reference

Section 7

# DBRECOV

The DBRECOV program usually is executed after a data base backup copy has been restored by running DBRESTOR in the event of a system failure. DBRECOV reads the logfile containing records of all data base modifications and re-executes the transactions against the restored data base. The DBRECOV file facility enables individual users to be informed of the extent of recovery. (For more information on Roll-Back Recovery, Roll-Forward Recovery and DBRECOV STOP-RESTART refer to Section 7.)

DBRECOV has been modified to make a mirror data base on a secondary system a workable maintenance method. The options used with DBRECOV for this type of recovery and maintenance method are RESTART, ABORT and PURGE. Example 4 shows a mirror data base maintenance step by step.

The commands associated with DBRECOV are >CONTROL, >EXIT, >FILE, >PRINT >RECOVER, >ROLLBACK and >RUN. Each command is discussed separately on the following pages.

## Operation

```
:RUN DBRECOV.PUB.SYS [,OPTION]
```

## Options

RESTART	will restart the roll-forward recovery process. Information in the RESTART file is used by DBRECOV to restart recovery from the point it was stopped.
ABORT	purges the RESTART file and returns the flags to the same settings as before the recovery process was started.
PURGE	deletes the current RESTART file before beginning the mirror data base process again. PURGE can also be used if ABORT fails to abort recovery (possibly due to an inconsistent RESTART file).

Initiates execution of the DBRECOV program which is in the PUB group and SYS account. The recovery system will print a header indicating the version, date and time. It will then prompt for a command input.

## Example 1

Roll-forward recovery of data base ORDERS.

```
:RUN DBRECOV.PUB.SYS
>RECOVER ORDERS
DATABASE ORDERS LAST DBSTORED MON, DEC 10, 1984, 8:30 AM
>RUN
```

## Example 2

Roll-forward recovery of multiple data bases ORDERS and RETAIL. PART and SALES are *filenames*, ADMIN and MKTG are *accounts*. The first 2 is the *rmode* and the second 2 is the *fmode*.

```
:RUN DBRECOV.PUB.SYS
>RECOVER ORDERS
DATABASE ORDERS LAST DBSTORED THURS, DEC 13, 1984, 6:40 PM
>CONTROL NOSTORE
>RECOVER RETAIL
DATABASE RETAIL LAST DBSTORED THURS, DEC 13, 1984, 6:40 PM
>FILE PART,JOHN.ADMIN
>FILE SALES,MARY.MKTG,2,2
>RUN
```

## Example 3

Roll-back recovery of multiple data bases ORDERS and RETAIL.

```
:RUN DBRECOV.PUB.SYS
>CONTROL NOABORTS
>ROLLBACK ORDERS,RETAIL
DATABASE ORDERS LAST USED MON, DEC 3, 1984, 6:00 PM
DATABASE RETAIL LAST USED TUE, DEC 4, 1984, 8:00 AM
>RUN
```

# DBRECOV

## Example 4

DBRECOV STOP-RESTART recovery on data base ORDERS. The following recovery process is done on a secondary system with the mirror data base maintenance and recovery process. The following example begins with a prompt for the user to continue or stop the roll-forward recovery process on the secondary system. When DBRECOV cannot find the next log file, the user can stop the recovery process and backup the secondary system.

```
UNABLE TO OPEN LOG FILE ORDER005
REPLY 'CONTINUE' OR 'STOP' ON CONSOLE.
```

```
STOP
```

```
DATA BASE(S) WITH RECOVERY SUSPENDED:
      ORDERS.DATAMGT.ADMIN
```

```
RESTART RECOVERY WITH LOG FILE:  ORDER005
QUIET BLOCK BEGINS AT RECORD      1005
NUMBER OF RECORDS IN STAGING DISC  1810
RESTART FILE NAME:                  ORDERLOG
:FILE STORE=L;DEV=TAPE
:STORE ORDER@;*L
```

```
:RUN DBRECOV.PUB.SYS,RESTART
WHICH DATA BASE?  ORDERS
```

```
DATA BASE(S) TO BE RESTARTED:
      ORDERS.DATAMGT.ADMIN
```

```
CONTINUE WITH RECOVERY (N/Y)?  Y
```

## Text Reference

Section 7

Used to control various options which affect the execution of DBRECOV. The options are STAMP, NOSTAMP, STORE, NOSTORE, ABORTS, NOABORTS, UNEND, NOUNEND, STOPTIME, ERRORS, STATS, NOSTATS, MODEX, MODE4, and EOF.

### Syntax

```
>CONTROL param [,param...]
```

If the >CONTROL command is not called, the following default conditions apply:

STAMP	is the data base timestamp and must correspond with those written to the logfile.
STORE	is the DBSTORE flag set in the data base root file.
ABORTS	causes transactions which failed to complete due to a program abort to be recovered.
NOUNEND	suppresses the posting of transactions which did not complete or abort prior to a system failure.
ERRORS= <i>nnnn</i>	during job (batch) execution allows no errors (recovery fails).
MODEX	DBRECOV proceeds with exclusive access to the data base, in output deferred mode (see discussion under DBCONTROL in Section 5).
NOSTATS	if the data base is not recovered, no tabulated information will be printed.
STOPTIME	DBRECOV will not check log record timestamps.
EOF= <i>nnnn</i>	DBRECOV will recover all log records.

The >CONTROL command is used to override the default conditions.

If a particular parameter is not specified within a >CONTROL command, the default condition remains in effect. Any number of parameters can be named in any order, but if more than one condition is specified for one parameter, the last condition entered applies. For example:

```
>CONTROL NOSTAMP, STAMP
      or
>CONTROL NOSTAMP
>CONTROL STAMP
```

In both cases, the STAMP condition cancels the previous NOSTAMP. Recovery proceeds with the timestamp check intact.

If more than one data base is specified for simultaneous recovery, they are all governed by the same >CONTROL options.



# DBRECOV

## >CONTROL

In the specifications below, default options are shown in [ ].

### Parameters

[STAMP]	is the timestamp in the data base root file. It is compared with the timestamp in each call to DBOPEN in the logfile. If the timestamps do not match, DBRECOV returns an error message, and terminates recovery for the offending data base.
NOSTAMP	disables the check of the data base and logfile timestamps. Allows recovery to proceed regardless of the data base and logfile timestamps.
[STORE]	is the DBSTORE flag and is checked to insure that the data base has not been modified between restoration and recovery. If the flag has been cleared the >RECOVER command fails.
NOSTORE	disables the check of the DBSTORE flag. Allows recovery to proceed whether or not the DBSTORE flag is set. Useful when the data base has been stored by :STORE or by :SYSDUMP rather than DBSTORE. Storing the data base using :STORE or :SYSDUMP is not recommended.
[ABORTS]	when transactions do not complete due to a program abort, TurboIMAGE appends a special DBEND to the logfile and considers the transactions completed. This enables DBRECOV to recover these transactions and thereby avoids suppressing all subsequent dependent transactions.
NOABORTS	causes DBRECOV to suppress transactions not originally completed by user programs. This option tells TurboIMAGE a user or program abort is abnormal, or incomplete. NOABORTS should only be used if all data base modifications were stopped immediately after the abort and recovery was initiated. Otherwise, recovery may fail due to record table overflow (see below). For more information on both ABORTS and NOABORTS refer to Section 7, "Program Abort and Recovery Considerations".
[NOUNEND]	causes DBRECOV to suppress transactions which did not complete due to a system failure. Recovery may fail due to a record table overflow (see next page).
UNEND	when transactions do not complete due to a system failure (logical transactions which have a DBBEGIN but no corresponding DBEND or DBABEND). These partial transactions are backed out instead of recovered. This may cause the data base to be logically inconsistent. This option can be used if a transaction started early in the day and is still in process before a system failure occurs or if a record table overflow occurs during the DBRECOV process.
STOPTIME= <i>mm/dd/yy</i> <i>hh:mm</i>	causes DBRECOV to impose an artificial end-of-file when the specified log record timestamp (supplied by MPE) is encountered. All log records with subsequent timestamps will not be recovered. (This feature is useful in the event of a user program failure; the data base may be recovered to a point in time before the suspect program began execution.)

Default condition: log record timestamps are not checked by DBRECOV.

**ERRORS=nnnn** controls the maximum number of non-fatal errors allowed during a job (batch) execution. Should *nnnn* be exceeded, DBRECOV terminates and sets the job control word to -1 to indicate an error. However, this check does not take effect until all commands have been parsed and processed.

Default condition: **ERRORS=0**. The number of errors allowed may be altered by retyping the revised **ERRORS** parameter.

**STATS** is used to obtain information from the logfile without actually recovering a data base. Requires use of a file equation to specify the logfile.

### EXAMPLE:

```
:FILE LOGFILE=ORDER001;DEV=TAPE;LABEL=LOG001
:RUN DBRECOV.PUB.SYS
>CONTROL STATS
>RUN
```

This example shows the logfile **ORDER001** residing on tape. The recovery system responds by printing tabulated information from logfiles, similar to tables printed after a data base recovery. However, no data bases are actually opened or recovered.

**[NOSTATS]** negates the **STATS** option; tabulated information is not printed unless a data base is recovered.

**[MODEX]** causes recovery to execute in exclusive (deferred) mode. No other users may access the data base concurrent with recovery.

**MODE4** recovery proceeds in **DBOPEN** mode 4, allowing users in mode 6 to access (read) the data base while recovery is in process.

**EOF=nnnn** causes DBRECOV to impose an artificial end-of-logfile when the specified log record number is encountered. All log records with subsequent numbers will not be recovered. (This feature is useful in the event of a user program failure; the data base may be recovered up to a record number preceding the suspect records. While logging is in progress, the MPE **:SHOWLOGSTATUS** command may be used to determine the current number of records logged before initiating a questionable program.)

Default condition: All log records are recovered by DBRECOV.

# DBRECOV

## >CONTROL

### RECORD NUMBERS AND TABLE OVERFLOW

DBRECOV identifies detail records by their record number. Suppressing aborted or unended transactions during recovery with the NOUNEND or NOABORTS options may cause subsequent detail calls to DBPUT to use different record numbers. In order to change old record numbers into new ones, DBRECOV uses a 1000-entry internal record table. The record table provides a "before" and "after" location of the record numbers for DBPUT calls. If more than 1000 detail records move into new locations, DBRECOV will abort and issue this message:

RECORD TABLE OVERFLOW

To avoid table overflow when using the NOABORTS and NOUNEND options, the data base administrator should halt all modifications against the data base and initiate recovery immediately after a user program aborts.

## Text Reference

Section 7

Used to terminate DBRECOV without recovering any data bases.

### Syntax

>EXIT
-------

### Text Reference

Section 7

# DBRECOV

## >FILE

Routes log records to individual user files, providing the application program with information about the outcome of recovery; provides a useful tool for auditing previous entries. One file for each user can be opened simultaneously by re-entering the >FILE command once for each user, or all users can be directed to a single file.

## Syntax

```
>FILE fileref,userref [,rmode,fmode]
```

## Parameters

*fileref* is an MPE filereference: *filename* [/lockword] [.group[.account]]. This is the destination file for each user's log records.

*userref* is a user reference, specifying which user's log records to copy to this user recovery file. The format is: *username* [/ident].*account*.

The optional identifier, which also must be passed to DBOPEN as part of the password parameter, uniquely identifies persons using the same logon.

*rmode* is for roll-forward recovery only. Directs recovery system to return log records associated with transactions successfully recovered. *Rmode* may take one of the following values:

*rmode*=0 no records associated with recovered transactions are returned. (Default value)

*rmode*=1 log records corresponding to the last successfully recovered call to DBEND of each transaction block are returned.

*rmode*=2 is the sequence of log records associated with the last successfully recovered transaction of each transaction block are returned. In addition, all DBMEMO log records which immediately follow this transaction are returned.

*rmode*=3 all log records associated with successfully recovered transactions for each transaction block are returned.

*fmode*

directs recovery system to return log records associated with transactions which failed to recover. Used with both roll-forward and roll-back recovery.

### WARNING

The (roll-forward) recovery system cannot guarantee that all records associated with unsuccessfully recovered transactions can be returned, since log records which reside in the log system's memory buffers are lost in the event of a system failure.

*Fmode* may take one of the following values:

<i>fmode</i> =0	no records associated with failed transactions are returned. (Default value)
<i>fmode</i> =1	log records corresponding to the first unsuccessfully recovered call to DBBEGIN of each transaction block are returned.
<i>fmode</i> =2	is the sequence of log records associated with the first unsuccessfully recovered transaction of each transaction block are returned.
<i>fmode</i> =3	all log records which could not be recovered are returned.

The >FILE command copies qualified DBOPEN and DBCLOSE log records to each user's recovery file. See "FILE COMMAND" in Section 7 for a full discussion qualifying the return of log records. The optional *rmode* and *fmode* specify the copies of additional log records.

Once the >FILE command is entered, the user recovery file is opened and any existing records are deleted. If the specified user file does not exist, an error is reported unless the file references the logon group and account, in which case the file is automatically created. The state of a log record (either recovered or not) is indicated by a flag word set by DBRECOV in the record itself. MPE WRITELOG records returned by DBRECOV are variable length, since DBRECOV eliminates the continuation records by appending their data to the original WRITELOG record. Consequently, DBRECOV will create recovery files with a variable length record format. However, fixed length records are permitted if the file already exists or a :FILE command is in effect. If a log record exceeds the record size of a user file with fixed length records, the log record is truncated and an error message is printed.

# DBRECOV

## >FILE

### Example

```
>FILE PART/MGR,MARY/TST.MKTG,2,2
```

PART is the *filename*. MGR is the *lockword*. MARY is the *username* and TST is the *identifier*. MKTG is the account. The first 2 is the *rmode* and the second 2 is the *fmode*.

Note that the >FILE command is repeated for each recovery file to be created and for each user whose records will be copied to a user recovery file.

### Text Reference

Section 7

Prints data bases specified for recovery (DBTABLE) or recovery files specified (FILETABLE). Can be used as a check before actually initiating recovery with the >RUN command.

## Syntax

```
>PRINT {DBTABLE  
      FILETABLE}
```

## Parameters

DBTABLE	returns names of data bases specified for recovery
FILETABLE	returns filereferences, userreferences, <i>rmodes</i> and <i>fmodes</i> specified in >FILE commands.

## Example

```
>PRINT DBTABLE
```

```
*****  
*                               DATA BASE STATISTICS                               *  
*                               *                               *  
*  NAME  GROUP  ACCOUNT  OPENS  TRANS  PUTS  DELETES  UPDATES  *  
*  ----  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  *  
*  ORDERS  TST  MKG           2    576   576           0           0  *  
*****
```

## Text Reference

Section 7



# DBRECOV

## >RECOVER

Used to designate the name of a data base to be roll-forward recovered; opens data base root file, validates *logid* and password with MPE, checks DBSTORE flag. Multiple data bases may be roll-forward recovered concurrently if they have all logged to the same logfile by retying the RECOVER command once for each data base or >RECOVER *data base name*, *data base name*.

## Syntax

```
>RECOVER data base name[.group][.account][/maintenance word]
```

If the >RECOVER command is accepted, the following message is returned:

DATABASE *data base name* LAST DBSTORED *day, date, time*

The following conditions must be satisfied before the >RECOVER command will be accepted:

1. The data base must be accessible to the user (data base administrator) running DBRECOV. This user must either be the creator of the data base or know the maintenance word. If the data base resides in a group or account different from the user's logon, the MPE file security must permit the user read and write access to the data base files.
2. The data base must be enabled for recovery.
3. The log identifier characteristics (name, password, logfile name and device type) must not have been altered since the logfile was generated. This restriction applies to MPE log commands as well as those provided for TurboIMAGE by DBUTIL. This is necessary since the MPE log identifier is used by TurboIMAGE to obtain the name and device type of the logfile.

The >RECOVER command will not be accepted if the *logid* is unknown to MPE. However, if the *logid* is known to MPE but specifies the wrong logfile, this condition is not sensed at this time and >RECOVER will be accepted. DBRECOV will generate erroneous data in the data base if the data base is recovered with the wrong log file.

4. The DBSTORE flag must be set, indicating that the data base has not been modified between restoration and roll-forward recovery. This check can be overridden by the NOSTORE option of the >CONTROL command.
5. No other users can be accessing the data base when >RECOVER is called. Exception: when MODE4 option of the >CONTROL command is specified, the data base may be concurrently accessed in Mode 6 (read only).

Note that the >RECOVER command itself does not initiate recovery, but makes several preparatory checks. The recovery system is actually initiated by the >RUN command.

### Example

```
>RECOVER ORDERS, RETAIL  
  DATABASE ORDERS LAST DBSTORED MON, DEC 3, 1984, 6:30 PM  
  DATABASE RETAIL LAST DBSTORED MON, DEC 4, 1984, 10:00 PM
```

"ORDERS" and "RETAIL" are *data base names*.

### Text Reference

Section 7

# DBRECOV

## >ROLLBACK

Rolls out any incomplete transactions following a system crash. Multiple data bases may be roll-back recovered concurrently by typing >ROLLBACK *data base name*, *data base name*.

### Syntax

```
>ROLLBACK data base name[group[.account]][/maintenance word]
```

### Parameters

*dbname* is the name of individual data base(s) to be rolled-back.

If the >ROLLBACK command is accepted, the following message is returned:

DATABASE *data base name* LAST USED *day, date, time*

The following conditions must be satisfied before the >ROLLBACK command will be accepted:

1. The data base must be accessible to the user (data base administrator) running DBRECOV. This user must either be the creator of the data base or know the maintenance word. If the data base resides in a group or account different from the user's logon, the MPE file security must permit the user read and write access to the data base files.
2. The data base must have been enabled for roll-back recovery.
3. The log identifier characteristics (name, password, logfile name and device type) must not have been altered since the logfile was generated. This restriction applies to MPE log commands as well as those provided by TurboIMAGE by DBUTIL. This is necessary since the MPE log identifier is used by TurboIMAGE to obtain the name and device of the logfile.
4. When roll-back is enabled, DBUTIL sets a roll-back flag to indicate that roll-back is enabled for the data base. The roll-back time stamp is updated when the data base is first opened and is logged to the log file. Roll-back recovery then uses the time stamp during recovery to verify the correct log file for each data base.
5. No other users can be accessing the data base when >ROLLBACK is called. The data base may be concurrently accessed by users when the >CONTROL command is specified with MODE4 option.

Note that the >ROLLBACK command itself does not initiate recovery, but makes several preparatory checks. The recovery system is actually initiated by the >RUN command.

The following commands are used with roll-back:

```
>FILE  
>PRINT  
>CONTROL
```

The >FILE command optional parameter *rmode* is not used with >ROLLBACK.

The following >CONTROL options are not applicable with roll-back:

STAMP, NOSTAMP, STORE, NOSTORE, STOPTIME

## Example

```
>CONTROL .NOSTATS  
>ROLLBACK ORDERS  
DATABASE ORDERS LAST USED MON, DEC 10, 1984, 6:00 PM  
>RUN
```

ORDERS is the *data base name*.

## Text Reference

Section 7

# DBRECOV

## >RUN

Initiates recovery-process. The recovery system opens the logfile and validates the log identifier before roll-forward recovery or roll-back recovery begins.

## Syntax

>RUN

In order for recovery to succeed, the logfile must be accessible to the data base administrator. This means that the data base administrator must either be the creator of the log identifier used to create the logfile, or have system manager capability. If the data base administrator does not have system manager capability, and if the logfile resides on disc in a group and account different from logon, then the administrator must have read access to the logfile according to MPE file security. File equations are permitted. However, the fully qualified file name of the expected logfile must be specified. If the logfile resides on tape, the data base administrator must know the volume identifier, so that the operator may respond to the logfile tape mount request.

If recovery succeeds, tabulated information is displayed and the program is terminated. A table of process statistics includes the number of DBPUT, DBDELETE, DBUPDATE log records processed and the total transactions for each process.

When using roll-forward recovery an asterisk (\*) may appear next to any process indicating that either a DBCLOSE record is missing or some transactions(s) could not be recovered. Therefore, no asterisk for a process in the table of process statistics indicates that all transactions were recovered.

The same table information is displayed when using roll-back recovery, however there is a slight difference. The database table will list all incomplete transactions or DBPUT, DBDELETE, DBUPDATE log records which were "rolled-out". An asterisk (\*) will appear next to these processes.

A table of data base statistics includes the same information totaled for each data base. A logging system table includes the log identifier, logfile information, and recovery file information if this facility is used. Refer to "Recovery Tables" in Section 7 for more information on Process, Database, Logging and Recovery Tables.

## Example 1

Roll-forward recovery of data base ORDERS.

```
:RUN DBRECOV.PUB.SYS
>RECOVER ORDERS
DATABASE ORDERS LAST DBSTORED TUE, DEC 4, 1984, 4:00 PM
>RUN
```

## Example 2

Roll-back recovery of data base ORDERS.

```
:RUN DBRECOV.PUB.SYS
>ROLLBACK ORDERS
DATABASE ORDERS LAST USED MON, DEC 10, 1984, 6:00 PM
>RUN
```

# DBRESTOR

Copies a data base from the backup volume(s) created by the DBSTORE program, or the MPE :STORE or :SYSDUMP commands, to disc.

## Operation

```
1  [:FILE DBRESTOR [=filename][;DEV=device][;REC=recsize][;{BUF  
2  :RUN DBRESTOR.PUB.SYS  
    :  
    :  
3  WHICH DATA BASE? data base name [/maintenance word]  
4  DATA BASE RESTORED  
   END OF PROGRAM
```

## Parameters

<i>filename</i>	is a name (up to 8 characters) that replaces DBRESTOR in the mount prompt at the operator's console.
<i>device</i>	is the device class name of the device from which the data base is to be recorded.
<i>recsize</i>	is the record size of the record to be restored. <i>recsize</i> must be at least as large as the record written to the device to avoid losing data.
<i>data base name</i>	is the name of a TurboIMAGE data base root file to be restored.
<i>maintenance word</i>	is the maintenance word defined by the data base creator. This word must be supplied by anyone other than the data base creator.

## Operation Discussion

- 1 Is an optional file equation which specifies the device class name for the device from which the data base is to be restored, the record size of the records to be restored, and whether the records are buffered or not. The default device class is TAPE.
- 2 Initiates execution of the DBRESTOR program which is in the PUB group and SYS account.
- 3 In session mode, DBRESTOR prompts for the data base name and maintenance word. In job mode, the data base name and maintenance word, if any, must be in the record immediately following the :RUN command.
- 4 After DBRESTOR has created the root file and data set files and restored the data to these files, it prints a confirmation message. If Intrinsic Level Recovery (ILR) is enabled when you run DBSTORE, DBSTORE stores the ILR file associated with the data base on the same tape or serial disc. When the data base is restored with DBRESTOR, the ILR file is automatically restored along with the data base.

## CONSOLE MESSAGES

After you supply the data base name and DBRESTOR opens the input file, a message is displayed on the system console. A tape or serial disc must be mounted on the appropriate unit and identified through an operator reply. Refer to the *Console Operator's Guide* for instructions about console interaction.

If the data base is on more than one volume, another message is displayed on the system console. The operator must mount the next volume in the sequence and ready the unit. If the volume which is mounted is not the correct format, the operator is notified through a console message. If the correct volume is available, the current one should be removed and the correct one mounted. The operator must then enter a reply on the console.

## Example

:JOB MRG.ACCOUNTA	Initiate job.
:RUN DBRESTOR.PUB.SYS	Initiate DBRESTOR.
ORDERS/SELL	Specify data base name and maintenance word
:EOJ	Terminate job.

After creating the files and restoring the file contents, DBRESTOR prints the following message on \$STDLIST:

DATA BASE RESTORED



# DBSTORE

Stores the data base root file and all data set files to a tape or serial disc in a format compatible with backup files created by the MPE :STORE and :SYSDUMP commands. DBSTORE differs from these commands in that it handles only TurboIMAGE data bases.

## Operation

```
1  [:FILE DBSTORE[=filename] [;DEV=device] [;REC=recsize] [;{BUF  
    NOBUF }]]  
2  :RUN DBSTORE.PUB.SYS  
    .  
3  WHICH DATA BASE? data base name [/maintenance word]  
4  DATA BASE STORED  
    END OF PROGRAM
```

Before copying the files, DBSTORE gains semi-exclusive access to the referenced data base; that is, DBSTORE determines that the only other data base activity consists of other users executing DBSTORE or application programs which open the data base in mode 6 or 8. If DBSTORE cannot gain semi-exclusive access, it terminates and prints the following message: DATA BASE IN USE.

You must be the data base creator or provide the maintenance word to use DBSTORE.

## Parameters

<i>filename</i>	is the name (up to 8 characters) that replaces DBSTORE in the mount request at the operator's console.
<i>device</i>	is the device class name of the device on which the data entries are to be stored.
<i>recsize</i>	is the record size of the record to be written to the device; <i>recsize</i> must be modulo 256 words and less than the configured record size for the device.
<i>data base name</i>	is the name of a TurboIMAGE data base to be stored or copied.
<i>maintenance word</i>	is the maintenance word defined by the data base creator. This word must be supplied by anyone other than the data base creator.

## Operation Discussion

- 1 Is the optional file equation which specifies the device class name for the device on which the data base is to be stored, the record size of the records written to the device, and whether records are to be buffered. The default device class is TAPE.
- 2 Initiates execution of the DBSTORE program which is in the PUB group and SYS account.
- 3 In session mode, DBSTORE prompts for the data base name and maintenance word. In job mode, the data base name and maintenance word, if any, must be in the record immediately following the :RUN command.
- 4 After DBSTORE has copied the root file and all data set files, it prints a message to signal completion. If Intrinsic Level Recovery (ILR) is enabled when you run DBSTORE, DBSTORE stores the ILR file associated with the data base on the same tape or serial disc. When the data base is restored with DBRESTOR, the log file is automatically restored along with the data base. This assures that the ILR file remains associated with the data base through the store and subsequent restore.

### NOTE

Serial disc packs must be initialized by the console operator using the VINIT subsystem. Refer to the *Console Operator's Guide*.

## LOGGING

DBSTORE updates a timestamp and store flag in the data base root file before storing the data base. The timestamp designates the date and time of the DBSTORE operation, and is used by DBRECOV to help identify the correspondence between logfiles and backup data bases.

The store flag is set by DBSTORE to indicate that the data base has been stored; this flag is cleared (reset) when the first modification to the data base occurs by a call to DBPUT, DBUPDATE, or DBDELETE. Both DBRECOV and DBUTIL interrogate the status of the DBSTORE flag. DBRECOV (roll-forward) checks this flag to insure that no one has modified the backup data base prior to recovery. DBUTIL checks this flag whenever logging and recovery is enabled, since a valid backup data base copy must exist for roll-forward recovery to be possible. If the store flag is not set when a DBUTIL user enables the logging option a warning is printed:

**WARNING: data base modified and not DBSTORED**

This warning does not necessarily indicate that a valid backup does not exist, since either :SYSDUMP or :STORE may have been used instead of DBSTORE. Since neither :SYSDUMP or :STORE update the data base timestamp and store flag, the protection afforded by these mechanisms is not available if this form of backup is selected. For this reason, it is highly recommended to use DBSTORE as the backup facility when logging. See Section 7 for further discussion of "Transaction Logging and Recovery".

# DBSTORE

If the mirror data base maintenance method is being used, storing the data base on the secondary system can be done differently than using the DBSTORE process. When using DBRECOV STOP-RESTART recovery on the data base, storing the data base, RESTART file, and the log files that were processed since the last successful RESTART can be stored with an MPE STORE. DBRECOV STOP-RESTART places a timestamp in the RESTART file and in the data base to identify which RESTART file to apply to which data base. If naming conventions have been followed, an MPE STORE @ can be used to store all the necessary files and data base(s). If a DBSTORE is used, the user must remember to MPE STORE the RESTART file and the log files. For more information on DBRECOV STOP-RESTART refer to Section 7 "The Mirror Data Base" and "Storing the Data Bases".

## CONSOLE MESSAGES

After you supply the data base name and DBSTORE opens the output file, a message is displayed on the system console. A tape or serial disc must be mounted on the appropriate unit and identified through an operator reply. Refer to the *Console Operator's Guide* for instructions about console interaction.

If more than one volume is required to store the data base, a request is displayed on the console for the next one. The next tape or disc must be mounted and the unit readied. (The serial disc unit must be switched from RUN to STOP to RUN prior to entering a reply if a new volume has not been mounted.) The volume which has been removed should be properly labeled with the data base name and volume number.

## Example

:JOB MGR.ACCOUNTA	Initiate job.
:RUN DBSTORE.PUB.SYS	Initiate DBSTORE program.
ORDERS/SELL	Supply data base name and maintenance word.
:EOJ	Terminate job.

After copying the ORDERS root file and all data sets, DBSTORE prints the following message on \$STDLIST:

DATA BASE STORED

Copies the data entries from each data set to specially formatted tape or serial disc volumes.

## Operation

```
1  [:FILE DBUNLOAD[=filename] [;DEV=device]]
2  :RUN DBUNLOAD.PUB.SYS  [,CHAINED]
   .                        [,SERIAL ]
   .
3  WHICH DATA BASE?data base name[/maintenance word]
   .
   .
4  DATA SET n: x ENTRIES EXPECTED, x ENTRIES PROCESSED.
5  END OF VOLUME m, y WRITE ERRORS RECOVERED
6  DATA BASE UNLOADED

END OF PROGRAM
```

(Refer to the following page for Operation Discussion.)

## Parameters

<i>filename</i>	is the name (up to 8 characters) that replaces DBUNLOAD in the mount prompt at the operator's console.  If you want information about your data set chains without actually performing a DBUNLOAD, supply \$NULL as the <i>filename</i> . This will cause a simulated unloading of the data base, preventing the need to mount a tape.
<i>device</i>	is the device class name of the device to which the data entries are to be copied. (Serial disc packs must be initialized by the console operator with the VINIT subsystem. Refer to the <i>Console Operator's Guide</i> ).
<i>data base name</i>	is the name of the TurboIMAGE data base to be unloaded.
<i>maintenance word</i>	is the maintenance word defined by the data base creator. This word must be supplied by anyone other than the data base creator.
<i>n</i>	is the number of data sets in the data base.
<i>x</i>	is the number of entries (expected) and the number of entries processed or copied from the specified data set.
<i>m</i>	is the number of the volume.
<i>y</i>	is the number of write errors from which DBUNLOAD has successfully recovered.

# DBUNLOAD

DBUNLOAD is necessary if you want to modify the data base structure, for example, to increase the capacity of a data set. To increase a capacity, unload the entries, purge the data base, change the schema and create a new root file, execute DBUTIL >>CREATE, and then reload the data entries from the volumes created by DBUNLOAD.

The data sets are unloaded in the order that they were defined in the original schema. No data set names are recorded on the backup volume(s); entries are merely associated with the corresponding data set from which they are read. DBUNLOAD calls the DBGET procedure to read each entry from each set of the data base using a *list* parameter of @; to read the complete entry. Values for data items appear in each entry in the same order as the items were mentioned in the data set definition in the schema. The Language ID is copied along with the data of the data base.

DBUNLOAD requires exclusive access to the data base. If the data base is already open by any other process, DBUNLOAD prints the message: DATA BASE IN USE and prompts again for a data base name.

DBUNLOAD operates in either chained or serial mode. The mode is determined by the entry point specified with the :RUN command. The default entry, if none is specified, is chained.

- In serial mode, DBUNLOAD copies the data entries serially in record number order. "Stand-alone" detail data sets, those which are not tied to any master data sets through specified search item paths, are always unloaded serially.
- In chained mode, DBUNLOAD copies all of the detail entries with the same primary path search item value to contiguous locations on the backup file. The ordering of the search item values from the primary path is based on the physical order of the matching value in the associated master data set. Figure 8-1 illustrates the method for unloading a data set in chained mode. After the data base is reloaded, chained access along the primary path is more efficient.

## BROKEN CHAINS

If a chained DBUNLOAD encounters a broken chain, it will unload all entries in the chain down to, but not including the break. It will then go to the end of the chain and unload all entries up to the break. In some instances, this will save all entries in the chain. In any case, the order of the entries is preserved. Information about each broken chain in a data set is printed before the end-of-the-data-set summary (statement 4 below).

## Operation Discussion

- 1 Is an optional file equation which specifies the device class name for the device on which the data entries are to be copied. The default is device class TAPE.
- 2 Initiates execution of the DBUNLOAD program which is in the PUB group and SYS account.
- 3 In session mode, DBUNLOAD prompts for the data base name and maintenance word. In job mode, the data base name and maintenance word, if any, must be in the record immediately following the :RUN command.
- 4 After copying a data set without detecting a broken chain, DBUNLOAD prints a message which includes the data set number and the number of entries copied.

If DBUNLOAD detects a broken chain, the following messages are also returned:

```
DATA SET  $n$ :  Broken Chain at Entry # $p$  [, following Entry # $q$ ]
              Chain Head is Entry # $r$  of Data Set # $s$ 
              Key =  $k$ 
               $l$  entries [expected,  $j$  entries salvaged]
```

where

$p$  is the entry number where the break was detected.  
 $q$  is the number of the entry last unloaded from the front of the chain, if any.  
 $r$  is the entry number of the chain head.  
 $s$  is the data set number of the chain head.  
 $k$  is the value of the key of the broken chain.  
 $l$  is the length of the chain according to the user label.  
 $j$  is the number of entries salvaged from the chain.

These 4 message lines are repeated for every broken chain in the data set, followed by the end-of-data-set summary which reports the number of lost entries, if any:

```
DATA SET  $n$ :   $x$  ENTRIES [EXPECTED,  $t$  LOST!!]
```

For example:

```
DATA SET 1:  3 ENTRIES
```

```
DATA SET 2:  Broken Chain at Entry #2, following Entry #1
              Chain Head is Entry #5 of Data Set #1
              KEY = AA
              4 entries expected, 3 entries salvaged
```

```
DATA SET 2:  11 ENTRIES EXPECTED; 1 LOST!!
```

- 5 When the end of a volume is encountered, DBUNLOAD prints this message:

```
END OF VOLUME  $a$ ,  $b$  WRITE ERRORS RECOVERED
```

where  $a$  is the number of the volume and  $b$  is the number of write errors from which DBUNLOAD successfully recovered. DBUNLOAD also instructs the operator to save the current volume and mount a new one by printing the following two messages on the system console (where  $n$  is the logical device number of the tape or disc drive and  $y$  is the volume number):

```
SAVE VOLUME ON LOGICAL DEVICE  $n$  AS  $y$ 
MOUNT NEXT VOLUME ON LOGICAL DEVICE  $n$ .
```

- 6 After the data sets have been successfully copied, DBUNLOAD issues a completion message.

```
DATA BASE UNLOADED
END OF PROGRAM
```

# DBUNLOAD

## CONSOLE MESSAGES

After you supply the data base name and DBUNLOAD opens the output file, a message is displayed on the system console. A tape or serial disc must be mounted on the appropriate unit and identified through an operator reply. Refer to the *Console Operator's Guide* for instructions about console interaction.

## USING CONTROL Y

When executing DBUNLOAD in session mode, you can press Control Y to request the approximate number of entries in the current data set which have already been written. DBUNLOAD then prints the following message on \$STDLIST:

```
<CONTROL Y>DATA SET n:x ENTRIES HAVE BEEN PROCESSED
```

## WRITING ERRORS

If an unrecoverable write error occurs, DBUNLOAD prints the message:

```
UNRECOVERABLE WRITE ERROR, RESTARTING AT BEGINNING OF VOLUME
```

and attempts to recover by starting the current volume again. It also sends this message to the system operator (where *n* is the logical device number of the unit):

```
WRITE PROBLEMS TRY ANOTHER VOLUME ON LOGICAL DEVICE n
```

If an excessive number of non-fatal write errors occur, DBUNLOAD again attempts to recover from the beginning of the volume after printing the following message on the \$STDLIST and sends the same message to the system operator as described for unrecoverable errors above:

```
EXCESSIVE WRITE ERROR RECOVERIES, RESTARTING AT BEGINNING OF VOLUME
```

## Example (Session Mode)

```
:RUN DBUNLOAD.PUB.SYS
```

```
.
```

```
WHICH DATA BASE? ORDERS
```

```
DATA SET 1: 3 ENTRIES EXPECTED, 3 ENTRIES PROCESSED.
```

```
DATA SET 2: 11 ENTRIES EXPECTED, 11 ENTRIES PROCESSED.
```

```
END OF VOLUME 1, 0 WRITE ERRORS RECOVERED
```

```
DATA BASE UNLOADED
```

```
END OF PROGRAM
```

## Example (Job Mode)

:JOB MGR.ACCOUNTA	Initiate job.
:RUN DBUNLOAD.PUB.SYS	Initiate execution of DBUNLOAD.
ORDERS	Specify data base name.
:EOJ	Initiate end of job.

Since the user in this example is the data base creator, a maintenance word is not provided. The DBUNLOAD program is executed in CHAINED mode by default because no entry is specified.

As the job executes, the following information is printed on the \$STDLIST:

```
DATA SET 1:      9 ENTRIES EXPECTED, 9 ENTRIES PROCESSED.
DATA SET 2:     50 ENTRIES EXPECTED, 50 ENTRIES PROCESSED.
DATA SET 3:     24 ENTRIES EXPECTED, 24 ENTRIES PROCESSED.
DATA SET 4:     12 ENTRIES EXPECTED, 12 ENTRIES PROCESSED.
DATA SET 5:      5 ENTRIES EXPECTED, 5 ENTRIES PROCESSED.
DATA SET 6:      0 ENTRIES EXPECTED, 0 ENTRIES PROCESSED.

END OF VOLUME 1,0 WRITE ERRORS RECOVERED

DATA BASE UNLOADED

END OF PROGRAM
```



# DBUNLOAD

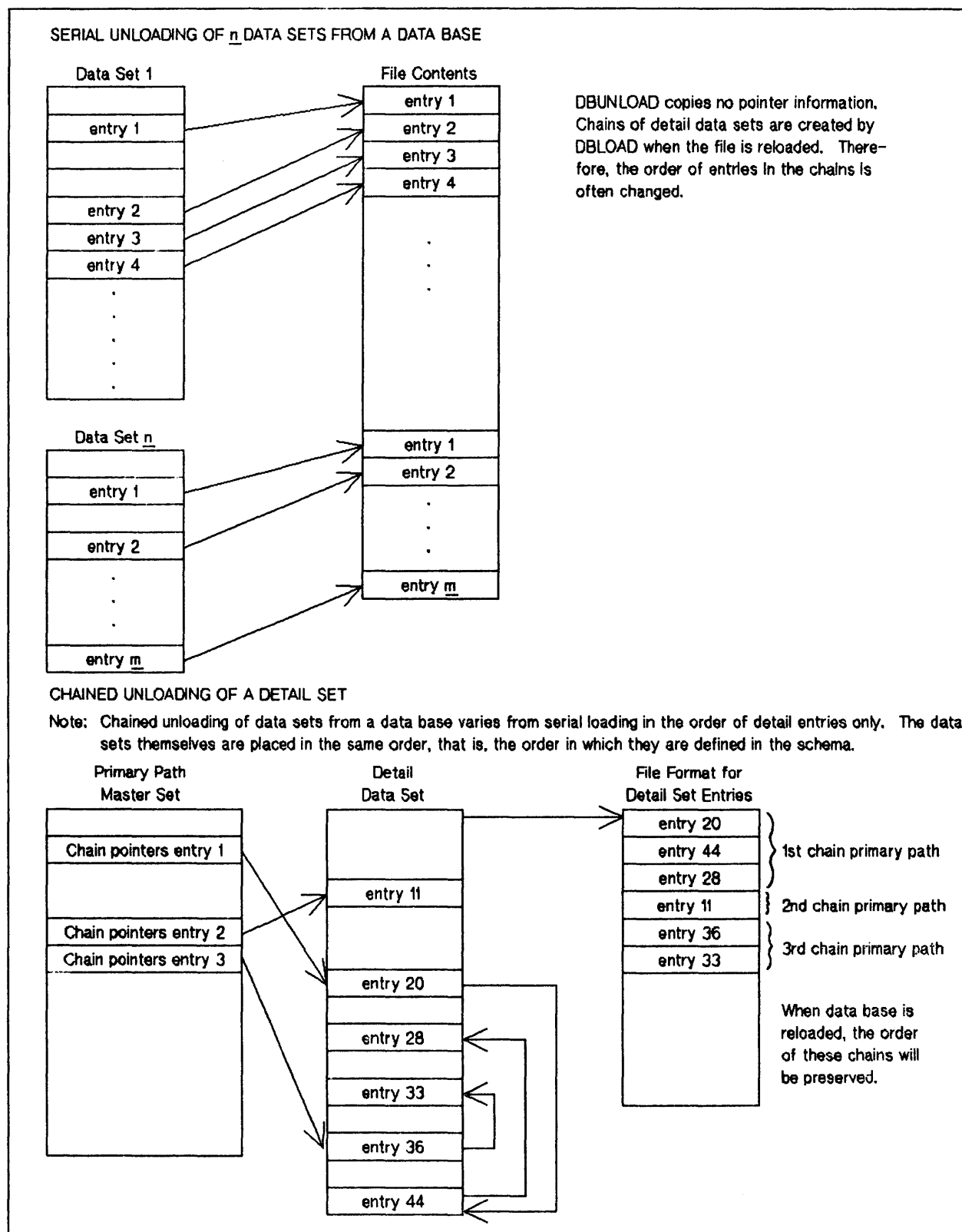


Figure 8-1. DBUNLOAD File, Sequence of Entries

The DBUTIL program performs several different functions according to the command you enter. Each DBUTIL command is described separately on the following pages.

## Operation

```
1  :RUN DBUTIL.PUB.SYS
2  >>command
```

## Operation Discussion

- 1 Initiates execution of the DBUTIL program which is in the PUB group and SYS account.
- 2 Prompts for a DBUTIL >>command. Enter one of the following:

HELP	VERIFY
CREATE	SET
ERASE	ENABLE
MOVE	DISABLE
PURGE	RELEASE
DEACTIVATE	SECURE
ACTIVATE	SHOW
EXIT	

DBUTIL commands may be abbreviated to the first three characters. For example, >>CREATE may be abbreviated to >>CRE.

When using the >>CREATE, >>PURGE, or >>ERASE commands, you can bypass the command prompt by specifying the full command as an entry point with the :RUN command; for example, :RUN DBUTIL.PUB.SYS, CREATE. If you use an entry point, TurboIMAGE prompts you for the data base name and, optionally, for the maintenance word.

Data base name: *data base name* [/maintenance word]

*data base name* is the name of a TurboIMAGE data base root file catalogued in the current session or job's account and log on group.

*maintenance word* is an optional ASCII string, one to eight characters long with no commas or semicolons, which defines a password to be used by anyone other than the data base creator to enable them to execute certain DBUTIL commands, and operate the other utility programs. (The data base creator may also define or change the maintenance word by using the SET command).

In job mode, the data base name and maintenance word, if any, must be in the record immediately following the :RUN command. Note that to perform any DBUTIL command except >>SHOW, >>HELP, or >>EXIT, you must have exclusive access to the data base or data-base-access file.

# DBUTIL

## >>ACTIVATE

Activates the data-base-access file for use with DBOPEN. Before using this command, read the description of remote data base access in Section 9.

This command should be used to prepare a data-base-access file before accessing a remote data base residing on another HP 3000.

## Syntax

```
>>ACT[IVATE] data-base-access file name
```

For example:

```
ACTIVATE ORDDBA
```

Where ORDDBA=*data-base-access file name*

## Parameter

*data-base-access file name* is the name of the data-base-access file that you created with the Editor.

The data-base-access file may have the same name as a data base on the remote system or it may have the name of another data-base-access file on the remote system.

## UNEXPECTED RESULTS

TurboIMAGE checks that the file code is 0, the record length does not exceed 128 characters, the file is unnumbered, and the file has at least three records. An appropriate error message is returned if any of these conditions is violated. If all of the conditions are satisfied, DBUTIL prints the message:

Verification follows:

and the syntax of the file is checked record by record. The monitoring messages associated with the file records are of the form:

```
FILE command:      <result>
DSLIN command:     <result>
HELLO command:     <result>
```

where <result> is "Looks good" if there are no errors associated with the record. Appendix A lists the record errors (results) which would cause the file to be rejected.

### Example

:RUN DBUTIL.PUB.SYS

Initiate DBUTIL execution.

>>ACT ORDDBA

Enter abbreviated form of  
ACTIVATE command and  
data-base-access file name.

Verfication follows

FILE command: Looks good

DSLIN command: Looks good

HELLO command: Looks good

HELLO command: Looks good

ACTIVATED

>>

DBUTIL checks the structure of the file named "ORDDBA" for correct format and activates the file. You will not be able to edit the file unless you deactivate it using the DBUTIL >>DEACTIVATE command.

# DBUTIL

## >>CREATE

Creates and initializes a file for each data set in the data base.

Once the Schema Processor has created the root file, the data base creator must build a file for each data set in the data base using the >>CREATE command. DBUTIL initializes each data set to zeros and saves it as a catalogued MPE file in the same log on group as the root file, on the device classes specified in the schema. The data set file names are created by appending two digits to the root file name. If the root file is named XXXX, then the first data set defined in the schema is in a file named XXXX01, the second data set file is named XXXX02. In order to save files for the maximum of 199 data sets per data base, files are incremented from XXXX01-99, XXXXA1-A9, XXXXB1-B9, up to XXXXJ9.

To execute the DBUTIL program to create and initialize the data base you must be the data base creator; that is, you must log on with the same user name, account and group that was used to run the Schema Processor and create the root file. After DBUTIL has created and initialized the data base files, it prints a confirmation message on the listfile device and prompts for another command.

## Syntax

```
>>CRE[ATE] data base name[/maintenance word]
```

For example:

```
CREATE ORDERS/SELL
```

Where ORDERS=*data base name* / SELL=*maintenance word*

## Parameters

*data base name* is the name of a TurboIMAGE data base being created.

*maintenance word* is an optional ASCII string, one to eight characters long with no commas or semicolons, which defines a password to be used by anyone other than the data base creator to enable them to execute certain DBUTIL commands and operate the other utility programs. (The data base creator may also define or change the maintenance word by using the >>SET command).

## Example (Session Mode)

<pre>:RUN DBUTIL.PUB.SYS . . &gt;&gt;CREATE ORDERS/SELL Data base ORDERS has been CREATED &gt;&gt;</pre>	<pre>Initiate DBUTIL execution.  Respond to DBUTIL prompt with &gt;&gt;CREATE command, data base name, and maintenance word.</pre>
--	--

DBUTIL creates, initializes, and saves files named ORDERS01, ORDERS02, and so forth, one file for each data set. These constitute the empty data base.

### Example (Job Mode)

:JOB MGR.ACCOUNTA	Initiate job.
:RUN DBUTIL.PUB.SYS	Initiate DBUTIL execution.
CREATE ORDERS/SELL	Enter >>CREATE command and parameters.
EXIT	Terminate DBUTIL.
:EOJ	Terminate job.

After the data files are created and initialized, DBUTIL prints the following message on the listfile device:

DATA BASE ORDERS HAS BEEN CREATED

<b>NOTE</b>
-------------

>>CREATE will fail if the native language defined for the data base is not supported at the system level. (Refer to Table A-3, Appendix A or *NLS/3000 Reference Manual* for more information.)

# DBUTIL

## >>DEACTIVATE

Deactivates the data-base-access file to allow modifications to the file or to disallow remote data base access.

This command is used before you change the contents of the data-base-access file. (Refer to Section 9 for more information about accessing remote data bases.)

If DBUTIL successfully deactivates the file, it prints a confirmation message on the listfile device.

## Syntax

```
>>DEA[CTIVATE] data-base-access file name
```

For example:

```
DEACTIVATE ORDDBA
```

Where ORDDBA=*data-base-access file name*

## Parameter

*data-base-access file name* is the name of the data-base-access file to be deactivated.

## Example

:RUN DBUTIL.PUB.SYS	Initiate DBUTIL execution
.	
.	
>>DEACTIVATE ORDDBA	Enter a >>DEACTIVATE command and the
DEACTIVATED	data-base-access file name.
>>	

Disables the access, automatic deferred output, dumping, logging and recovery options.

## Syntax

```
>>DIS[ABLE] data base name[/maint word] FOR option[,option..]
```

For example:

```
DISABLE ORDERS FOR LOGGING,RECOVERY
```

Where ORDERS=*data base name* and LOGGING,RECOVERY=*options*

## Parameter

*data base name* is the name of a TurboIMAGE data base root file cataloged in the current session or job's account and logon group.

## Options

ACCESS	disables user access to the data base.
AUTODEFER	disables automatic deferred output for the data base. AUTODEFER must be disabled if ILR or roll-back recovery are to be enabled for a data base.
DUMPING	disables the automatic dumping of the user's stack and the data base control block in the event of a TurboIMAGE abort. Under most circumstances dumping should be disabled. When enabled, DUMPING creates a file (before TurboIMAGE aborts) which may prove helpful in determining the cause of such problems as a corrupted control block.
ILR	disables Intrinsic Level Recovery facility. ROLLBACK must be disabled first.
LOGGING	disables the data base roll-forward logging facility.
RECOVERY	disables the data base roll-forward recovery facility.
ROLLBACK	disables the data base roll-back logging facility.



# DBUTIL

## >>DISABLE

### Default Conditions

Access is Enabled  
Autodefer is Disabled  
Dumping is Disabled  
ILR is Disabled  
Logging is Disabled  
Recovery is Disabled  
Roll-Back is Disabled

### Example

```
:RUN DBUTIL.PUB.SYS
.
.
>>DISABLE ORDERS FOR ACCESS
Access is Disabled
>>
```

Enables the access, automatic deferred output, dumping, logging and recovery options.

## Syntax

```
>>ENA[BLE] data base name[/maint word] FOR option [,option...]
```

For example:

```
ENABLE RETAIL FOR LOGGING
```

Where RETAIL=*data base name* and LOGGING=*option*

## Parameter

*data base name* is the name of a TurboIMAGE data base being enabled.

## Options

ACCESS	enables user access to the data base.
AUTODEFER	enables automatic deferred output for the data base. Output deferred mode (AUTODEFER) allows the user the ability to speed up processing time by deferring buffer posting until it is required by buffer management. Since AUTODEFER overrides the normal flushing of file labels and buffers on DBPUTs, DBDELETES and DBUPDATES TurboIMAGE will be unable to recover lost data in the event of a system failure. Roll-back recovery and ILR are incompatible with AUTODEFER, therefore output deferred mode should be used only in a batch situation where the data base has been backed-up prior to batch processing. Roll-back recovery and ILR must be disabled prior to enabling AUTODEFER.
DUMPING	for development and debugging only. When enabled, the TurboIMAGE abort procedure copies the user's stack and the data base control block to a file if a TurboIMAGE procedure aborts.
ILR	enables Intrinsic Level Recovery facility.
LOGGING	enables the data base roll-forward logging facility.
RECOVERY	enables the data base roll-forward recovery facility.
ROLLBACK	enables the data base roll-back logging facility and automatically enables ILR.

# DBUTIL

## >>ENABLE

### Default Conditions

Access is Enabled  
Autodefer is Disabled  
Dumping is Disabled  
ILR is Disabled  
Logging is Disabled  
Recovery is Disabled  
Roll-Back is Disabled

### Example

```
:RUN DBUTIL.PUB.SYS
.
.
>>ENABLE ORDERS FOR RECOVERY
    Recovery is Enabled
>>
```

Reinitializes all data sets in the data base to their empty condition.

The data sets remain as catalogued MPE files. To execute DBUTIL to reinitialize the data sets you must be the data base creator or supply the correct maintenance word. This utility function should be performed before data saved by DBLOAD is loaded back into the data base unless it was recreated.

After DBUTIL has completely reinitialized the data sets, it prints a confirmation message on the listfile device.

## Syntax

```
>>ERA[SE] data base name [/maintenance word]
```

For example:

```
ERASE ORDERS/SELL
```

Where ORDERS=*data base name* / SELL=*maintenance word*

## Parameters

*data base name* is the name of a TurboIMAGE data base being erased.

*maintenance word* is the maintenance word defined by the data base creator when the data base is created with DBUTIL. This word must be supplied by anyone other than the data base creator.

## Example

```
:RUN DBUTIL.PUB.SYS
```

Initiate DBUTIL execution

```
.
```

```
>>ERASE ORDERS/SELL
```

Enter >>ERASE command, data base name, and maintenance word.

```
Data base ORDERS has been ERASED
```

```
>>
```

DBUTIL reinitializes the data set files ORDERS01, ORDERS02, and so forth to their original empty, zeroed condition. In addition, the data base flags are reset to their default conditions. Logging is disabled if it was previously enabled.

# DBUTIL

## >>EXIT

Terminates DBUTIL execution.

## Syntax

>>EXI[T]

## Example

```
>>CREATE ORDERS
      Data base ORDERS has been CREATED
>>EXIT

END OF PROGRAM
```

Create a data base

If no other DBUTIL functions  
are to be performed, terminate,  
DBUTIL with >>EXIT command.

Displays each of the DBUTIL commands.

## Syntax

```
>>HEL[P] [commandname]
```

## Parameter

*commandname* is the name of a specific DBUTIL command whose format you want to display. The name may be abbreviated to the first three letters.

If you do not specify a *commandname*, the >>HELP command lists the names of all valid DBUTIL commands.

If you specify a *commandname*, the correct syntax for that command is displayed.

## Example

```
>>HELP  
Commands are:
```

HELP	CREATE	ERASE	PURGE	DEACTIVATE
ACTIVATE	VERIFY	SET	ENABLE	DISABLE
SHOW	EXIT	RELEASE	SECURE	TRACE
MOVE				

Commands may be abbreviated to the first three letters.  
For help on a particular command type: 'HELP command name'.

```
>>HELP CREATE
```

```
CRE[ATE] data base name [/maintenance word]
```

```
>>
```

For more information on TurboIMAGE Tracing refer to the *TurboIMAGE Profiler User Guide*.

# DBUTIL

## >>MOVE

Moves TurboIMAGE files across devices.

## Syntax

```
>>MOVE TurboIMAGE file name TO device
```

For example:

```
MOVE ORDERS05 to DISC2
```

Where ORDERS05=*file name* and DISC2=*device*

## Parameters

*file name* is a TurboIMAGE root, data set, or ILR file name. (Enter the file name only, no group/account specification is allowed.) The user must be the creator of the file.

*device* is the device class name or logical device number to which the TurboIMAGE file should be moved.

## Example

```
>>MOVE ORDERS06 to 3
Data base last stored on TUE, DEC 4, 1984, 8:32 PM
Data base has been modified since last store date.
The data base should be backed up before doing a MOVE operation.
Do you still want to continue the MOVE operation (Y/N)? Y
Starting file copy ...
... file copy completed.
Purging old copy of file "ORDERS06"
New copy of file "ORDERS06" saved as a permanent file
File "ORDERS06" moved to device 3
```

The data set file ORDERS06 has been moved to logical device number 3. This file contains the INVENTORY data set and was originally assigned device class name DISC1 in the schema. A DBUTIL >>SHOW ORDERS DEVICE can be performed to obtain a listing of where all the data set files for the data base reside.

It is recommended to do a DBSTORE of the data base prior to performing a MOVE. This precaution is advisable in the event of a system failure occurring during the MOVE operation. When a MOVE has been initialized the process will check the root file flag to determine if the data base has been modified since the last backup copy was made. The program will prompt the user to continue or to terminate the MOVE command and proceed with a DBSTORE of the data base before moving TurboIMAGE files to another device. If the user responds "NO" to the continue prompt, the message "MOVE operation stopped." will be printed on the terminal.

The following steps outline the process involved in moving TurboIMAGE files from one device to another. The MOVE process:

1. retrieves information from the "old" file. ("Old" indicates the file on the originally specified device.)
2. checks the device specified by the user for validity and existence, and determines if there is sufficient space for the "new" file. ("New" indicates the file being moved to another device.)
3. checks the root file to determine the data base state.
4. copies the old file to the new file.
5. sets the flag in the root file.
6. purges the old file, then saves the new file.
7. resets the flag in the root file.



# DBUTIL

## >>PURGE

Purges the root file and all the data sets of the referenced data base.

Purging removes the files from the catalog and returns the disc space to the system. As with >>ERASE, you must be the data base creator or must provide the maintenance word to use DBUTIL with the >>PURGE entry. Before running the DBRESTOR program to restore a data base, you should use this utility function to purge the data base.

If DBUTIL successfully purges the data base, it prints a confirmation message on the listfile device.

## Syntax

```
>>PUR[GE] data base name [/maintenance word]
```

For example:

```
PURGE ORDERS/SELL
```

Where ORDERS=*data base name* and SELL=*maintenance word*

## Parameters

*data base name* is the name of a TurboIMAGE data base being purged.

*maintenance word* is the maintenance word defined by the data base creator when the data base is created with DBUTIL. This word must be supplied by anyone other than the data base creator.

## UNEXPECTED RESULTS

The following messages are printed if an unexpected situation occurs: Refer to Appendix A for other error messages.

MESSAGE	MEANING
No root file, >>PURGE operation proceeding	DBUTIL was unable to locate the root file, but will attempt to purge any data set files.
Data set XXXXk has been purged	DBUTIL successfully purged the root file and the $n$ data sets of the data base. However, DBUTIL also discovered and purged an unexpected data set named XXXXk where $k$ is a number greater than the number of data sets defined for the data base ( $n$ ).
Data set XXXXk is missing	DBUTIL successfully purged the root file and all existing data sets but data set XXXXk is unexpectedly missing. In this case $k$ is less than the number of data sets defined for the data base.
Data base >>PURGE is not complete	A fatal error occurred while DBUTIL was attempting to purge the data base. The fatal error message is printed above this one. Some of the data sets have been purged.

## Example

```
:RUN DBUTIL.PUB.SYS
```

Initiate DBUTIL execution.

```
.
```

```
>>PURGE ORDERS
```

Enter >>PURGE command and data base name

```
Data base ORDERS has been PURGED
```

assuming there is no maintenance word).

DBUTIL confirms that the user is logged on with the same user name, account, and group which were used to create the data base. It then determines whether the root file exists and if so, purges the root file and any files named ORDERS01, ORDERS02, and so forth. Even if the root file does not exist, any data set files with names based on the root file name are purged.

# DBUTIL

## >>RELEASE

Suspends file system security provisions for the data base root file and data set files, allowing access to the data base from other groups and accounts.

### Syntax

```
>>REL[EASE] data base name
```

### Parameter

*data base name* is the name of a TurboIMAGE data base being released.

>>RELEASE suspends file system security provisions for all of the data base files at the file, group, and account levels, but leaves TurboIMAGE security intact. Releasing the file system security allows the data base to be accessed by users from other groups and accounts, without relinquishing the privacy of all other files in the data base group. Only the creator of the data base can release security. In addition, the group's home volume set must be mounted.

The data base file security remains suspended until the creator issues a >>SECURE command. Suspension remains valid after job termination, or system failure followed by coldload or reload.

Restores security provisions that were released by a >>RELEASE command for the data base root file and data set files.

## Syntax

>>SEC[URE] *data base name*

For example:

SECURE ORDERS

Where ORDERS=*data base name*

## Parameter

*data base name* is the name of a TurboIMAGE data base being secured.

The SECURE command reinstates the file system security provisions for the entire data base. (These security provisions can only be suspended by the >>RELEASE command.) Only the creator of the data base can successfully issue this command. In addition, the group's home volume set must be mounted.

# DBUTIL

## >>SET

Changes or removes the maintenance word or specifies the number of input/output buffers to be allocated by TurboIMAGE in the DBB depending on the number of users concurrently accessing the data base. Only the data base creator can change or remove the maintenance word. Also sets the log identifier into the root file, or modifies access class passwords, or sets a subsystem flag.

## Syntax

```
>>SET data base name
      [/maint word] {
                     MAINT=maintenance word
                     BUFFSPECS=num buffers(from-users/to-users)
                               [,num buffers(from-users/to-users)]...
                     LOGID=log identifier
                     PASSWORD classnum=[password]
                     SUBSYSTEMS={
                                NONE
                                READ
                                RW
                                }
                     LANGUAGE=language id
                     }
```

For example:

```
>>SET ORDERS MAINT=SELL
```

Where ORDERS=*data base name* and SELL=*maintenance word*

or

```
>>SET ORDERS/SALE BUFFSPECS=4(1/5),7(6/10),9(11/15),10(16/20)
```

Where 4=*num buffers* (1=*from-users* / 5=*to-users*)

## Parameters

*data base name* is the name of a TurboIMAGE data base root file catalogued in the current session or job's account and log on group.

*maint word* is the current maintenance word for the data base, and must be supplied by anyone other than the data base creator.

*maintenance word* is the new maintenance word for the data base. If omitted, the currently defined maintenance word is removed and the data base has no maintenance word. Only the data base creator can change or remove the maintenance word.

<i>num buffers</i>	is the number of buffers to be allocated by TurboIMAGE in the DBB for the range of users specified between the parentheses that follow. The minimum number of buffers allowed is 4 and the maximum is 255. To increase performance with the >>SET statement, as a starting point the user may want to specify that between 15 and 25 buffers be allocated in the Data Base Buffer Area Control Block (DBB). The number of buffers allocated and the actual amount of performance increase obtained depends on whether ILR is enabled, how many users are accessing the data base, amount of main memory available and many other factors.
<i>from-users</i>	is the minimum number of concurrent users (access paths) for which the preceding <i>num buffers</i> should be allocated. The minimum <i>from-users</i> value allowed is 1 and the maximum is 120. The value must be less than the immediately following <i>to-users</i> value.
<i>to-users</i>	is the maximum number of concurrent users for which the preceding <i>num buffers</i> should be allocated. The minimum <i>to-users</i> value allowed is 1 and the maximum 120. The value must be greater than the immediately preceding <i>from-users</i> value.
<i>language id</i>	is the native language assignment for the data base. This command can be issued only on a virgin root file or an empty data base. The message "Language changed" appears after using the SET command to change the language id. Refer to the <i>NLS Reference Manual</i> for name and number information.
<i>log identifier</i>	is an MPE log identifier obtained using the MPE :GETLOG command. DBUTIL first checks to insure that the log identifier is valid, and then prompts for a password. Entry of the correct password will cause the log identifier to be stored in the root file and used whenever the logging capability is enabled.
<i>classnum</i>	is the access class whose password is being changed. A number from 1 to 63.
<i>password</i>	is the new password being assigned to a particular access class. Up to 8 characters are allowed. If omitted, any password previously assigned to that class is removed. (You must be the data base creator.)
NONE	is the option used to prohibit use of any sub-system (for example QUERY) on TurboIMAGE.
READ	is the option that allows only read access to the data base by sub-systems. The sub-system checks the root file flag to determine what access a sub-system is allowed.
RW	is the option that allows read/write access to the data base by sub-systems. The sub-system checks the root file flag to determine what access a sub-system is allowed.

# DBUTIL

## >>SET

The *from-users/to-users* ranges must be specified in increasing order. The ranges may not overlap but they need not be consecutive. If *num buffers* is not specified for a particular number of users, the default number of buffers is used. These are the default settings assigned by TurboIMAGE:

b(1/2)	b+4( 9/10)	b+7(15/16)
b+1(3/4)	b+5(11/12)	b+8(17/18)
b+2(5/6)	b+6(13/14)	b+9(19/120)
b+3(7/8)		

The value of b is equal to either: 1) the largest number of search items in any detail data set in the data base plus 3, or 2) whichever is larger. If p is the maximum number of search items (the path count), the value of b can be represented as  $b = \max(p+3, 8)$ .

For example, the largest path count for a detail data set in the ORDERS data base is 4. (This is the path count for the CUST data set.) Therefore, the value of b for the ORDERS data base is:  $b = \max(4+3, 8) = 8$ .

The default buffer specifications in this case are:

8(1/2),9(3/4),10(5/6),11(7/8),12(9/10),13(11/12),14(13/14),15(15/16),16(17/18),17(19/120).

## Example

```
:RUN DBUTIL.PUB.SYS
```

Initiate DBUTIL execution.

```
>>SET ORDERS MAINT=
```

Maintenance word changed

Remove current maintenance word.

```
>>
```

```
:RUN DBUTIL.PUB.SYS
```

```
>>SET ORDERS BUFFSPECS=5(1/120)
```

For data base ORDERS

BUFFER SPECIFICATIONS:

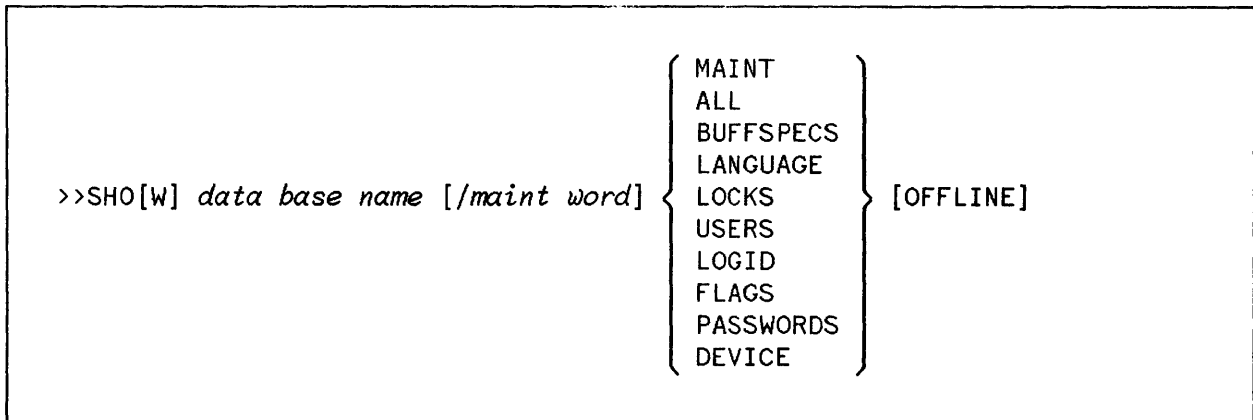
5(1/120)

```
>>
```

Specify 5 buffers to be allocated for  
from 1 to 120 users (access paths).  
DBUTIL confirms the specifications by  
listing them.

Displays information about the data base on a terminal or line printer. The information may include a list of processes that have the data base open, the status of locks in the data base, the log identifier and flags, and the current buffer specifications. This command should be used with care for data base maintenance functions since it obtains exclusive control of the data base for several seconds preventing all other access.

### Syntax



For example:

```
SHOW ORDERS/SELL ALL OFFLINE
```

Where ORDERS=*data base name* / SELL=*maintenance word*

### Parameters

<i>data base name</i>	is the name of a TurboIMAGE data base root file catalogued in the current session or job's account and log on group.
<i>maint word</i>	is the current maintenance word for the data base. It must be supplied by anyone other than the data base creator.
MAINT	displays the maintenance word, if any.
ALL	displays all the information provided with MAINT, BUFFSPECS, LOCKS, USERS, LOGID, and FLAGS.
BUFFSPECS	displays the current buffer specifications which may either be the TurboIMAGE default setting or the values specified with the DBUTIL >>SET command.
FLAGS	displays the state (enabled or disabled) of the logging, roll-back, ILR, recovery, restart, subsystem access, autodefer, access and dumping options.
LANGUAGE	displays state of the native language declaration for the data base. The language attribute will have been specified either at schema processing time or through the >>SET command. Refer to the <i>NLS/3000 Reference Manual</i> for more information.



# DBUTIL

## >>SHOW

LOCKS	displays the status of locks currently obtained (or requested).
LOGID	displays the current MPE log identifier for the data base.
USERS	displays a list of the processes that have the data base open with the program file name and other information. (Refer to examples below.)
OFFLINE	requests that the information be listed on the line printer. The formal designator for the list file is DBUTLIST. (Passwords and maintenance word will not be printed.)
PASSWORDS	displays the access class numbers from 1 to 63 together with the passwords assigned to them. (You must be the data base creator.)
DEVICE	displays the TurboIMAGE root, ILR, and data set files and where files reside (device class name or logical number) for a data base.

The >>SHOW commands may be executed at any time except when another process has the data base opened in an exclusive access mode (mode 3 or 7).

### Example (Show Users)

```
:RUN DBUTIL.PUB.SYS
```

```
>>SHOW ORDERS USERS
```

1.....	2.....	3.....	4.....	5
PIN	PATH	EXECUTING PROGRAM	JOBNUM	MODE
21	1	INVENTORY.IMAGE.DATAMGT	#S116	1
22	1	BROWSE.IMAGE.DATAMGT	#S118	5
28	1	BROWSE.IMAGE.DATAMGT	#S112	5
29	1	INVENTORY.IMAGE.DATAMGT	#S115	1
31	1	ORDENTRY.IMAGE.DATAMGT	#S117	1

The columns of information are as follows:

- 1 The Process Identification Number (PIN). This is a number assigned to a process by the operating system when the process is created. The table above indicates that the process has opened the data base ORDERS.
- 2 The access path number. The access paths for each process are numbered consecutively beginning with 1. (Refer to the discussion of access paths in Section 4.)
- 3 The name of the program file, its group and account.
- 4 The number of the job or session in which the process is running.
- 5 The access mode in which the data base is open.

Note that DBUTIL does not call DBOPEN and therefore it is not listed as an executing program.

**Example (Show All)**

```
>>SHOW ORDERS ALL          Display all information for ORDERS
    For data base ORDERS    data base.

MAINTENANCE WORD:  SELL

Access is enabled.
Autodefer is disabled.
Dumping is disabled.
Rollback recovery is disabled.
Recovery is enabled.
ILR is disabled.
Logging is enabled.
Data base last stored on MON, DEC 3, 1984, 1:09 PM
Data base has not been modified since last store date.
Restart is disabled.
Subsystem access is READ/WRITE.

LOGID:  ORDERLOG is valid
        Password is correct

The language is 0:NATIVE-3000

BUFFER SPECIFICATIONS:
5(1/120)

No other users accessing the data base
>>
```

The listing above indicates that the current buffer specifications provide for 5 buffers to be allocated when there are between 1 and 120 concurrent users of the data base. The list above also shows that the data base is enabled for roll-forward recovery, logging, and user access. In this example, the Restart flag is disabled. This flag is set by DBRECOV when the user has requested to suspend recovery between log files. The logid is shown and the password is verified and the maintenance word is displayed. The messages which appear during the SHOW command may vary depending on what information is available on the data base. If the maintenance word and logid are not present the following appears:

Logid is not present.

Maintenance word is not present.

The message regarding Native Language Support will also vary. The following message will be printed if the data base does not support NLS:

This data base has been created before support of Native Languages.

# DBUTIL

## >>SHOW

### FORMAT OF SHOW DEVICE LIST

The following list shows the TurboIMAGE files for data base ORDERS, along with the data set names and the device where each resides. In the following example the root file ORDERS, ILR file ORDERS00, and the data set files are shown. The file devices are listed as specified, device class names, ORDERS06 data set was moved using the DBUTIL >>MOVE command to logical device number 3.

### Example (Show Device)

```
>>SHOW ORDERS DEVICE
  For data base ORDERS
```

MPE file Name	Data Set Name	Device
ORDERS.IMAGE.DATAMGT		DISC
ORDERS00.IMAGE.DATAMGT		DISC
ORDERS01.IMAGE.DATAMGT	Customer	DISC
ORDERS02.IMAGE.DATAMGT	Date-Master	DISC1
ORDERS03.IMAGE.DATAMGT	Product	DISC1
ORDERS04.IMAGE.DATAMGT	Sales	DISC1
ORDERS05.IMAGE.DATAMGT	Sup-Master	DISC1
ORDERS06.IMAGE.DATAMGT	Inventory	3

```
>>
```

### FORMAT OF SHOW LOCKS LIST

DBUTIL lists the locking information sequentially by locking level: data base locks followed by data set locks, followed by data entry locks. The names of locked entities (for example, the data base, data set, or lock descriptor for data entries) appear in upper case followed by a list of other processes waiting at that locking level. DBUTIL indicates in lower case the reason each process is waiting. This message is preceded by a hyphen so that it can be identified on terminals or listings from a line printer without lower case.

If the term (PENDING) appears after a locked entity, it indicates that the lock has been obtained but control cannot be returned to the caller until other locks have been released. The same process identification will appear elsewhere in the list together with an explanation of why it is waiting.

Infrequently, the term (TEMP) may appear. TurboIMAGE places a temporary lock on a data set while it processes an existing data entry lock request. Temporary locks occur only when a user requests data entry locks on different items. Whenever the lock item changes, TurboIMAGE must wait until all existing locks on the current lock item are cleared before it places a lock on the new lock item. During the wait the lock is termed "TEMP". These locks are held very briefly and only under rare circumstances. The Process Identification Numbers (PINs) and job/session numbers listed are the same as those shown by the MPE commands such as :SHOWJOB and :SHOWQ.

**Example (Show Locks)**

>>SHOW ORDERS LOCKS OFFLINE      List the status of locks requested  
and held in the ORDERS data base on  
the line printer.

The line printer listing looks like this:

HP32215C.00.00 TurboIMAGE/3000: DBUTIL MON, DEC 10, 1984, 5:06 PM

For data base ORDERS

	LOCKED ENTITY - ( - waiting process)	PIN/ PATH	PROGRAM NAME	JOBNUM
1	DATA SET SALES.....	30/1	BROWSE	#S126
2	-waiting for data set unlock: .....	17	INVENTORY	#S128
	-waiting for data set unlock: .....	32	ORDENTRY	#S129
	-waiting for data set unlock: .....	21	ORDENTRY	#S118
3	DATA SET CUSTOMER.....	30/1	BROWSE	#S126
	DATA SET INVENTORY.....	30/1	BROWSE	#S126

- 1 Indicates process 30 (program BROWSE executing in session 126) has the SALES data set locked through access path 1.
- 2 Shows a queue of processes waiting for the SALES data set to unlock. For example, in the first line, process 17 (program INVENTORY executing in session 128) is waiting. Since it is listed first in the queue, it will be the next process to resume execution after the SALES data set is unlocked. It may be waiting to place a lock on the data set or entries in the set.
- 3 Indicates process 30 (program BROWSE, session 126, access path 1) has the CUSTOMER data set locked. No processes are waiting for the lock to be released.

# DBUTIL

## >>SHOW

Here is another example of a locking list that might appear when the >>SHOW LOCKS command is entered.

### Example (Show Locks)

HP32215C.00.00 TurboIMAGE/3000: DBUTIL MON, DEC 10, 1984, 4:20 PM

For data base ORDERS

	LOCKED ENTITY / ( - waiting process)	PIN/ PATH	PROGRAM NAME	JOBNUM
1	DATABASE (PENDING) .....	22	BROWSE	#S118
	-waiting for zero locks within database:...	22	BROWSE	#S118
2	DATA SET INVENTORY.....	29/1	INVENTORY	#S115
3	ORDERS: QUANTITY<= 50 .....	28/1	BROWSE	#S112
4	CUSTOMER: CUST-NAME = DON'S MERCANTILE...	31/1	ORDENTRY	#S117

- 1 Indicates process 22 (program BROWSE, session 118) has obtained a lock on the data base and yet it cannot continue until existing locks held in the data base are released. In this example, the reason for the pending lock is listed on the line below.
- 2 Indicates process 29 (program INVENTORY, session 115, access path 1) has the INVENTORY data set locked.
- 3 Indicates that process 28 (program BROWSE, session 112, access path 1) has all entries in the SALES data set with QUANTITY less than or equal to 50 locked.
- 4 Indicates process 31 (program ORDENTRY, session 117, access path 1) has all entries in the CUSTOMER data set with LAST-NAME equal to DON'S MERCANTILE locked.

Note that all subsequent requests for locks must be made to wait until process 22 releases its data base lock.

Reports whether a data-base-access (DBA) file is activated or deactivated and checks the validity of the DBA file..

## Syntax

>>VER[IFY] *data-base-access file name*

For example:

VERIFY ORDDBA

Where ORDDBA=*data-base-access file name*

## Parameter

*data-base-access file name* is the name of a data-base-access file.

## Example

:RUN DBUTIL.PUB.SYS

Initiate DBUTIL execution.

.

>>VERIFY ORDDBA

Enter >>VERIFY command and data-base access file name.

Data-base-access file

ORDDBA is ACTIVATED

>>

# USING A REMOTE DATA BASE

SECTION

9

If you want to access a data base that resides on one HP 3000 computer system while operating a session on another HP 3000 computer system, you may do so provided both systems are configured with Distributed Systems (DS/3000) capability. Access can also be established between a TurboIMAGE data base and an IMAGE/3000 data base using the remote data-base-access methods described in this section. If you are not familiar with DS/3000 you should read the *DS/3000 Reference Manual* before you begin accessing a remote data base.

DS/3000 recognizes the computer to which your terminal is directly connected as the local HP 3000 and the computer with which you establish a communications link as the remote HP 3000. The session that you initiate on the local HP 3000 is a local session and a session on a remote HP 3000 is a remote session.

You can access an IMAGE/3000 data base from TurboIMAGE. Access from an IMAGE/3000 data base to a TurboIMAGE data base is allowed also. However, if you have expanded the limits on your TurboIMAGE data base (for example exanded to 199 data sets in your data base), remote access from an IMAGE/3000 data base is not allowed. This is due to the size of the Remote Data Base Control Block in IMAGE/3000. (Refer to "Remote Data Base Access" in Section 10 for more information.)

You may use a data base on a remote HP 3000 either from a program that is running on the remote system or from a program running on your local HP 3000. There are various ways to open a communications line and initiate a remote session. For example, you can establish a communications link and remote session and then run a remote program accessing a data base on the remote machine as illustrated in Figure 9-1.

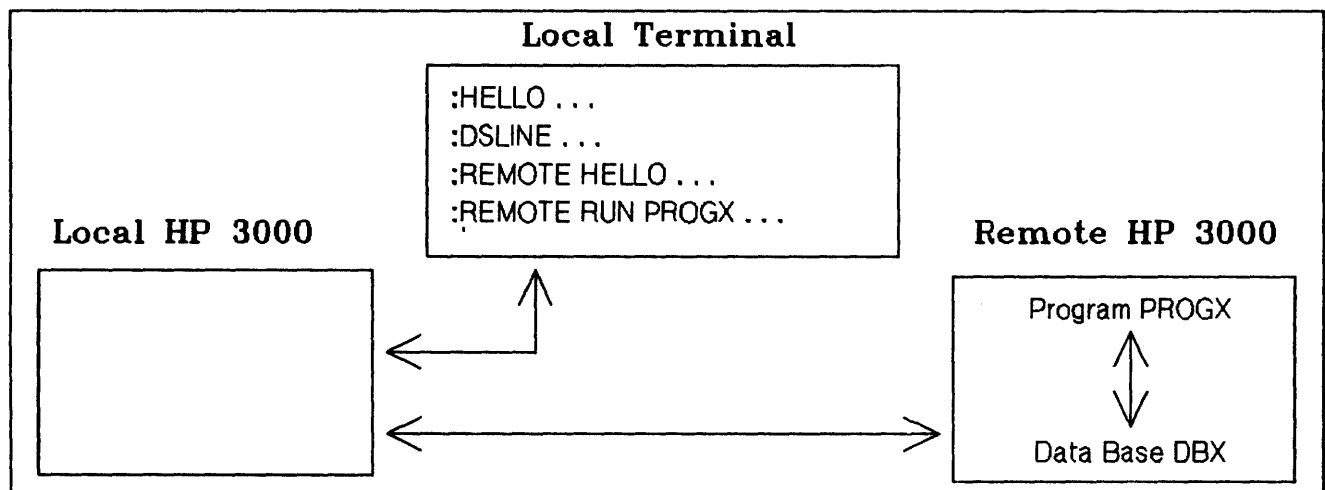


Figure 9-1. Using a Remote Program

Refer to Sections 1 through 3 of the *DS/3000 Reference Manual* for a detailed description of this method.

## ACCESS THROUGH A LOCAL APPLICATION PROGRAM

If you want to access a remote data base using a local application program, there are three methods you may choose from. In all cases, a local program accesses a remote data base and the data is passed across the communication line.

### Method 1

#### ESTABLISHING COMMUNICATIONS LINK AND REMOTE SESSION INTERACTIVELY

To use the first method, you interactively establish a communications link and a remote session and enter a FILE equation for each remote data base. You can access more than one data base by issuing multiple FIND commands. The FILE equation specifies which data base is to be accessed on which remote system and device. A local application program can now access a remotely located data base, as shown in Figure 9-2.

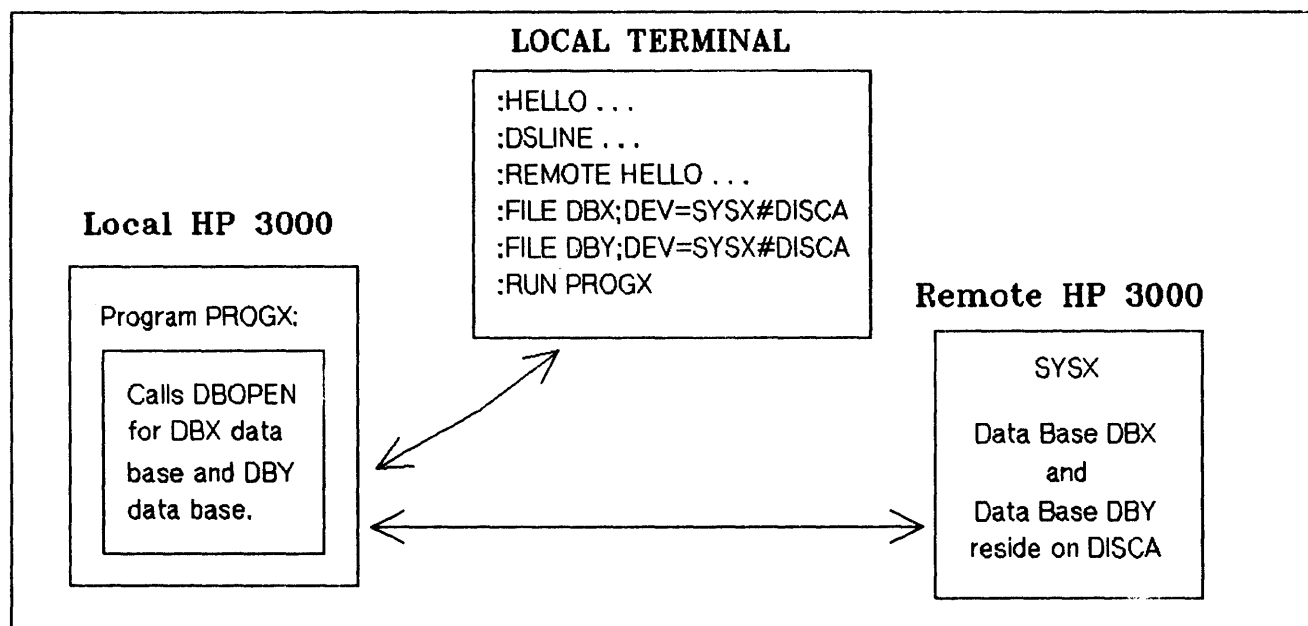


Figure 9-2. Using Method 1

For details about using this method refer to the *DS/3000 Reference Manual*.



## Method 2

### USING THE COMMAND INTRINSIC

The second method is very similar to the first, but you use the MPE COMMAND intrinsic within your application program to establish the communications link, remote session and remote data base access. In order to use this method in an application program which is coded in COBOL or RPG you must write a procedure in SPL or FORTRAN and call the procedure.

To use this method you must issue a REMOTE HELLO command (either with the DSLINE parameter or issue the DSLINE as a separate command) and a FILE equation by calling the COMMAND intrinsic for each of these commands. Use of the COMMAND intrinsic is explained in the *MPE Intrinsics Reference Manual*, and information about accessing remote files is given in the *DS/3000 Reference Manual*. Figure 9-3 contains a diagram of Method 2.

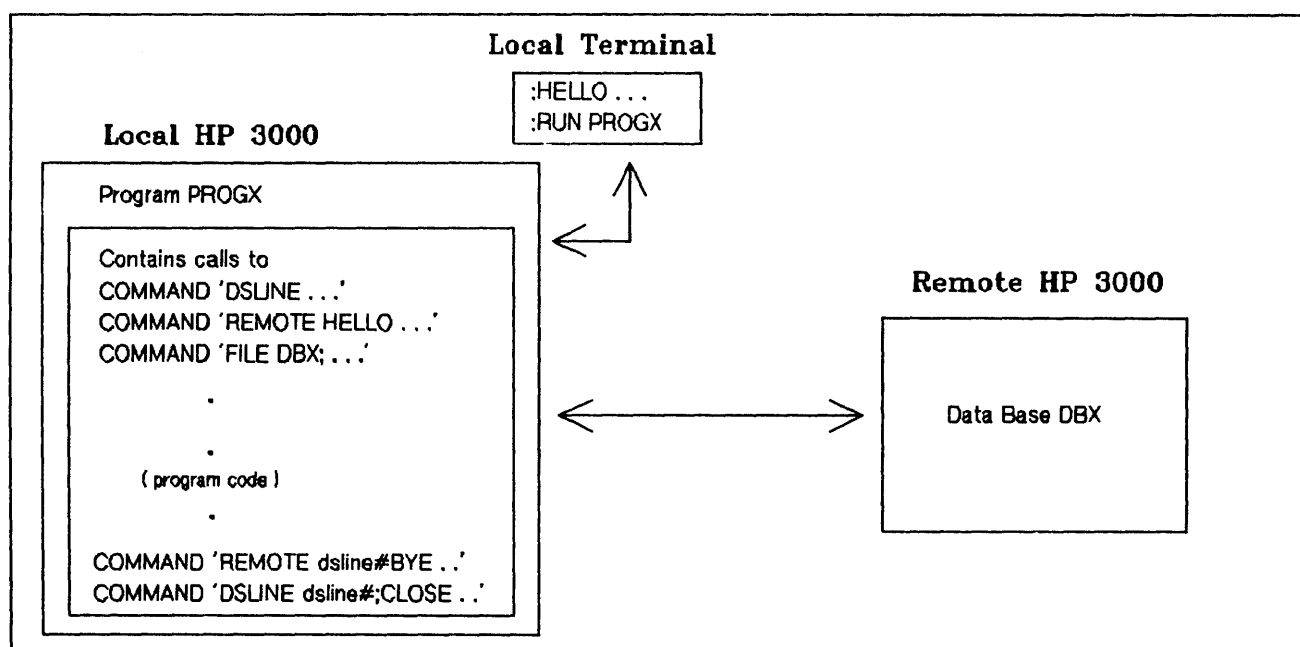


Figure 9-3. Using Method 2

If you want to access more than one remotely located data base with an application program, you must enter one FILE equation for each remote data base. It is important to remember that a REMOTE HELLO to the same remote computer should not be repeated within a process since the second request for a remote session would log off the first one.

When the application program calls the DBCLOSE procedure or is ready to terminate execution, it must programmatically issue REMOTE BYE and DSLIN commands for the communications line specified with the foregoing COMMAND intrinsic.

If you use this method, any change in the data base name, account or password information requires modification of the application program. Since the application program maintains logical control over the commands that are issued it is responsible for checking all status words returned by the remote system.

## Method 3

### USING A DATA-BASE-ACCESS FILE

The third method involves creating a special file which we shall call the data-base-access file (DBA file). This file provides TurboIMAGE with the necessary information to establish a communications link and a remote session. It also specifies the remote data base or data-base-access file name so that the necessary TurboIMAGE intrinsics can be executed on the remote computer.

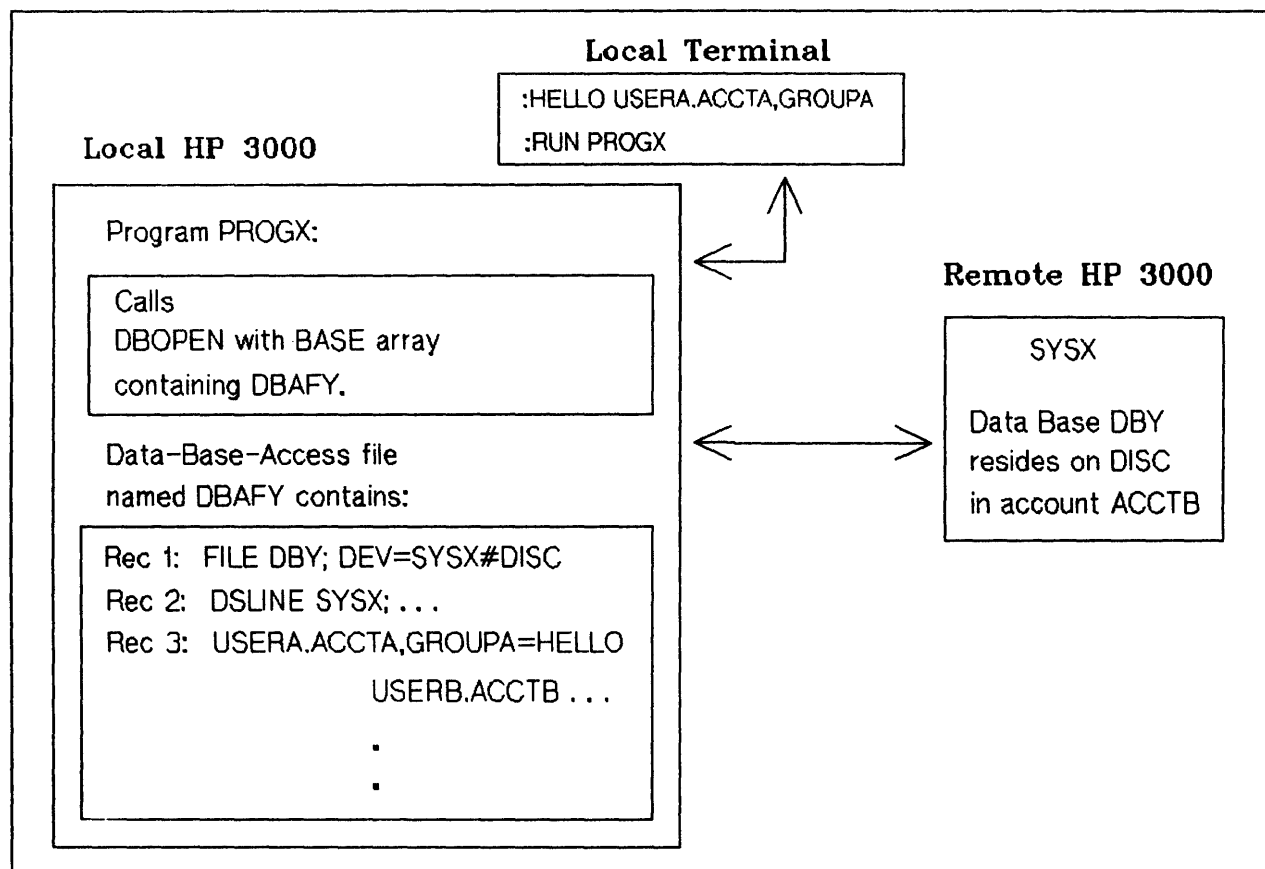


Figure 9-4. Using Method 3

It should be noted that with Method 3, which uses the data-base-access file only one data base can be accessed using each data-base-access file per DSLINE. For example, if two computers are linked through two DSLINES, you can open one data base on each line.

When the user or an application program calls DBOPEN with the data-base-access file name, the remote session is established and the remote data base is opened. Then other TurboIMAGE intrinsics can perform desired operations on the data base. Under this method the remote session is automatically released when the data base is closed (with or without an explicit DBCLOSE call).

A second REMOTE HELLO on one DSLINE terminates the previous REMOTE HELLO. For multiple remote data base access, Method 1 or Method 2 is recommended. If the data-base-access file is used, an automatic REMOTE BYE and DSLINE commands are issued on the communications line specified in the data-base-access file when the application program terminates execution.

By using this approach, the data base administrator can set up a user-table which provides more control over the data base users, and thus, enhances data base security. To create the data-base-access file you may use the Editor (EDIT/3000). First use the :SET LENGTH command to accommodate the largest record to be included in the data-base-access file, if the record exceeds the default length specified in the *EDIT/3000 Reference Manual*. The length must be less than or equal to 128 characters. The content of this file should be created in the format shown below.

## Syntax

```
Record 1  FILE dbname1[=dbname2];DEV=dsdevice#[DISC]

Record 2  DSLINE dsdevice[;LINEBUF=buffer-size][;LOCID=local-id-sequence]
          [;REMID=remote-id-sequence][;PHNUM=telephone-number]
          [;EXCLUSIVE][;QUIET]

Record 3  lusername.lacctname[,lgrouppname]=HELLO rusername[/rupasw]
          .racctname[/rapasw][,rgrouppname[/rgpasw]] [;TIME=cpusecs]
          [;PRI=priority] [;HIPRI
                        [;INPRI=inputpriority] ]

Record 4 through Record n - Same format as Record 3.
Specifies other "user.account,group" identification.
```

## Parameters

<i>dbname1</i>	the name of the data base or the data-base-access file on the remote system you want to access, or is the formal file designator used in the program if <i>dbname2</i> is specified. (Required parameter)
<i>dbname2</i>	is the name of the data base or the data-base-access file on the remote system you want to access. (Optional parameter)
<i>dsdevice</i>	is the device class name or logical device number assigned to the DS/3000 communication driver IODS0 during system configuration. (Required parameter)
<i>buffer-size</i>	is a decimal integer specifying the size (in words) of the DS/3000 line buffer to be used in conjunction with the communication line. The integer must be within the range 304 < <i>buffer-size</i> <8192. The default value is the buffer size entered in response to the PREFERRED BUFFER SIZE prompt during system configuration. (Optional parameter)

- local-id-sequence* is a string of ASCII characters contained within quotation marks or a string of octal numbers separated by commas and contained within parentheses. If you wish to use a quotation mark within an ASCII string, use two successive quotation marks.
- In the case of an octal sequence, each octal number represents one byte and must be within the range 0-377. The maximum number of ASCII characters or octal numbers allowed in the string is 16.
- The supplied string of ASCII characters or octal numbers define the ID sequence that will be sent from your HP 3000 to the remote HP 3000 when you attempt to establish the telephone connection. If the remote HP 3000 does not recognize the supplied ID sequence as a valid one, the telephone connection is terminated. The default value is the ASCII or octal string entered in response to the LOCAL ID SEQUENCE prompt during system configuration. (optional parameter)
- remote-id-sequence* same format as *local-id-sequence*. The supplied string of ASCII characters or octal numbers define those remote HP 3000 ID sequences that will be considered valid when you attempt to establish the telephone connection. If the remote HP 3000 does not send a valid ID sequence, the telephone connection is terminated. The default set of remote ID sequences consists of the ASCII and octal strings entered in response to the REMOTE ID SEQUENCE prompt during system configuration. (Optional parameter)
- telephone-number* is a telephone number consisting of digits and dashes. The maximum length permitted (including both digits and dashes) is 20 characters. If YES was entered in response to the DIAL FACILITY prompt during system configuration, this telephone number will be displayed at the operator's console of your HP 3000 and the operator will then establish the telephone connection by dialing that number at the MODEM. The default telephone number is the first one entered in response to the PHONE NUMBER prompt during system configuration. (Optional parameter)
- EXCLUSIVE specifies that you want exclusive use of the particular communications line. If the specified HSI or SSLC is already open and you have specified the exclusive option, DS/3000 will deny you access to the line (you cannot open it). Opening an EXCLUSIVE line requires the user to have CS capability. This capability may be granted by a system manager or account manager. (Optional parameter)
- QUIET specifies that the message identifying the DS line number will be suppressed. The messages associated with subsequent REMOTE HELLO and REMOTE BYE commands will also be suppressed. In this case, the terminal operator is totally unaware that remote processing is taking place. (Optional parameter)
- lusername* is a user name on the local HP 3000, as established by an account manager, that allows you to log-on under this account. This name is unique within the account. It contains from 1 to 8 alphanumeric characters, beginning with a letter. An at-sign (@) may be used to indicate the log-on user name. (Required parameter)

<i>lacctname</i>	is the name of your account on the local HP 3000 as established by a system manager. It contains 1 to 8 alphanumeric characters, beginning with a letter. An at-sign (@) may be used to indicate the log-on account. (Required parameter)
<i>lgrouppname</i>	is the name of a file group to be used for the local file domain and central processor time charges, as established by an account manager. It contains from 1 to 8 alphanumeric characters, beginning with a letter. An at-sign (@) may be used to indicate the log-on group. (Optional parameter)
<i>rusername</i>	is a user name on the remote HP 3000 that allows you to log on under the remote account. It follows the same rules as <i>username</i> . An at-sign (@) may be used to indicate <i>rusername</i> as with <i>lususername</i> . (Required parameter)
<i>rupasw</i>	is the password assigned to <i>username</i> . (Optional parameter)
<i>racctname</i>	is the name of the log on account on the remote HP 3000. It follows the same rules as <i>lacctname</i> . An at-sign (@) may be used to indicate <i>racctname</i> is the same as <i>lacctname</i> . (Required parameter)
<i>rapasw</i>	is the password assigned to <i>racctname</i> . (Optional parameter)
<i>rgrouppname</i>	is the name of the log on group on the remote HP 3000. It follows the same rules as <i>lgrouppname</i> . An at-sign (@) may be used to indicate <i>rgrouppname</i> is same as <i>lgrouppname</i> . (Optional parameter)
<i>rgpasw</i>	is the password assigned to <i>rgrouppname</i> . (Optional parameter)
TIME= <i>cpusecs</i>	is the maximum central processor time that your remote session can use, entered in seconds. When this limit is reached, the remote session is aborted. It must be a value from 1 to 32767. To specify no limit, enter a question mark or omit this parameter. Default: No limit. (Optional parameter)
PRI= <i>priority</i>	is the execution priority class that the Command Interpreter uses for your remote session, and also the default priority for all programs executed within the remote session. BS is highest priority; ES is lowest. If you specify a priority that exceeds the highest that the system permits for <i>racctname</i> or <i>rusername</i> , MPE assigns the highest priority possible below BS. Default: CS. NOTE: DS and ES are intended primarily for batch jobs; their use for sessions is generally discouraged.
INPRI= <i>inputpriority</i>	is the relative input priority used in checking against access restrictions imposed by the job fence, if one exists. It takes effect at log-on time. It must be a value from 1 (lowest) to 13 (highest priority). If you supply a value less than or equal to the current job fence set by the console operator the session is denied access. Default: 8 if logging of session/job initiation is enabled, 13 otherwise. (Optional parameter)
HIPRI	is a request for maximum session-selection input priority, causing the remote session to be scheduled regardless of the current job fence or execution limit for sessions. NOTE: You can specify this only if you have system manager or supervisor capability. (Optional parameter)

## Syntax Considerations

The following provides syntax rules that apply:

- No spaces are allowed around the periods in the optional file reference, or separating *dsdevice* and the # sign, in Record 1.
- Passwords are not allowed with the local user, account, and group names. They are not necessary since the local user passes the security password checks when logging on the local session.

### NOTE

Remote logon parameters must define a valid logon known to the remote machine. For example, if a particular user name requires a password on the remote machine, the password parameter is required in the data-base-access file and must be supplied in the HELLO command.

## FILENAME

After you have created the file with the Editor, you must KEEP it UNNumbered. The file name must follow the same rules as a data base name. It must be an alphanumeric string from 1 to 6 characters, the first character must be alphabetic.

## USER IDENTIFICATION

Records 3 through n define a table that tells TurboIMAGE which user, account, and group names on the local computer may access which user, account, and group names on the remote computer. You may specify remote user identification for more than one local user by creating a record for each local "user.account,group" in the format of Record 3 shown above. An at-sign (@) may be substituted for any user, account, or group name in the record. If an at-sign is substituted for *lusername*, *lacctname*, or *lgrouname*, the name is replaced with the corresponding name specified at log-on time.

TurboIMAGE searches for a match between the local user, account and group names in the user table and the names used to log on to the local session. When a match has been found, TurboIMAGE performs a REMOTE HELLO using the corresponding *rusername*, *racctname*, *rgrouname*, and passwords, if present. If an at-sign is found, it is replaced with the corresponding name to the left of =HELLO. For example, if the record contains USERA.ACCTA,GROUPA=HELLO@.ACCTB,@, TurboIMAGE replaces the first at-sign with USERA and the second with GROUPA. If an at-sign is not found, no substitutions are made. In either case, the information to the right of =HELLO is used as the remote log-on identification.

The following is an example of proper syntax.

## Example

```
Record 1  FILE ORDERS;DEV=DSL1#DISC
Record 2  DSLINE DSL1
Record 3  USERA.ACCTA, GROUPA=HELLO USERB.ACCTA,GROUPB
Record 4  @.ACCTA,GROUPA=HELLO USERA.ACCTA,GROUPA
Record 5  USERB.ACCTB,@=HELLO USERB.ACCTX,@
End of file
```

If a user logs on with the log-on identification indicated in the first column below, TurboIMAGE will use the corresponding USER.ACCT,GROUP identification in the second column to establish communication with the remote system.

## Example

	Log-on Identification	Remote Identification
User1	USERA.ACCTA,GROUPA	USERB.ACCTA,GROUPB
User2	USERB.ACCTA,GROUPA	USERA.ACCTA,GROUPA
User3	USERB.ACCTB,GROUPB	USERB.ACCTX,GROUPB
User4	USERA.ACCTB,GROUPB	None, no match found.

The first user's log-on identification matches the local user, account, and group names specified in Record 3, so the remote names specified in that record are used. The second user's account matches Record 3 but the user name does not, so TurboIMAGE looks for another table entry with account ACCTA. Since the entry in Record 4 specifies any user (@) of ACCTA if their group is GROUPA, the second user's remote identification will be that specified in Record 4.

The third user logs on to ACCTB and a match is found in Record 5 since it specifies the same user name and accepts any group in the account.

The fourth user's account matches Record 5 but the user name does not match. Therefore, the fourth user cannot access the remote data base with this application program.

## ACTIVATING A DATA-BASE-ACCESS FILE

After you have constructed a data-base-access file, you must use the DBUTIL utility program to activate the file. The complete syntax for running the utility program is given in Section 8. Here is a summary of the operating instructions:

```
:RUN DBUTIL.PUB.SYS
.
.
>>ACTIVATE data-base-access file name

Verification follows:

FILE command:    <result>
DSLIN command:   <result>
HELLO command:   <result>

ACTIVATED

>>EXIT
```

DBUTIL verifies that the file to be activated:

- has a file code of zero
- is an UNNUMBERED, ASCII file
- has a record length  $\leq 128$  characters
- has at least three records.

If any of these conditions is not satisfied, activation fails and one of the following messages is printed:

```
filename is NOT a suitable data-base-access file

filename is already ACTIVE
```

If all of the above are satisfied, DBUTIL prints the following message:

```
Verification Follows:
```

Then the utility program verifies the syntax of:

- Record 1
- Record 2 through dsdevice, which must be identical to the dsdevice specified in Record 1
- Records 3 through  $n$ , through the parameter *rgpasw*.



This means that for Records 2 through  $n$  only the positional parameters (those whose function is determined by their relative position within the command) are verified by DBUTIL. The remaining key word parameters are checked by the command interpreter at DBOpen time.

If all of the above conditions are met, DBUTIL successfully activates the data-base-access file, by changing the file code to the TurboIMAGE reserved code -402, which makes it a privileged (PRI) file.

## DEACTIVATING A DATA BASE-ACCESS FILE

In order to deactivate the data-base-access file, you use the DEACTIVATE command of the DBUTIL utility program. Complete syntax for this program is given in Section 8. Here is a summary of the operating instructions:

```
:RUN DBUTIL.PUB.SYS
.
.
>>DEACTIVATE data-base-access file name
DEACTIVATED
>>EXIT
```

You may want to do this in order to edit the content of the data-base-access file or prevent access through this file to the remote data base.

## REFERENCING THE DATA BASE

To reference the data base from your local application program, use the data-base-access file name instead of the root file name when calling the TurboIMAGE procedure. The word array specified as the base parameter must contain a pair of blanks followed by the left-justified data-base-access file name and terminated by a semicolon or blank ( $\Delta$ ). TurboIMAGE recognizes the -402 file code and establishes a communications link to the remote HP 3000. If the data base is successfully opened, TurboIMAGE replaces the pair of blanks with the extra data segment number of the assigned Remote Data Base Control Block. The base parameter must remain unchanged for the remainder of the process. When the application program calls the DBCLOSE procedure or terminates execution, automatic REMOTE BYE and DSLINE commands are issued to terminate the session and close the communications line.

The example in Figure 9-5 illustrates how to create and activate a data-base-access file. In this case, the file named ORDDBA is to be used to gain access to the ORDERS data base residing on a remote system in the PAYACCT account. The remote system is referenced by dsdevice name MY.

After the data-base-access file is created using the Editor, it is enabled by using the DBUTIL utility program.

```
1  :HELLO MEMBER1.PAYACCT
    .
2  :EDITOR

HP32201A.7.15 EDIT/3000  MON, DEC 3, 1984, 1:36 PM
(C) HEWLETT-PACKARD CO. 1983
3  /ADD
    1      FILE ORDERS;DEV=MY#
    2      DSLINE MY
    3      MEMBER1.PAYACCT=HELLO MEMBER1.PAYACCT
    4      MEMBER2.PAYACCT=HELLO @.PAYACCT
    5      //
4  /KEEP ORDDBA,UNN
    /END

5  :RUN DBUTIL.PUB.SYS
    >>ACTIVATE ORDDBA
        Verification follows:
        FILE command:    Looks good
        DSLINE command:  Looks good
        HELLO command:   Looks good
        HELLO command:   Looks good
        ACTIVATED
    >>EXIT

END OF PROGRAM
```

Figure 9-5. Preparing a Data-Base-Access File

## Discussion

- 1 Initiate MPE session by logging on with appropriate user name and account.
- 2 Initiate text editor execution.
- 3 Enter the Editor ADD command in response to the first prompt, then enter the lines to define the data-base-access file.
- 4 Save the work file in a disc file named ORDDBA. Then terminate Editor.
- 5 Initiate execution of DBUTIL and activate the data-base-access file ORDDBA. Verification messages will follow (in session mode). Exit from DBUTIL.

Figure 9-6 illustrates use of the data-base-access file through a program named APPLICAN. After logging on to the local system, the user runs the program named APPLICAN from the local session. The base array in this program contains  $\Delta\Delta$  ORDDBA. When a call to DBOPEN is executed, TurboIMAGE establishes a communication line and remote session. When the program closes the data base, TurboIMAGE closes the line and terminates the remote session.

```

1  :HELLO MEMBER2.PAYACCT
      .
      .
2  :RUN APPLICAN
3  DS LINE NUMBER = #L4
    HP3000 / MPE V/E  G.00.00  MON, DEC 3, 1984,  1:56 PM
4  WELCOME TO SYSTEM B.

    CPU=2.  CONNECT=1. MON, DEC 3, 1984,  1:59 PM
    1 DS LINE WAS CLOSED
5  :BYE

```

Figure 9-6. Using a Data Base-Access File

## Discussion

- 1 Initiate MPE session on the local system by logging on with the appropriate user name and account.
- 2 Execute application program APPLICAN.
- 3 TurboIMAGE establishes a communications line and remote session.
- 4 When the data base is closed, TurboIMAGE closes the communications line and terminates the remote session.
- 5 Log off local system.

## USING QUERY

When you use QUERY to retrieve information from a data base, you must specify a data base name, password and access mode before you can actually access the data base. The "DATA BASE=" prompt can be answered with a remote data base name or the data-base-access file name. Note, however, that performance can be significantly improved if you run QUERY in remote session, thereby accessing the data base on the system where it resides, rather than run QUERY locally to access a remote data base. A detailed description of QUERY/DS is provided in the *QUERY Reference Manual*.

In addition to the data elements discussed in Section 2, data bases comprise a number of structure elements. TurboIMAGE creates and uses these, along with various internal techniques, to provide rapid and efficient access to the data base content. This section describes these structures and techniques to give you a complete understanding of the way TurboIMAGE works. A summary of design considerations is included at the end of this section.

## DATA SET STRUCTURAL ELEMENTS

The following internal structures are used by TurboIMAGE to manage the information in data sets.

### Pointers

TurboIMAGE uses pointers to link one data set record to another. A pointer is a two word entity (doubleword) containing the block number in the first three bytes plus one byte which contains the offset record number.

### Data Chains

A data chain is a set of detail data set entries that are bi-directionally linked together by pairs of pointers. All entries having a common search item value are placed in the same chain. Each chain has a first and a last member. The pointer pairs constitute backward and forward pointers to the entry's predecessor and successor within the chain. The first member of a chain contains a zero backward pointer and the last member of a chain contains a zero forward pointer. A single chain may consist of at most  $2^{31}-1$  (2,147,483,647) entries.

### Media Records

TurboIMAGE transfers information to and from a storage location on disc in the form of a media record. A media record consists of both an entry and its pointers or a null record if no data entry is present.

## Media Records of Detail Data Sets

For each detail entry, the media record consists of the entry itself preceded by all of its related data chain pointer pairs. The number of pointer pairs corresponds to the number of paths specified for the data set within the schema. Figure 10-1 illustrates a media record for a detail data set defined with two paths. The first set of pointers corresponds to the first path defined in the set part of the schema and the second set corresponds to the second path.

backward pointer path 1	forward pointer path 1	backward pointer path 2	forward pointer path 2	entry...
-------------------------------	------------------------------	-------------------------------	------------------------------	----------

Figure 10-1. Media Record for Detail Entry

## Chain Heads

TurboIMAGE locates the first or last member of a chain within a detail data set by using a *chain head*. The chain head for a particular chain is stored with an entry in the corresponding master data set whose search item value is the same as the detail search item value defining the chain. Each chain head is six words long. The first word is a double integer count of the number of member entries in the referenced chain. The count is zero if the chain is empty. The remaining four words contain two doubleword pointers. One points to the last chain entry, the other to the first chain entry. If the count is zero, these pointers are both zero. If the count is 1, these pointers have the same value.

## Primary Entries

Selection of record addresses for master entries begins with a calculated address determined by an algorithm applied to the value of each entry's search item. The algorithm is described later in this section. Each such calculated address is known as a *primary address* and each entry residing at its primary address is called a *primary entry*.

## Secondary Entries

A new entry with a unique search item value will be assigned the same primary address as an existing primary entry whenever the search item values of both entries generate the same calculated address. When this occurs, the entries are called *synonyms*. TurboIMAGE assigns the new entry a *secondary address* obtained from unused records in the vicinity of the primary entry. All entries residing at secondary addresses are called secondary entries.

## Synonym Chains

A primary entry may have multiple synonyms. A *synonym chain* consists of the primary entry and all of its synonyms. Each synonym chain is maintained by a five-word chain head in the media record of the primary entry and five-word links in the media records of the secondary entries. Note that a master data set entry may contain both a synonym chain head and multiple detail chain heads. These are two distinct types of chain heads.

If no secondary entries are present, the synonym chain count is 1 and the pointers to the first and last secondary entries are zeros. If  $N$  secondaries are present, the chain count is  $N+1$  and the pointers reference the first and last secondary entries.

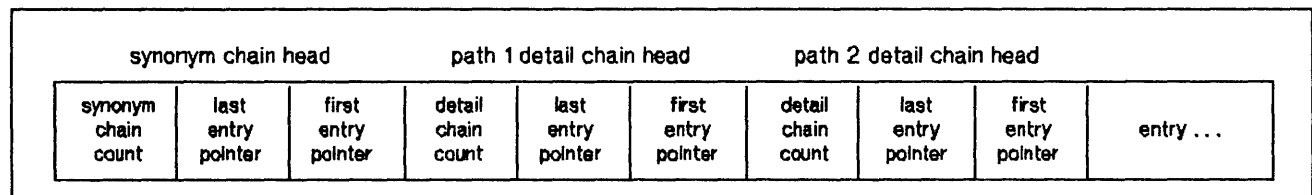
The first word of the five-word link in the media record of each secondary entry is always zero. The remaining four words consist of two doubleword pointers bi-directionally linking the secondary entries of the synonym chain to each other. As with detail chains, the first member of the synonym chain contains a zero backward pointer and the last member of the chain contains a zero forward pointer.

## Media Records of Master Data Sets

Media records of master data entries are composed of the following:

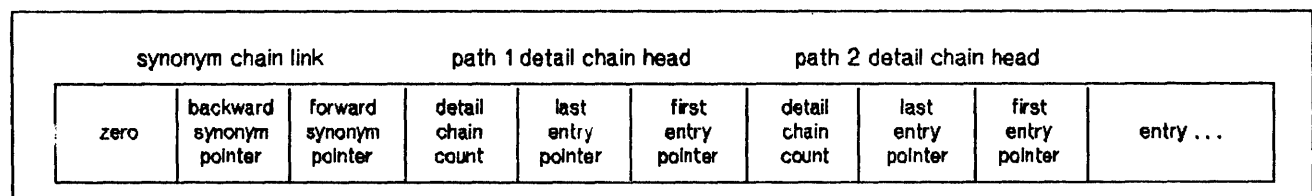
- A five-word field serving as a synonym chain head for primary entries or a synonym chain link for secondary entries.
- A 6 times  $n$  word field in which the chain heads of all related detail chains are maintained.  $n$  is the number of relationships defined for the master data set. There may be between 0 and 16 relationships.
- The entry itself.

Figure 10-2 illustrates the media record for a primary entry of a master data set with two paths defined.



**Figure 10-2. Media Record for Primary Entry**

Figure 10-3 illustrates a media record for a secondary entry of a master data set with two paths defined.



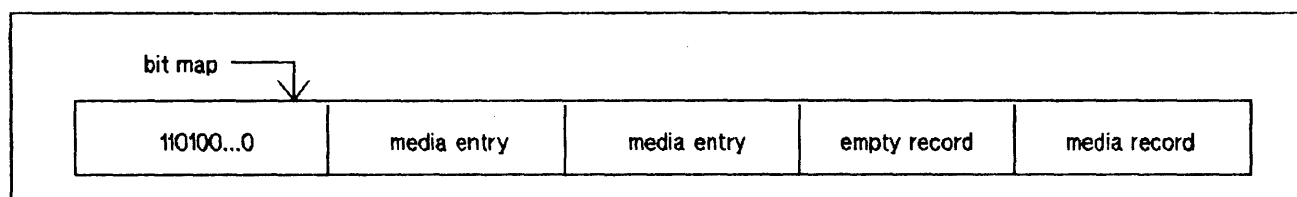
**Figure 10-3. Media Record for Secondary Entry**

When more than one detail chain head is present, they are physically ordered left-to-right in the order that the associated paths are specified in the schema.

## Blocks and Bit Maps

Each group of media records involved in a single disc transfer is a *block*. The first word or words of each block contain a bit map employed by TurboIMAGE to indicate whether a particular media record of the block contains a data set entry or is empty. There is one bit for each record in the block. The bits occur in the bit map in the same order that the records occur in the block. The bit map occupies as many integral words as are required to contain one bit for each record in the block. If a bit is zero, the corresponding record is empty. If a bit is one, the record contains a data entry preceded by the associated structure information.

The format of a block is illustrated in Figure 10-4. The sample block contains four records and the third record contains no entry.



**Figure 10-4. Block with Blocking Factor of Four**

## **RUN-TIME TurboIMAGE CONTROL BLOCKS**

As mentioned in Section 4, TurboIMAGE uses dynamically-constructed control blocks resident in privileged extra data segments to provide and control user access to a data base through the TurboIMAGE procedures. The contents of these control blocks are maintained by TurboIMAGE, and it is not necessary to know the details in order to use the TurboIMAGE procedures. However, the following descriptions are provided for those who prefer to understand the control blocks and their functions.

### **Local Data Base Access**

Four structures are involved in local data base access:

- The Data Base System Control Block (DBS),
- the Data Base Globals Control Block (DBG),
- the Data Base Buffer Area Control Block (DBB),
- the Data Base User Local Control Block (DBU).

The DBS is created when the first user opens any data base on a system. There is only one DBS per system and it is used as a system wide table to locate the current DBG and DBB for any previously opened data base.

Both the DBG and the DBB are created when the first user opens the data base (DBOPEN). They remain allocated until the last user closes the data base. There is only one DBB and DBG per each opened data base, regardless of the number of users.

The DBG is derived mostly from the root file and contains global information required by TurboIMAGE intrinsics during run-time. In addition, the DBG contains the lock table which holds user level locking information. The DBG is used as a reference area for global data and lock information.

The DBB contains a set of (I/O) buffers which are shared by all concurrent users accessing the data base. These buffers may contain data from any of the data sets. The number of I/O buffers vary in number with the number of concurrent users and can be controlled by the data base administrator. (Refer to the DBUTIL >>SET command in Section 8.) Each buffer is as large as the largest block of the data base. The DBB also contains information pertaining to logging and recovery. The DBB is used to retrieve, log and update data located within the data set files.

A two level resource priority locking scheme is used within the DBB to allow single-buffer operations to access the control block concurrently. This involves DBGET, DBFIND and DBUPDATE processes. DBPUT and DBDELETE operations are unable to access the DBB concurrently. These multi-buffer operations must hold a global lock on the DBB throughout the operation.



The DBU is created when a user issues a DBOPEN, and remains allocated until the corresponding DBCLOSE is issued. One DBU exists for each user of each open data base. The DBU contains information pertaining to the user's own individual access to the data base. This includes information about the user's access mode, record position, list specifications and security table. All TurboIMAGE intrinsics process on the DBU except accesses for global and buffer area information found in the DBG and DBB.

When accessing a local data base, the TurboIMAGE procedures usually make use of, and may modify information, in the DBG, the DBB, and the specific user's DBU for this data base.

## Remote Data Base Access

TurboIMAGE provides the capability of accessing a data base on a remote HP3000 from a user program running on the local HP 3000, as described in Section 9. This capability is provided in conjunction with DS/3000 and is accomplished by transmitting TurboIMAGE data base access requests (DBGGET, DBPUT, and so forth) to the remote computer where they are executed and the results returned to the local calling process. The control block structures used by TurboIMAGE on the remote computer which contains the data base are those described in the preceding paragraphs.

On the local computer running the user application program, a structure called the Remote Data Base Control Block (RDBCB) is constructed and used by TurboIMAGE. One RDBCB exists for each user accessing a remote TurboIMAGE data base (each access path to a remote data base). The RDBCB is created when the user opens the data base and is released when the user closes the data base. It resides in a privileged extra data segment associated with the user application process on the local computer. The RDBCB contains data base, set, and item information plus the work areas necessary on the remote computer. Access between a TurboIMAGE data base and an IMAGE/3000 data base residing on different computers is allowed. If the new limitations have been used on the TurboIMAGE data base then access from an IMAGE/3000 data base will not be successful. This is because the IMAGE/3000 RDBCB will be too small to handle TurboIMAGE's expanded limitations. However, remote access from TurboIMAGE to IMAGE/3000 will always be allowed. Returned data and status information is also processed in the RDBCB and is transferred to the appropriate user stack areas before TurboIMAGE returns to the local calling process.

## Control Block Sizes

It is impossible to predict the exact length of the control blocks used by TurboIMAGE to manage user access to data bases. However, Table 10-1 contains formulas that provide a way of approximating their sizes.

The exact length of the DBU and the exact current length of the DBG are returned in the status array by DBOPEN. (These lengths do not include the few words of overhead used by MPE.)

**Table 10-1. Formulas for Approximating Control Block Sizes**

CONTROL BLOCK	APPROXIMATE LENGTH (IN WORDS)
DBG	$4000 + r + i + s$
DBB	$4000 + 26s + n(b + 13)$
DBU	$4000 + 25s + i + i*s$
RDBCB	$4000 + 12i + 13s$
<i>in which</i>	
r	is the ROOT LENGTH as reported by the Schema Processor.
b	is the BUFFER LENGTH as reported by the Schema Processor.
n	is the number of I/O buffers in the DBCB. This number normally varies with the number of concurrent users of a data base. DBUTIL can be used to display and change the values which TurboIMAGE will use.
i	is the number of data items in the data base.
s	is the number of data sets in the data base.

## INTERNAL TECHNIQUES

Although it is not necessary to know the following techniques to use TurboIMAGE, an understanding of them may help you design a more efficient data base.

### Primary Address Calculation

TurboIMAGE employs two distinct methods of calculating primary addresses in master data sets. The first method applies to master data sets with search items of type I, J, K, or R. The low order (rightmost) 31 bits of the search item value, are used to form a 32-bit doubleword value. This doubleword value is then decremented by one, reduced modulo the data set capacity and incremented by one to form a primary address. For example, if an integer search item has a value of 529 and the data set capacity is 200, the primary address is 129. The calculation is as follows:

$$\begin{array}{rcl}
 529 - 1 & = & 528 \\
 & & - 400 \quad (200 \times 2) \\
 & & ---- \\
 & & 128 \\
 & & + 1 \\
 & & ---- \\
 & & 129
 \end{array}$$

The second method of primary address calculation applies to master data sets with search items of type U, X, Z, or P. In this case, the entire search item value regardless of its length is used to obtain a positive doubleword value. This value is reduced modulo the data set capacity and then incremented by one to form a primary address. The algorithm used to obtain the doubleword intermediate value attempts to approximate a uniform distribution of primary addresses in the master data set, regardless of the bias of the master data set search item values.

The intent of the two primary address algorithms is to spread master entries as uniformly as possible throughout the record space of the data file. This uniform spread reduces the number of synonyms occurring in the master data set.

<b>NOTE</b>
-------------

In general, a master data set with a capacity equal to a prime number or to the product of two or three primes may yield fewer synonyms than master data sets with capacities consisting of many factors.

## Migrating Secondaries

In some cases, secondary entries of master data sets are automatically moved to storage locations other than the one originally assigned. This most often occurs when a new master data entry is assigned a primary address occupied by a secondary entry. By definition, the secondary entry is a synonym to some other primary entry resident at their common primary address. Thus, the new entry represents the beginning of a new synonym chain. To accommodate this new chain, the secondary entry is moved to an alternate secondary address and the new entry is added to the data set as a new primary entry. This move and the necessary linkage and chain head maintenance is done automatically.

A move can also occur when the primary entry of a synonym chain having one or more secondary entries is deleted. Since retrieval of each entry occurs through a synonym chain, each synonym chain must have a primary entry. To maintain the integrity of a synonym chain, TurboIMAGE always moves the first secondary entry to the primary address of the deleted primary entry.

## Space Allocation for Master Data Sets

Space allocation for each master data set is controlled by a doubleword free space counter resident in the user label of the data set, and by all the bit maps, one in each block of the data set.

When a new entry is added, TurboIMAGE decrements the free space counter and sets the bit corresponding to the newly assigned record address to a 1. If the bit is a zero before the record is added, the assigned record address is the primary address. If the bit is a one before the record is added, it indicates that an entry already exists. A secondary address is determined by a cyclical search of the bit maps for a 0 indicating an unused record and this address is assigned to the entry being added.

## Space Allocation for Detail Data Sets

Space allocation for each detail data set is controlled by a doubleword free space counter, a doubleword end-of-file pointer and a doubleword pointer to a *delete chain*. The end-of-file pointer contains the record address of the highest-numbered entry which has existed so far in the data set. The delete chain pointer locates the record address of the entry which was most recently deleted. When each detail data set is first created, the end-of-file pointer and delete chain pointer are both zero.

When a new entry is added to a detail data set, TurboIMAGE assigns to it the record address referenced by the delete chain pointer, unless the pointer is zero. If the chain pointer is zero, the end-of-file pointer is incremented and then used as the assigned record address. The free space counter is decremented in either case.

When an existing entry is deleted, its media record is zeroed, the first two words are replaced with the current delete chain pointer, and the block is written to disc. The delete chain pointer is set to the address of the newly deleted entry and the free space counter is incremented.

The delete chain is, in effect, a last in-first out linked list reusable media record space. Reusable space is always allocated in preference to the unused space represented by the record addresses beyond the end-of-file pointer.

Addition and deletion of data entries also requires data chain maintenance and turning on or turning off the corresponding bit of the appropriate bit map. Both of these are necessary for retrieval integrity but neither play a role in space allocation for detail data sets.

## Buffer Management

TurboIMAGE maintains buffers in the DBB for all users of an open data base. The intrinsics of DBFIND, DBGET, DBUPDATE, DBPUT and DBDELETE use these buffers for data transfers. When an intrinsic issues a request for a data block, TurboIMAGE will search the buffer area for the block. If the data block is not found then TurboIMAGE must pick an overlay candidate buffer. The candidate buffer will hold a copy of the data set block read in from disc. This selection process uses a "modified, Least-Recently-Used" (LRU) buffer replacement algorithm. The purpose of this algorithm is to delay writing dirty buffers out to disc while retaining those buffers which will most likely be reused. Three criteria are used to determine the selection of the buffer. The selection process is listed below in descending priority:

- Is the buffer dirty?
- What is the possibility that the buffer will be needed again?
- When was the buffer last used?

TurboIMAGE buffer management provides the capability to issue buffer writes as either MPE waited I/O or MPE no-wait I/O. The user may alter the buffer posting mechanism by changing options in TurboIMAGE or MPE. In TurboIMAGE Output Deferred Mode (AUTODEFER) can be used along with MPE disc caching to increase I/O performance, however ILR, and roll-back recovery must be disabled. The user must take into consideration performance, type of I/O and data recoverability when determining which options to use for buffer management.

## Locking Internals

Within the DBG is a lock area which is initially 128 words long and may be expanded to 8192 words in length subject to system availability of the necessary resources. It contains a fixed area of 64 words. The remaining area is managed dynamically and provides space for the following entries:

### ACCESSOR ENTRIES

Each of these is 8 words long. One is created for each successful call to DBOPEN (each access path). Although located in the lock area, this is the link with which TurboIMAGE controls access to the data base. It is deleted when DBCLOSE is called for the access path, and the space is reused.

### SET ENTRIES

Each of these is 16 words long. One is created for every data set that is specified in a lock request. Therefore, the maximum number of set entries is equal to the number of data sets in the data base. These are never deleted.

### DESCRIPTOR ENTRIES

These entries contain the internal form of the lock descriptors specified in locking mode 5 or 6. Each entry is  $8 + V$  words long rounded up to the next multiple of 8.  $V$  is the number of words required to hold the value of the data item used for locking. These entries disappear when the locks are released (when DBUNLOCK is called) and the space is reused.

TurboIMAGE issues three different types of error messages:

- Schema Processor Error Messages listed in Tables A-1 through A-3.
- Library Procedure Error Messages listed in Tables A-4 through A-7.
- Utility Error Messages listed in Tables A-8 through A-10.

Schema processor error messages result from errors detected during processing of the data base schema. The library procedure error messages consist of condition words returned to the calling program from the library procedures. The utility error messages are caused by errors in execution of the data base utility programs.

## SCHEMA PROCESSOR MESSAGES

The Schema Processor accesses three files:

- The textfile (DBSTEXT) containing the schema records and Schema Processor commands for processing.
- The listfile (DBSLIST) containing the schema listing, if requested, and error messages, if any.
- The root file, if requested, created as a result of an error-free schema.

Any file error which occurs while accessing any of these files causes the Schema Processor to terminate execution. A message indicating the nature of the error is sent to \$STDLIST (and to the listfile, if listfile is different from \$STDLIST).

Table A-1 lists the various file error messages. Each such message is preceded by the character string:

**\*\*\*\*\*FILE ERROR\*\*\*\*\***

Additionally, the Schema Processor prints a standard MPE file information display on the \$STDLIST file. Refer to *MPE Intrinsic Reference Manual* or *Error Messages and Recovery Manual* for the meaning of MPE file information displays.

Schema Processor command errors may occur. They neither cause termination nor do they prohibit the creation of a root file. In some cases, however, the resultant root file will differ from what might have occurred had the commands been error free. Command errors are added to an error count which, if it exceeds a limit (see Section 3), will cause the Schema Processor to terminate execution.

## Error Messages

Table A-2 lists the various Schema Processor command error messages. Each such message is preceded by the character string:

\*\*\*\*\*ERROR\*\*\*\*\*

Data base definition syntax errors may be detected by the Schema Processor. Their existence does not cause termination but does prohibit root file creation. Discovery of one may trigger others which disappear after the first is corrected. Also, detection of one may preclude detection of others which appear after the first is corrected. Syntax errors are also added to an error count which, if excessive, will cause Schema Processor termination.

Table A-3 lists the various syntax error messages. As with command errors, each syntax error is preceded by the character string:

\*\*\*\*\*ERROR\*\*\*\*\*

If the LIST option is active (see Section 3), error messages for command errors and syntax errors appear in the listfile following the offending statement. If the NOLIST option is active, only the offending statement, followed by the error message, is listed.

Table A-1. TurboIMAGE Schema Processor File Errors

MESSAGE	MEANING	ACTION
READ ERROR ON <i>file name</i>	FREAD error occurred on the specified file.	Check <i>textfile</i> or :FILE command.
UNABLE TO CLOSE <i>file name</i>	FCLOSE error occurred on specified file. May be caused by duplicate file in group with same name as root file.	Change data base name or purge file of same name. Or, be sure correct file and file name used. Check :FILE commands used. If other cause, consult <i>Intrinsics Reference Manual</i> for similar message.
UNABLE TO USE <i>file name</i>	Specified file cannot be FOPENed or its characteristics make it unsuitable for its intended use.	Same as above.
UNABLE TO WRITE LABEL OF <i>file name</i>	FWRITELABEL error occurred on specified file.	Same as above.
UNEXPECTED END-OF-FILE ON <i>file name</i>	Call to FREAD or FWRITE on specified file has yielded unexpected end of file condition.	Same as above.
WRITE ERROR ON <i>file name</i>	FWRITE error occurred on the specified file.	Same as above.

Table A-2. TurboIMAGE Schema Processor Command Errors

MESSAGE	MEANING	ACTION
COMMAND CONTINUATION NOT FOUND	If the schema processor command is continued to the next record, the last non-blank character of the preceding line must be an ampersand (&) and the continuation record must start with a dollar sign (\$).	Examine the schema textfile to find any incorrect commands. Edit the textfile and run the Schema Processor again.
COUNT HAS BAD FORMAT	The numbers in ERRORS, LINES, or BLOCKMAX parameters of the \$CONTROL command are not properly formatted integer values.	Same as above.
ILLEGAL COMMAND	The Schema Processor does not recognize the command. Valid commands are \$PAGE, \$TITLE, and \$CONTROL.	Same as above.
IMPROPER COMMAND PARAMETER	One of the commands in the parameter is not valid.	Same as above.
MISSING QUOTATION MARK	Character string specified in \$PAGE or \$TITLE command must be bracketed by quotation marks ("").	Same as above.
SPECIFIED TITLE IS TOO LONG	Character string in \$TITLE or \$PAGE command exceeds 104 characters.	Same as above.



Table A-3. TurboIMAGE Schema Syntax Errors

MESSAGE	MEANING	ACTION
AUTOMATIC MASTER DATA SET MUST HAVE SEARCH ITEM ONLY	AUTOMATIC master data sets must contain entries with only one data item. The data item must be a search item.	Same as above.
BAD CAPACITY OR TERMINATOR	Either the number in the CAPACITY: statement is not an integer between 1 and 2 -1, or a semicolon is missing	Examine the schema textfile to find the error and edit the file. Run the Schema Processor again.
BAD DEVICE CLASS NAME	The device class name specified contains an invalid character. The name must be less than 8 characters and begin with a letter.	Same as above.
BAD DEVICE CLASS NAME OR TERMINATOR	The device class name specified contains an invalid character or was not ended with a semi-colon ";"	Same as above.
BAD LANGUAGE	Language name contains invalid characters, or the language number is not a valid integer.	Same as above.
BAD PATH COUNT OR TERMINATOR	The path count in the master data set definition is not an integer from 1 to 16 (for an automatic), or 0 to 16 (for a manual). This message may also mean the path count is not followed by a (").	Same as above.
BAD CHARACTER IN USER CLASS NUMBER	User class number in password is not an integer from 1 to 63.	Same as above.
BAD DATA BASE NAME OR TERMINATOR	Data base name in BEGIN DATA BASE statement is not a valid data base name beginning with an alphabetic character and having up to 6 alphanumeric characters. Or, the name is not followed by a semicolon (;).	Same as above.

Table A-3. TurboIMAGE Schema Syntax Errors (Continued)

MESSAGE	MEANING	ACTION
BAD DATA SET TYPE	The data set type designator is not AUTOMATIC (OR A), MANUAL (OR M), or DETAIL (OR D).	Same as above
BAD PATH CONTROL PART DELIMITER	Data item defined as sort item in detail data set is not properly delimited with parentheses ().	Same as above.
BAD PATH SPECIFICATION DELIMITER	Name of master data set following search item name in detail data set definition is not followed by a ")", or by a sort item name in parentheses.	Same as above.
BAD READ CLASS OR TERMINATOR	Read user class number defined for either a data set or data item is not an integer from 0 to 63, or it is not terminated by a comma (,) or slash (/).	Same as above.
BAD SET NAME OR TERMINATOR	The data set name does not conform to the rules for data set names (1 to 16 characters beginning with a letter. The name may include + - * / ? ' # % & @), or it is not terminated by the correct character for the context it in which it appears.	Same as above.
BAD SUBITEM COUNT OR TERMINATOR	Subitem count for a data item defined in schema item part is not an integer from 1 to 255.	Same as above.

Table A-3. TurboIMAGE Schema Syntax Errors (Continued)

MESSAGE	MEANING	ACTION
BAD SUBITEM LENGTH OR TERMINATOR	Subitem length for data item defined in schema item is not an integer from 1 to 255.	Examine schema textfile to find the error and edit the file. Run Schema Processor again.
BAD TERMINATOR-' ' OR ' ' EXPECTED	Items within an entry definition must be separated from each other with commas and terminated with a semicolon.	Same as above.
BAD TERMINATOR-' ' EXPECTED	Password or capacity was not followed by a semicolon.	Same as above.
BAD TYPE DESIGNATOR	Data item defined in schema item part is not defined as type I,J,K,R,U,X,Z, or P.	Same as above.
BAD WRITE CLASS OR TERMINATOR	Write user class number shown for the data set or data item is not an integer from 0 to 63, or it is not terminated by a right parenthesis ')' or comma.	Same as above.
'CAPACITY' EXPECTED	CAPACITY statement must follow entry definition in the definition of data sets in the set part of schema.	Same as above.
DATA BASE HAS NO DATA SETS	No data sets were defined in the set part of schema. The data base must contain at least one data set.	Same as above.
DATA BASE NAME TOO LONG	Data base name has more than six characters.	Same as above

Table A-3. TurboIMAGE Schema Syntax Errors (Continued)

MESSAGE	MEANING	ACTION
DUPLICATE ITEM SPECIFIED	The same data item name was used more than once in the entry definition of data sets.	Examine the Schema textfile to find the error and edit the file. Run the Schema Processor again.
DUPLICATE SET NAME	The same data set name was used to define more than one data set in the set part of schema.	Same as above.
'ENTRY' EXPECTED	Each set defined in the set part of schema must contain ENTRY: statement followed by the data item names of the data items in entry.	Same as above.
ENTRY TOO BIG	The number and size of the data items defined for an entry makes an entry too big for maximum block size. The block size is specified by \$CONTROL, or BLOCKMAX= command or default.	Same as above.
ENTRY TOO SMALL	A detail data set that is not linked to any master data set must have a data entry length of two or more words. This length is determined by adding the size in words of each data item defined in the data entry.	Same as above.

Table A-3. TurboIMAGE Schema Syntax Errors (Continued)

MESSAGE	MEANING	ACTION
ILLEGAL USER CLASS NUMBER	User class number defined in schema password part is not an integer between 1 and 63.	Examine schema textfile to find incorrect statement and edit the textfile. Run Schema Processor again.
ILLEGAL ITEM NAME OR TERMINATOR	The data item name does not conform to naming rules. (Names must start with a letter and may have up to 16 alphanumeric characters including + - ? / # \$ & * @ ). Or if in the item part, it is not followed by a comma.	Same as above.
ITEM TOO LONG	The length of a single data item may not exceed 2047 words.	Same as above.
LANGUAGE EXPECTED	The Schema Processor expected to find a LANGUAGE statement after the comma following the BEGIN DATA BASE name statement.	Same as above.
LANGUAGE NOT SUPPORTED	Language specified is not currently supported on your system, or it is not a valid language.	Same as above.
MASTER DATA SET LACKS EXPECTED DETAILS	Master data set was defined with a non-zero data count, but the number of detail search items which back-referenced the master is less than the value of the path count.	Same as above.
MASTER DATA SET LACKS SEARCH ITEM	A master data set was defined without defining one of the data items in the set as a search item.	Same as above.
MORE THAN ONE KEY ITEM	A master data set cannot be defined with more than one search item.	Same as above.

Table A-3. TurboIMAGE Schema Syntax Errors (Continued)

MESSAGE	MEANING	ACTION
MORE THAN ONE PRIMARY MASTER	User has defined more than one primary path for a detail data set.	Examine the Schema textfile to find the incorrect statement and edit the file. Run the Schema Processor again.
'NAME:' OR 'END.' EXPECTED	Schema Processor expected , at this point, to find the beginning of another data set definition, or the end of schema.	Same as above.
NATIVE LANGUAGE SUPPORT ERROR	NLS/3000 returned an error.	Notify the System Manager
'PASSWORDS:' NOT FOUND	'PASSWORDS:' statement must immediately follow the BEGIN DATA BASE statement in schema. If it does not, DBSCHEMA terminates execution.	Examine the schema textfile to find the incorrect statement and edit the textfile. Run Schema Processor again.
PASSWORD TOO LONG	A password defined in data schema cannot exceed eight characters.	Same as above.
REFERENCED SET NOT A MASTER	The data set referenced by the detail data set search item is another detail data set instead of a master data set.	Same as above.

Table A-3. TurboIMAGE Schema Syntax Errors (Continued)

MESSAGE	MEANING	ACTION
SCHEMA PROCESSOR LACKS NEEDED TABLE SPACE	Schema Processor is unable to expand its data stack to accommodate all of the translated information which will make up the root file. It continues to scan the schema for the proper form, but will not perform all of the checks for correctness nor will it create a root file. To process the schema correctly, the operating system must be configured with a larger maximum stack size.	Ask system manager to increase maximum stack size.
SEARCH ITEM NOT SIMPLE	All data items defined in data schema as search items must be simple items.	Examine the schema textfile to find the incorrect statement and edit it. Run the Schema Processor again.
SEARCH ITEMS NOT OF SAME LENGTH	Master search item must be the same length as any related detail data set search item.	Same as above.
SEARCH ITEMS NOT OF SAME TYPE	Master search item must be of the same type as any related detail data set search item.	Same as above.
SET HAS NO PATHS AVAILABLE	More detail data set search items have specified a relationship with a master data set than the number specified in the master sets's path count.	Same as above.
SORT ITEM OF BAD TYPE	Data item defined as sort item must be of type U,K, or X.	Same as above.
SORT ITEM NOT IN DATA SET	Detail data set's entry definition does not include an item which is specified as a sort item for another item in the entry.	Same as above.

Table A-3. TurboIMAGE Schema Syntax Errors (Continued)

MESSAGE	MEANING	ACTION
<b>SORT ITEM SAME AS SEARCH ITEM</b>	The same item cannot be both a search and a sort item for the same path.	Examine the Schema text file and find the incorrect statement and edit it. Run the Schema Processor again.
<b>TOO MANY DATA ITEMS</b>	The item part of schema cannot have more than 1023 data item names	Same as above.
<b>TOO MANY DATA SETS</b>	The data base cannot have more than 199 data sets.	Same as above.
<b>TOO MANY ERRORS</b>	The specified or default number of errors has been exceeded. Processing is terminated.	Correct the errors, or increase the <b>ERROR</b> parameter value.
<b>TOO MANY ITEMS SPECIFIED</b>	The data set entry cannot have more than 255 data items.	Examine the schema textfile and find the incorrect statement. Edit it and run Schema Processor again.
<b>TOO MANY PATHS IN DATA SET</b>	Detail data set entries cannot have more than 16 search items.	Same as above.
<b>UNDEFINED ITEM REFERENCED</b>	A data item appearing in the data set definition was not previously defined in the item part of schema.	Same as above.
<b>UNDEFINED SET REFERENCED</b>	Master data set referenced by detail search item was not previously defined in the set part of schema.	Same as above.



## LIBRARY PROCEDURE ERROR MESSAGES

The success of each call to a TurboIMAGE library procedure is reflected upon return to the user by the hardware condition code and the value of a condition word returned in the first word of the status area.

If the procedure fails to execute properly, the hardware condition code is set to CCL (Condition Code Less) and the procedure returns a negative integer in the condition word. Table A-4 describes the negative condition words resulting from file system and memory management failures, while Table A-5 describes the negative condition words resulting from calling errors and communications errors respectively.

If the procedure operates properly but encounters an exceptional condition, such as end-of-file, the hardware condition code is set to CCG (Condition Code Greater) and the procedure returns a positive integer in the condition word. Table A-6 describes the positive condition words resulting from exceptional conditions.

If the procedure operates properly and normally, the hardware condition code is set to CCE (Condition Code Equal) and the procedure returns zero in the condition word.

In addition to returning a condition word, all TurboIMAGE library procedures put information about the procedure call into the fifth through tenth words of the status area. This information may be useful in debugging your programs, because it describes the conditions under which the particular results were obtained. This information is used by DBEXPLAIN and DBERROR when they are interpreting the results of TurboIMAGE calls.

In a few cases this information is not returned by the TurboIMAGE procedure because it uses the same words in the status area for returning other data. Specifically, successful execution of DBFIND, DBGET, DBUPDATE, DBPUT, or DBDELETE puts the other information here as described in Section 5 of this manual.

For all other returns, from a library procedure, the specified words of the status area have the following contents:

Word	Contents
5	The PB-relative offset within the calling program's code segment of the current procedure call to the TurboIMAGE library procedure (the location of the PCAL instruction in the SPL code).
6	Bits 7-15: The intrinsic number of the called library procedure. Bits 0-3: Zero or the access mode in which the data base is opened.
7	The DB-relative word address of the <i>base</i> parameter.
8	The DB-relative word address of the <i>password</i> , <i>qualifier</i> , or <i>dset</i> parameter.
9	The value of the <i>mode</i> parameter.
10	The PB-relative offset within the library procedure code segment at which return to the calling program was initiated (for HP use only).

Consult the *MPE Commands Reference Manual* for a discussion of PB and DB registers, and the *Systems Programming Language Reference Manual* for more information about the PCAL instruction.

## Abort Conditions

In general, four types of error conditions can cause TurboIMAGE to abort the calling process:

1. A call from a user process with the hardware DB register not pointing to the process stack.
2. A faulty calling sequence.
3. An internal error in an MPE file intrinsic which the calling procedure cannot correct.
4. An internal inconsistency in the data base or the DBG, DBB, or DBU discovered by a library procedure.

In case 1, the procedure prints the standard MPE run-time abort message described in Section 2, of the *Error Messages and Recovery Manual*. In cases 2, 3, and 4, TurboIMAGE prints additional information on the standard list device about the error prior to printing the standard MPE abort message. The first line of this information is:

ABORT: *procedure name* ON DATA BASE *name*;

where *procedure name* is the name of the library procedure which caused the abort and *name* is the name of the data base being accessed at the time of the abort. Table A-7 describes additional lines of information which may appear prior to the standard MPE abort message.

Some of the abort conditions are due to an error in one of the MPE file intrinsics FOPEN, FREADLABEL, FREADDIR, FWRITE LABEL, FWRITEDIR, or FCLOSE. Aborts of this type generally occur after the procedure has possibly altered the data base so that the data base structure has been damaged in some way. Each of the messages in Table A-7, which refer to a TurboIMAGE data file, is followed by an MPE file information display which lists all of the characteristics of the MPE data set or root file where the error occurred, along with an MPE error number. For more information about file error codes consult Section 2 of the *Error Messages and Recovery Manual* and for the file information display, consult Appendix A of that manual.

**I FILES.** When TurboIMAGE detects an internal inconsistency or other abnormal situation, it may create a special "I" file before it terminates. The "I" file consists of the user's stack and TurboIMAGE's data base control blocks. TurboIMAGE only creates these "I" files if a data base user has run DBUTIL and specified ENABLE FOR DUMPING. So, if you want an "I" file, you must specifically request it through this DBUTIL command. Note that "I" files are useful for debugging only if the data base is known to be structurally sound.

Table A-4. TurboIMAGE Library Procedure File System and Memory Management Errors

CCL	PROCEDURE	MEANING	ACTION
-1	MPE intrinsic FOPEN failure	<p>For DBOPEN, error may indicate that data base could not be opened. Possible reasons:</p> <ul style="list-style-type: none"> <li>• Data base name string not terminated with semicolon or blank</li> <li>• Data base does not exist or is secured against access by its group or account security</li> <li>• Data base is already opened exclusive or in mode incompatible with requested mode</li> <li>• MPE file system error occurred.</li> </ul> <p>For DBOPEN, DBINFO, DBFIND, DBGET, DBUPDATE, DBPUT, and DBDELETE, error may occur if:</p> <ul style="list-style-type: none"> <li>• The process has too many files open external to the data base</li> <li>• Data set does not exist or is secured against access</li> <li>• Some other MPE file system error has occurred.</li> </ul>	Determine which of probable causes applies and either modify application program or see system manager about file system error.
-2	MPE intrinsic FCLOSE FAILURE	This is an exceptional error (should never happen) and is returned only by DBOPEN or DBCLOSE. Indicates a hardware or system software failure.	Notify system manager of error.
-3	MPE intrinsic FREADDIR failure	This is an exceptional error (as -2 above) and is returned by DBOPEN, DBFIND, DBGET, DBUPDATE, DBPUT, DBDELETE.	Same as above.
-4	MPE intrinsic FREADLABEL failure	This is an exceptional error (as -2 above) and is returned by DBOPEN, DBINFO, DBFIND, DBGET, DBUPDATE, DBPUT, DBDELETE.	Same as above.
-5	MPE file error <i>nn</i> returned by FWRTIEDIR	This exceptional condition could be returned when DBPUT, DBDELETE, DBUPDATE, or DBCLOSE calls FWRTIEDIR.	Same as above.
-6	MPE file error <i>nn</i> returned by FWRTIELABEL	This exceptional condition could be returned when DBPUT, DBDELETE, or DBCLOSE calls FWRTIELABEL.	Same as above.

Table A-4. TurboIMAGE Library Procedure File System and Memory Management Errors (Cont.)

CCL	PROCEDURE	MEANING	ACTION
-9	MPE intrinsic GETDSEG failure	<p>This is an exceptional error and is returned by DBOPEN when it cannot obtain extra data segment for use as Data Base Control Block. This occurs if required virtual memory space is unavailable or if required DST entry is unavailable.</p> <p>Second word of user's status area is size (in words) of data segment which memory management was unable to supply. Third word is MPE failure code returned by GETDSEG intrinsic.</p>	Notify system manager of problem or wait until system is less busy.

NOTE: For condition words -1 through -6, second word of calling program's status area is the data set number for which file error occurred (zero indicates root file). Third word is MPE failure code returned by FCHECK intrinsic. Refer to *Error Messages and Recovery Manual*, Section 2, for meaning of this code.

Table A-5. TurboIMAGE Library Procedure Calling Errors

CCL	CONDITION	MEANING	ACTION
-11	Bad <i>base</i> parameter	For DBOPEN, the first two characters in <i>base</i> are not blank, or data base name contains special characters other than period. For all other procedures, either first two characters in <i>base</i> do not contain the value assigned by DBOPEN, or exceptionally, the parameters passed to procedure are incorrect in type, sequence or quantity.	Check application program's procedure call. Correct error in call.
-12	No covering lock	For DBUPDATE, DBPUT, and DBDELETE, data base has been opened in DBOPEN Mode 1 but there is no lock to cover entry in question. DBPUT or DBDELETE to master requires data set or data base be locked. In all other cases, entry, set, or data base can be locked.	Modify program to apply proper lock or change mode.

Table A-5. TurboIMAGE Library Procedure Calling Errors (Continued)

CCL	CONDITION	MEANING	ACTION
-14	Illegal intrinsic in current access mode	For DBPUT and DBDELETE data base has been opened in DBOPEN Mode 2, 5, 6, 7, or 8. These procedures may not be used with these access modes. For DBUPDATE, data base has been opened in DBOPEN Mode 5, 6, 7, or 8. DBUPDATE may not be used with these modes.	Modify program. Alter either mode or procedure call or notify current user that operation cannot be performed.
-21	Bad password	For DBOPEN, user class granted does not permit access to any data in data base. This is usually due to incorrect or null password.	Supply correct password.
-21	Bad data set reference	<p>For DBINFO (modes 104, 201, 202, 301, and 302), DBCLOSE, DBFIND, DBGET, DBUPDATE, DBPUT, DBDELETE, when data set reference is:</p> <ul style="list-style-type: none"> <li>•Numeric but out of range of the number of data sets in data base</li> <li>•An erroneous data set name</li> <li>•A reference to data set which is inaccessible to user class established when data base opened.</li> </ul> <p>For DBFIND, this error is also returned if referenced data set is a master. Erroneous data set name may arise when a terminating semicolon or blank is omitted.</p>	Check application program's procedure call. Correct error in call.
-21	Bad data item reference	<p>For DBINFO (modes 101, 102, and 204), data item reference is:</p> <ul style="list-style-type: none"> <li>•Numeric but out of range of the number of data items in data base</li> <li>•An erroneous data item name</li> <li>•A reference to data item which is inaccessible to user class established when data base opened.</li> </ul> <p>An erroneous data item name may arise when a terminating semicolon or blank is omitted.</p>	Check application program's procedure call. Correct error in call.

Table A-5. TurboIMAGE Library Procedure Calling Errors (Continued)

CCL	CONDITION	MEANING	ACTION
-23	Data set not writable	For DBPUT and DBDELETE, data base has been opened in DBOPEN Mode 1, 3, or 4 and user has read but not write access to the referenced data set.	Modify access mode set in procedure call or notify current user operation cannot be performed.
-24	Data set is an automatic master	For DBPUT, the referenced data set is an automatic master.	Modify data set name in call or in data set type in schema.
-31	Bad <i>mode</i>	This error occurs in all procedures when the <i>mode</i> parameter is invalid. For DBGET, <i>mode</i> is 7 or 8 and referenced data set is a detail, or <i>mode</i> is 5 or 6 and referenced data set is a detail without search items.	Correct mode in procedure call.
-32	Unobtainable <i>mode</i>	For DBOPEN, root file cannot be FOPENed with access options (AOPTIONS) required for the specified <i>mode</i> : Second word of calling program's status area is required AOPTIONS, and third word is the AOPTIONS granted to DBOPEN by MPE file system.  This error usually occurs either due to concurrent data base access by other users or due to MPE account or group security provisions.	See the <i>MPE Intrinsic Reference Manual</i> for meaning of AOPTIONS words.  Action depends on program's design. Normally notify user that requested access mode is not available.
-51	Bad <i>list</i> length	For DBGET, DBUPDATE, and DBPUT, the <i>list</i> is too long. This may occur if <i>list</i> is not terminated with a semicolon or blank. It may also occur for otherwise legitimate lists which are too long for TurboIMAGE's work area.  It will never occur for numeric lists.	Shorter <i>list</i> array contents. If necessary, change to numeric list.
-52	Bad <i>list</i> or bad <i>item</i>	For DBGET, DBUPDATE, or DBPUT, the <i>list</i> parameter is invalid. <i>list</i> either has a bad format or contains a data item reference which:	Check procedure call. Correct error in call or parameter.

Table A-5. TurboIMAGE Library Procedure Calling Errors (Continued)

CCL	CONDITION	MEANING	ACTION
-52	(continued)	<ul style="list-style-type: none"> <li>.Is out of range of the number of data items in the data base</li> <li>.Reference an inaccessible data item</li> <li>.Duplicates another reference in the list.</li> </ul> <p>For DBFIND, the <i>item</i> parameter contains data item reference which either:</p> <ul style="list-style-type: none"> <li>.Is out of range of the number of data items in the data base</li> <li>.Is not a search item for referenced data set.</li> </ul>	
-53	Missing search or sort item	For DBPUT, a search or sort item of referenced data set is not included in <i>list</i> parameter.	Check procedure call. Correct error in call or parameter.
-60	Illegal file equation on root file	When using a :FILE command with the data base name or a data-base-access file name, only the file designators and DEV= parameters are allowed.	Re-enter :FILE command without illegal parameters.
-80	Output Deferred not allowed with ILR enabled	DBCONTROL (mode 1) was used to request deferred output, but deferred output cannot be used when ILR or ROLLBACK are enabled. Output deferred is not initiated.	Do not use deferred output; or run DBUTIL and disable ILR or ROLLBACK.
-90	Root file bad: unrecognized state: %octal integer	For DBOPEN, this error is returned if the root file is in an unrecognized state. The octal integer represents an ASCII error code.	Restore old copy of the root file for the data base.
-91	Bad root modification level	For DBOPEN, the software version of the DBOPEN procedure is incompatible with version of schema processor which created root file.	Check with system manager that you have correct TurboIMAGE software. If necessary ask HP support personnel about conversion.
-92	Data base not created	For DBOPEN, the referenced data base has not yet been created and initialized by the DBUTIL program (in CREATE mode).	Run DBUTIL to create data base. Try application program again.

Table A-5. TurboIMAGE Library Procedure Calling Errors (Continued)

CCL	CONDITION	MEANING	ACTION
-94	Data base bad	For DBOPEN, referenced data base was damaged while being modified in output deferred mode.	Either DBLOAD from backup tape; or DBUNLOAD to ERASE data and then DBLOAD.
-95	Data base bad	For DBOPEN, data base was damaged by a file system failure, system failure, or TurboIMAGE abort while DBUTIL CREATE command was creating the data base.	Since data base was not created, run DBUTIL CREATE command to create the data base.
-96	Data base bad	For DBOPEN, data base was damaged by a file system failure, system failure, or TurboIMAGE abort while DBUTIL ERASE command was erasing the data base.	Since data was not erased, run DBUTIL ERASE command to erase the data from the data base.
-97	Data base bad - ILR enable in process (enable again)	DBOPEN attempted to open data base, but the prior ENABLE of ILR log file was not complete.	Run DBUTIL with ENABLE command to enable ILR log file.
-98	Data base bad - ILR disable in process (disable again)	DBOPEN attempted to open a data base, but prior disable of ILR log file was not complete.	Run DBUTIL with DISABLE command to disable ILR log file.
-100	DSOPEN failure	While executing a DBOPEN, TurboIMAGE has encountered a hardware failure trying to obtain a communications line.	Try opening the data base again. If error persists contact your HP Customer Engineer.
-101	DSCLOSE failure	This is an exceptional error returned by DBOPEN or DBCLOSE. It indicates a hardware or system software failure.	Notify system manager of problem.
-102	DSWRITE failure	A line failure has occurred while attempting an operation on a remote data base. May be returned by DBOPEN, DBFIND, DBGET, DBPUT, DBUPDATE, DBDELETE, DBLOCK, DBUNLOCK, DBINFO, or DBCLOSE.	Try calling the procedure again. If error persists notify system manager.

NOTE: Condition Codes -100 through -107 are communication errors.

For condition codes -100 through -102, third word of calling program's status area is MPE failure code returned by DSCHECK intrinsic.



Table A-5. TurboIMAGE Library Procedure Calling Errors (Continued)

CCL	CONDITION	MEANING	ACTION
-103	Remote stack too small	Command Interpreter on remote HP 3000 cannot obtain stack space necessary to execute a DBOPEN or DBLOCK.	Ask system manager of remote system to increase available stack size.
-104	Remote system does not support TurboIMAGE	Remote HP 3000 does not contain TurboIMAGE software. May be returned by DBOPEN.	Ask system manager of remote system to obtain and load TurboIMAGE software.
-105	MPE intrinsic GETDSEG failure on remote HP 3000	This is an exceptional error and is returned by DBOPEN on the remote system when it cannot obtain an extra data segment for the DBG. Second word of status array is size of data segment.	Notify system manager of problem.
-106	Remote data inconsistent	This is an exceptional error returned by same intrinsics as -102 (see above). It indicates a hardware or system software failure.	Notify system manager of problem.
-107	DS procedure call error	This is an exceptional error returned by same intrinsics as -102 (see above). It indicates a hardware or system software failure.	Notify system manager of problem.
-110	MPE OPENLOG intrinsic failure	This error may occur following a call to DBOPEN when a data base is enabled for logging. Refer to <i>MPE Intrinsics Manual</i> for listing of second values of status array and error messages.  OPENLOG returned error number NN to DBOPEN.	Notify system manager or HP support.
-111	MPE WRITELOG Intrinsic failure	When a data base is enabled for logging, this error may be returned by DBOPEN, DBCLOSE, DBPUT, DBUPDATE, DBDELETE, DBMEMO, DBBEGIN, DBEND. Refer to <i>MPE Intrinsics Manual</i> for listing of second values of status array and error messages.	Notify data base administrator.

NOTE: Condition Codes -100 through -107 are communication errors.

Table A-5. TurboIMAGE Library Procedure Calling Errors (Continued)

CCL	CONDITION	MEANING	ACTION
-112	MPE CLOSELOG intrinsic failure	When a data base is enabled for logging, this error may be returned by DBCLOSE. Refer to <i>MPE Intrinsic Manual</i> for listing of second second values of status array and error messages.  CLOSELOG returned error number NN to DBCLOSE.	Notify data base administrator.
-113	FLUSHLOG returned error number <i>nn</i> to DBEND	User called DBEND in mode 2 to write the logging buffer to disc, but the MPE logging facility returned an error.	Look up the error message in table 10-12 of the <i>MPE Intrinsic manual</i> .
-120	Not enough stack to perform DBLOCK	DBLOCK cannot obtain enough stack space.	:RUN or :PREP with STACK= or MAXDATA= to get more stack space.
-121	Descriptor count error	DBLOCK detected an error in the descriptor count (first word of qualifier array) in locking mode 5 or 6.	Count must be a positive integer.
-122	Descriptor list bad. Is not entirely within stack	DBLOCK checked the list and found that it did not lie between DL and the top of stack. May be caused by a bad <i>length</i> field.	Check length of each descriptor, and descriptor count.
-123	Illegal <i>relop</i> in a descriptor	DBLOCK encountered a <i>relop</i> field containing characters other than >=, <=, =Δ or Δ=.	Check contents of <i>qualifier</i> array.
-124	Descriptor too short. Must be greater than or equal to 9	DBLOCK encountered a lock descriptor less than 9 words long.	Check contents of <i>qualifier</i> array.
-125	Bad set name/number	DBLOCK <i>qualifier</i> array contains an invalid data set name or number. (Refer to error -21 for rules.)	Check contents of <i>qualifier</i> array. Be sure names delimited by semicolon or space if less than 16 bytes long.
-126	Bad item name/number	DBLOCK <i>qualifier</i> array contains an invalid data item name or number. (Refer to error -21 for rules.)	Same as above.

Table A-5. TurboIMAGE Library Procedure Calling Errors (Continued)

CCL	CONDITION	MEANING	ACTION
-127	Attempt to lock using a compound item	DBLOCK does not allow compound items in lock descriptors.	Modify locking strategy to lock on a non-compound item.
-128	Value field too short in a descriptor	A <i>value</i> field in a DBLOCK lock descriptor must be at least as long as the data item for which it is specified.	Perhaps the length word is too small in the descriptor.
-129	P-type item longer than P28 specified	DBLOCK does not allow P-type data items longer than 28 in lock descriptors (27 digits plus sign).	Modify locking strategy to lock on a different item.
-130	Illegal digit in a P-type value	DBLOCK has encountered a P-type value in a lock descriptor with an invalid packed decimal digit.	Check <i>qualifier</i> array contents to determine why data is invalid. Correct data representations are described in the <i>Machine Instruction Set Manual</i> , Section III.
-131	Lowercase character in type-U value	DBLOCK has encountered a lower-case character in a type-U value specified in a lock descriptor.	Same as above.
-132	Illegal digit in type Z value	Lock descriptor value specified to DBLOCK contains an invalid zoned decimal digit.	Same as above.
-133	Illegal sign in type Z value	Lock descriptor value specified to DBLOCK contains an invalid zoned decimal sign.	Same as above.
-134	Two descriptors conflict	DBLOCK has detected two lock descriptors in the same call that lock the same or part of the same data base entity. (For example, lock on set and data base in same request.)	Check <i>qualifier</i> array contents for conflicting lock descriptors.
-135	Second lock without CAP=MR	A second call to DBLOCK has been made without an intervening DBUNLOCK call and program does not have MR capability.	Read discussion of multiple calls to DBLOCK in Section 4 of this manual if you plan to use CAP=MR.

Table A-5. TurboIMAGE Library Procedure Calling Errors (Continued)

CCL	CONDITION	MEANING	ACTION
-136	Descriptor list exceeds 2047 words	DBLOCK allows at most 2047 word long lock descriptor lists ( <i>qualifier</i> array).	Change <i>qualifier</i> array contents so lock descriptor list is shorter.
-151	Text length greater than 512 bytes	Returned by DBBEGIN, DBEND, or DBMEMO.	Modify program.
-152	DBCLOSE called while a transaction is in process; DBBEGIN called while a transaction is in process	A transaction is in process.	Call DBEND before DBCLOSE. Call DBEND before the next DBBEGIN.
-153	DBEND called while no transaction in progress		DBBEGIN must precede a call to DBEND.
-160	File Conflict: a file already exists with the ILR log file name	DBUTIL ENABLE command was unable to build ILR log file because of a duplicate file name.	Purge or rename the duplicate file.
-161	Cannot check for an ILR log file conflict: file system error <i>nn</i>	DBUTIL ENABLE command was physically not able to check for file conflict; for example, a private volume was not mounted.	Check file system error number.
-162	Cannot build ILR log file: file system error <i>nn</i>	DBUTIL ENABLE command could not build the ILR log file because of a file system error.	Check file system error number.
-163	Cannot initialize ILR log file: file system error <i>nn</i>	DBUTIL ENABLE command or call to DBOPEN was not able to perform 2nd phase of initialization because of file system error.	Check file system error number.
-164	Cannot initialize ILR log header: file system error <i>nn</i>	DBUTIL ENABLE command or call to DBOPEN was not able to perform 1st phase of initialization because of file system error.	Check file system error number.
-165	Cannot save ILR log file: file system error <i>nn</i>	DBUTIL ENABLE command could not save the ILR log file as a permanent file because of a file system error.	Check file system error number.

Table A-5. TurboIMAGE Library Procedure Calling Errors (Continued)

CCL	CONDITION	MEANING	ACTION
-166	Cannot purge ILR log file: file system error <i>nn</i>	DBUTIL DISABLE, PURGE, or ERASE command could not purge an ILR log file because of a file system error.	Check file system error number.
-170	Cannot open ILR log file: file system error <i>nn</i>	DBUTIL DISABLE command or call to DBOPEN was not able to open ILR log file due to file system error.	Check file system error number.
-171	Cannot close ILR log file: file system error <i>nn</i>	DBUTIL DISABLE command or call to DBCLOSE was not able to close ILR log file due to file system error.	Check file system error number.
-172	Cannot read ILR log file: file system error <i>nn</i>	DBUTIL DISABLE command or call to DBOPEN was not able to read ILR log file due to file system error.	Check file system error number.
-180	ILR log invalid - internal file name does not match root file	DBUTIL DISABLE command or call to DBOPEN attempted to open ILR log file, but file names inconsistent. Unlikely to occur unless disc corrupted, or TurboIMAGE root file or ILR log file altered by privileged mode user.	Use DBRESTOR to restore data base.
-181	ILR log invalid - internal group name does not match root file	DBUTIL DISABLE command or call to DBOPEN attempted to open ILR log file, but group names do not match. (See -180 for possible cause.)	Use DBRESTOR to restore data base.
-182	ILR log invalid - internal account name does not match root file	DBUTIL DISABLE command or call to DBOPEN attempted to open ILR log file, but account names do not match. (See -180 for possible cause.)	Use DBRESTOR to restore data base.
-183	ILR log invalid - internal creation date does not match root file	DBUTIL DISABLE command or call to DBOPEN was unable to open ILR log file because its creation date was not the same as the creation date of the TurboIMAGE root file. Could be caused by storing the data base followed by a partial restore with MPE :RESTORE that excluded either the root file or the ILR log file. If there was an intervening ENABLE or DISABLE, the creation dates will not match and only a partial DBSTORE/DBRESTOR occurred.	To avoid this problem always store data base with DBSTORE and restore with DBRESTOR. Using these utilities assures that root file and ILR log file are consistent.

Table A-5. TurboIMAGE Library Procedure Calling Errors (Continued)

CCL	CONDITION	MEANING	ACTION
-184	ILR log invalid - internal last access date access data does not match root file	DBUTIL DISABLE command or call to DBOPEN was unable to open ILR log file because its last access date does not match the date in the TurboIMAGE root file. Possible caused by a store and then a partial restore (using MPE :RESTORE) of data base.	Always store data base with DBSTORE and restore with DBRESTOR. To assure consistent dates in root file and log file.
-185	Cannot get extra data segment of size %XXXXXX for ILR	When ILR is enabled TurboIMAGE allocates an extra data segment of %XXXXXX words; subsequently, this size is reduced to the actual size needed.	Run again.
-187	ILR is already enabled for this data base	DBUTIL ENABLE command attempted to enable a data base for ILR when ILR was already enabled.	Warning only.
-188	ILR is already disabled for this data base	DBUTIL DISABLE command attempted to disable a data base for ILR when ILR was already disabled.	Warning only.
-200	Data Base Language not system supported	DBOPEN attempted to open the data base and found that the language of the data base is not currently configured. The collating sequence of the language is unavailable; DBOPEN cannot open the data base.	Notify system manager.
-201	Native Language Support not installed	NLS/3000 internal structures have not been built at system start-up. The collating sequence table of the language of the data base is unavailable; DBOPEN cannot open the data base.	Notify system manager.
-202	MPE Native Language Support error #1 returned by NLINFO	The error number given was returned by NLS/3000 on a NLINFO call in DBOPEN.	Notify system manager.

Table A-6. TurboIMAGE Library Procedure Exceptional Conditions

CCG	CONDITION	MEANING	ACTION
0	Logging not enabled for this user	User has called DBBEGIN, DBMEMO, or DBEND but user was not currently logging to the data base.	In order to log, user must be changing the data base, and this user opened the data base for read only access (modes 5-8).
10	Beginning of file	DBGET has encountered beginning of file during a backward serial read. (There are no entries before the one previously accessed.)	Appropriate action depends on program design.
11	End of file	DBGET has encountered the end of file during a forward serial read. (There are no entries beyond the most recently accessed one.)	Same as above.
12	Directed beginning of file	DBGET has been called for a directed read with a record number less than 1.	Same as above.
13	Directed end of file	DBGET has been called for a directed read with a record number greater than the capacity of data set.	Same as above.
15	End of chain	DBGET has encountered end of chain during a forward chained read.	Same as above.
16	Data set full	DBPUT has discovered that data set is full.	Restructure data base with larger capacity for this data set. (See Section 7).
17	No master entry	DBFIND is unable to locate master data set entry (chain head) for specified detail data set's search item value.	Appropriate action depends on program design.
17	No entry	DBGET has been called to reread an entry, but no "current record" has been established or a call to DBFIND has set the current record to 0. DBGET is unable to locate master data set entry with specified search item value.	Appropriate action depends on program design.

NOTE: CCG means "Condition Code Greater than."

Table A-6. TurboIMAGE Library Procedure Exceptional Conditions (Continued)

CCG	CONDITION	MEANING	ACTION
17	(continued)	<p>DBGET has discovered that selected record is empty (does not contain an entry).</p> <p>DBUPDATE or DBDELETE was called when the "current record" was not established or was empty.</p>	<p>Appropriate action depends on program design.</p> <p>Same as above.</p>
18	Broken chain	<p>For DBGET with <i>mode</i> parameter equal to 5 (forward chained read), the "next entry" on current chain (as designated by internally maintained forward pointer for data set) contains backward pointer which does not point to most recently accessed entry (or zero for first member of a chain).</p> <p>For DBGET with <i>mode</i> parameter equal to 6 (backward chained read), the "next entry" on current chain in a backward direction (as designated by internally maintained backward pointer for data set) contains a forward pointer which does not point to most recently accessed entry (or zero for last entry in a chain).</p> <p>This error can arise in DBOPEN access modes 1, 5, and 6 because another user can make data base modifications concurrent with this user's accesses. When this error occurs, no data is moved to user's stack, although internal pointers maintained by TurboIMAGE in the DSCB are changed to new "offending" entry. (It becomes the current entry.) Note that this error check does not detect all structural changes. DBGET (mode 5 or 6) makes check only when preceding call on data set was successful DBFIND or DBGET.</p>	<p>Begin reading chain again from first or last entry.</p> <p>Same as above.</p>
20	Data base locked or contains locks	<p>DBLOCK Mode 2: The data base cannot be locked. Refer to value of status word three: if 0, data base already locked; if 1, data base contains locked sets or entries.</p>	<p>Appropriate action depends on program design.</p>



Table A-6. TurboIMAGE Library Procedure Exceptional Conditions (Continued)

CCG	CONDITION	MEANING	ACTION
20	(continued)	Mode 4, 6: The lock cannot be granted because the whole data base is already locked.	
22	Data set locked by another process	DBLOCK has detected that the data set is locked by another process or this process through a different access path. Returned in DBLOCK modes 4 and 6 only.	Appropriate action depends on program design.
23	Entries locked within set	DBLOCK has detected that data entries within requested data set are locked by another process or this process through a different access path. Returned in DBLOCK mode 4 only.	Same as above.
24	Item conflicts with current locks	Lock descriptors passed to DBLOCK specify a data item that is different than one used to set existing locks. TurboIMAGE allows no more than one data item per data set to be used at one time for locking purposes. Returned in DBLOCK mode 6 only.	Same as above.
25	Entries already locked	DBLOCK has detected that data entries requested to be locked are already locked by another process or this process through a different access path. Returned in DBLOCK mode 6 only.	Same as above.
41	Critical item	DBUPDATE has been asked to change value of search or sort item.	Correct call or notify user cannot update item.
42	Read only item	DBUPDATE has been asked to change value of a data item for which the user does not have write access.	Notify user, cannot update item. Or change password in program.
43	Duplicate search item value	DBPUT has been asked to insert data entry into a master data set with a search item value which already exists in data set.	Appropriate action depends on program design.
44	Chain head	DBDELETE has been asked to delete master data set entry which still has one or more non-empty chains.	Same as above.

Table A-6. TurboIMAGE Library Procedure Exceptional Conditions (Continued)

CCG	CONDITION	MEANING	ACTION
50	Buffer too small	Calling program's buffer (identified by buffer parameter) is too small to accommodate the amount of information that DBGET or DBINFO wishes to return without extending into the parameters area. This message is returned only if the buffer is the last item in the user's stack, and will overflow the stack boundaries.	Correct procedure call, or change buffer name or size.
51	Insufficient stack for BIMAGE temporary buffer	The stack size is not large enough for the temporary buffer used by the BASIC TurboIMAGE interface routines: XDBGET, XDBPUT, XDBUPDATE, and XDBINFO.	Ask system manager to increase maximum stack size.
52	Invalid number of parameters	Call to BIMAGE interface procedure has either too many or too few parameters.	Correct procedure call.
53	Invalid parameter	Call to BIMAGE interface procedure has an invalid parameter, for example, a parameter of wrong type.	Correct parameter name in call or parameter itself.
54	Status array too small	The status array specified in call to BIMAGE interface procedure has less than 10 elements.	Dimension status array with 10 elements.
60	Data base access disabled	DBOPEN has been called when the data base has been disabled for access.	Notify data base administrator.
61	Data base opened more than 63 times by same process.	DBOPEN has been called when the specified data base has already been opened 63 times by the same process.	
62	DBG Control Block is Full	<p>TurboIMAGE is unable to expand the trailer area in the DBG by enough to process a DBGET, DBPUT, or DBUPDATE list.</p> <p>Or, if DBLOCK returned condition:</p> <p>Lock area within DBG is full or system would not allow TurboIMAGE to expand DBG, or trailer could not be expanded to hold descriptor list.</p>	

Table A-6. TurboIMAGE Library Procedure Exceptional Conditions (Continued)

CCG	CONDITION	MEANING	ACTION
63	Bad DBG	Another process sharing data base has aborted because of logical inconsistency or internal error in TurboIMAGE, leaving DBG in potentially inconsistent state. All user accesses through existing DBG are disabled (except for DBCLOSE, mode 1). Returned by all intrinsics.	.
64	PCBX data segment area full	DBOPEN is unable to open data base because there is no room for DBG entry in PCBX area (MPE portion of data stack).	
66	The current DBG for the data base does not appear correct (TurboIMAGE internal error)		
1xx	Missing chain head	User has attempted to add detail data entry with a search item value that does not match any existing search item value in corresponding manual master data set. The digits xx identify the offending path number established by order in which their search items occur in set part of schema.	Notify user cannot add entry or add manual master entry and try again.
2xx	Full chain	User has attempted to add detail data entry to a chain which already contains the maximum allowable (2,147,483,647) entries. The digits xx identify the offending path number (as described in 1xx).	Consult with data base manager. May need to delete some entries from chain or restructure data base.
3xx	Full master	User has attempted to add detail data entry with a search item value in corresponding automatic master data set and new master entry cannot be created because automatic master data set is full. xx is offending path number, (as described in 1xx).	Restructure data base, increasing capacity of automatic master. (See Section 7).

Table A-7. TurboIMAGE Library Procedure Abort Condition Messages

MESSAGE	MEANING	ACTION
BUFFER SUPPLY CRISIS	Internal software inconsistency caused TurboIMAGE to mismanage its buffer space.	Notify the system manager and possibly HP support personnel of the error. Save FID information if it is printed.  You may need to perform data base recovery procedures (See Section 7).
CRITICAL LABEL READ ERROR ON data set	Procedure was unable to read the label of the data base file.	Same as above.
CRITICAL READ ERROR ON data set	Procedure encountered an MPE file read error while reading the data base file.	Same as above.
LABEL WRITE ERROR ON data set	Procedure was unable to complete the writing of a user label on a data base file.	Same as above.
LOST FREE SPACE IN data set	Internal software inconsistency caused unused record locations in a data set to become lost or unavailable. TurboIMAGE prints out file information display for the data set file.	Same as above.
NEGATIVE MOTIVE ATTEMPT: n	An internal software inconsistency has been found by TurboIMAGE while attempting to move data to or from a user's stack.	Same as above.
UNABLE TO CLOSE DATA SET	Procedure was unable to close a data base file.	Same as above.
UNABLE TO OPEN A data set	Procedure was unable to open a data base file.	Same as above.
WRITE ERROR ON data set	Procedure found an MPE file write error while writing into data base file.	Same as above.

Table A-7. IMAGE Library Procedure Abort Condition Msgs. (Cont.)

MESSAGE	MEANING	ACTION
WRONG NUMBER OF PARAMETERS OR BAD ADDRESS FOR PARAM #n	An address referenced by one of the parameters is not within the user's stack area in memory (roughly between the DL and Q registers). The n is a positional number of the parameter in the procedure's calling sequence. The first parameter is #1, the second #2, and so on.	Same as above.

## UTILITY ERROR MESSAGES

Two types of error messages are generated by the Utility programs. The first type consists of conditional errors associated with accessing the desired data base. For all utility programs except DBUTIL, errors generating these messages can be corrected without terminating the run if you are in session mode. DBUTIL errors of this type may terminate the program. After printing the error message the DBUTIL program reprompts with two greater than symbols (>>). Other utilities reprompt with the message: "WHICH DATA BASE?", allowing you to re-enter the data base reference. If you wish to terminate the utility program at this point you may type a carriage return with or without leading blanks. If you are in job mode, conditional errors always cause program termination. Conditional error messages and their meanings are described in Table A-8.

Unconditional errors occur in utility programs after successful execution has already begun. These errors usually cause program termination. The accompanying messages and their meanings are described in Table A-9.

Certain errors, external to the utilities, can result in utility program termination. Those caused by the operating system or initiated by the console operator are explained in Section 2 or Section 7 of the *Error Messages and Recovery Manual*. Errors initiated by the library procedures called by the utilities are described in Table A-4 through A-7 of this manual.

Table A-8. TurboIMAGE Utility Program Conditional Messages

MESSAGE	MEANING	ACTION
ACCESS DISABLED UNTIL RECOVERY HAS COMPLETED		Users will be able to access the data base after the recovery process has completed.
ACCT name more than 8 characters	The acctname part of DBNAME2 is too long.	Use the EDITOR to change the FILE command.
APPARENT BACKUP/LOGFILE MISMATCH	A call to DBPUT, DBUPDATE, or DBDELETE, that apparently succeeded when logging has failed during recovery.	Make sure the proper backup data base was restored and that the matching logfile was used for recovery.
AUTODEFER must be disabled before ILR can be enabled	AUTODEFER option in DBUTIL has been enabled for the data base. ROLLBACK or ILR cannot be enabled while output deferred mode is enabled.	Run DBUTIL and >>DISABLE AUTODEFER and >>ENABLE ILR or ROLLBACK for the data base.
BAD DATA BASE REFERENCE	Data base reference following the utility program :RUN command has a syntax error.	Correct the error in session mode or press return to terminate the program.
BAD MAINTENANCE WORD	User invoking the utility is not the creator of the referenced data base and has not supplied the correct maintenance word.	Same as above.
cannot open data base	The data base cannot be opened at this time. It may already be open in a mode that does not allow concurrent access.	Try the DBUTIL command again later.
CAN'T RECOVER DATA BASES IN STATISTICAL MODE	The CONTROL STATS command was entered subsequent to a RECOVER command.	Rerun DBRECOV without using statistical mode, or use statistical mode without specifying data bases or recovery
CAN'T OPEN DATA BASE- dberror message		Check the DBERROR message to determine the problem.

Table A-8. TurboIMAGE Utility Program Conditional Messages (Continued)

MESSAGE	MEANING	ACTION
CHECKSUM FAILURE ON LOG RECORD #n -EOF ASSUMED	May occur if the system failed and a warmstart recovery is not performed. Also may occur if a statistical recovery is used against an open logfile.	You may want to use a warmstart recovery in the future, but this is not absolutely necessary.
Data-base-access file name may not be qualified	The DBA file name can not be qualified with group and account.	Log on in the same group and account as the data-base-access file.
data-base-access file does not exist	No such file exists in the log-on user or account.	Check the files in group to determine the correct filename.
data-base-access filename too long	The filename has more than six characters.	Rename the file and try DBUTIL again.
DATA BASE ALREADY CREATED	You have invoked DBUTIL in the CREATE mode and specified the name of a data base which already exists.	In session mode, correct the error or press return to terminate program.
Data base already DISABLED for <i>option</i>	The <i>option</i> has already been disabled for this data base.	Press return to terminate the program.
Data base already ENABLED for <i>option</i>	The <i>option</i> has already been enabled for this data base.	Same as above.
Data base cannot be erased until recovery has completed		Press return to terminate program. After the recovery process has completed the user can run DBUTIL again.
Data base cannot be purged until recovery has completed		Same as above.

Table A-8. TurboIMAGE Utility Program Conditional Messages (Continued)

MESSAGE	MEANING	ACTION
DATA BASE CONTAINS INVALID LOGGING IDENTIFIER	The log identifier is the data base root file does not exist in the MPE table.	Use DBUTIL to set a valid log identifier into the data base.
DATA BASE HAS BEEN MODIFIED SINCE LAST RESTORE	The DBSTORE flag has not been reset. This may mean a backup data base has not been restored, or that there have been modifications since the last restoration.	Restore the backup data base, or use the CONTROL NOSTORE option and reenter the RECOVER command.
DATA BASE <i>n</i> HASN'T BEEN CREATED YET	The data base creator must run DBUTIL and CREATE the data base.	In session mode, correct the error or press return to terminate the program.
DATA BASE IN USE	Utility program cannot get exclusive access to the referenced data base due to other current users.	In session mode, press return to terminate the program.
DATA BASE IS NOT EMPTY. LANGUAGE CANNOT BE CHANGED	The language can be changed only on a virgin root file or an empty data base.	Same as above.
DATA BASE IS REMOTE	You must be logged on to the same group and account containing the root file to use DBUTIL.	Do a remote logon and run DBUTIL from your remote session.



Table A-8. TurboIMAGE Utility Program Conditional Messages (Continued)

MESSAGE	MEANING	ACTION
DATA BASE NAME MAY NOT BE QUALIFIED	The data base name can not be qualified with group and account.	Log on to the same group and account as the data base and reenter the command.
DATA BASE NAME TOO LONG	The data base name you specified has more than six characters.	Try the command again with the correct name.
DATA BASE NOT ENABLED FOR RECOVERY	The recovery flag in the data base root file is disabled.	Be sure you are running recovery against the appropriately restored data base. You can override the disable flag with DBUTIL if necessary.
DATA BASE LANGUAGE NOT SYSTEM SUPPORTED	The language of the data base is not currently configured on your system.	Notify the system manager.
data base or access file does not exist	No such file exists in log-on group or account.	Check the files in your group to find the correct file name.
DATA BASE OR FILE ACCESS NAME REQUIRED	The command must include the data base or access file name.	Reenter the command.
DATA BASE OR FILE ACCESS NAME TOO LONG	File name has more than six characters.	Reenter the command with the correct name.
DATA BASE REQUIRES CREATION	The data base creator must run the DBUTIL program in CREATE mode prior to executing DBUNLOAD, DBLOAD, or DBUTIL in ERASE mode.	In session mode, correct the error or press return to terminate the program.
DATA BASE TABLE OVERFLOW	You tried to recover more than 20 data bases at the same time.	Run DBRECOV for the remaining data bases.

Table A-8. TurboIMAGE Utility Program Conditional Messages (Continued)

MESSAGE	MEANING	ACTION
DATA SET <i>n</i> IS MISSING	The data set may have been purged, or was not restored from a STORE or SYSDUMP.	Contact your data base administrator or system manager.
dbname DBSTORE OF time, date REQUIRED	DBSTORE timestamp in data base does not match those in DBOPEN records.	Restore the proper data base or use the proper logfile. You can also use the NOSTAMP option in recovery.
DBLOAD to different Data Base Name/Maintenance Word is not allowed	The DBLOAD data base name and maintenance word must be the same as those used for DBUNLOAD.	Use a data base name and maintenance word which were used for the DBUNLOAD.
DBLOAD to different group or account is not allowed.	DBLOAD must be done from the same group and account as the DBUNLOAD file.	Log on to the same group and account from which the DBUNLOAD was done. Then use DBLOAD again.
DUPLICATE FILE NAME	The required file name is already assigned to another file. Example: if data base ORDERS requires six data sets (ORDERS01-ORDERS06), then a previously defined file ORDERS03 will trigger this message. This can occur if a data base is not purged before running DBRESTOR.	In session mode correct the error or press return to terminate the program.
DBNAME1 MORE THAN SIX CHARACTERS	dbname1 is too long.	Use EDITOR to change the FILE command.
'DEV' MISSING	Either the keyword DEV or the following "=" was not found.	Same as above.
DEVICE NOT 'DISC'	The string following the "#" sign was not "DISC".	Same as above.

Table A-8. TurboIMAGE Utility Program Conditional Messages (Continued)

MESSAGE	MEANING	ACTION
DSDEVICE DOESN'T MATCH FILE COMMAND	The dsdevice name in the DSLINE command was not the same as the dsdevice name in the FILE command.	Use Editor to change the command which is in error.
DSDEVICE NAME MORE THAN 8 CHARACTERS	DSDEVICE name is too long.	Same as above.
EMBEDDED BLANK IN DBNAME2	One of the periods separating username from groupname or groupname from acctname was preceded or followed by a blank.	Use EDITOR to change the file command.
ERROR READING ROOT FILE LABEL	DBUTIL is unable to read the root file.	Contact your HP systems engineer.
ERROR READING ROOT FILE RECORD	DBUTIL is unable to read a root file record.	Same as above.
ERROR WRITING ROOT FILE LABEL	DBUTIL found an error while writing the root file label.	Same as above.
ERROR WRITING ROOT FILE RECORD	DBUTIL found an error while writing to a root file record.	Same as above.
EXCEEDS ACCOUNT DISC SPACE	Amount of space required by the data base plus the space already assigned to other files exceeds the disc space available to the account.	Request system manager to increase account's disc space.
EXCEEDS GROUP DISC SPACE	The amount of disc space required by the data base plus the space assigned to other files in the group exceeds the amount of disc space available to the account.	Request the system manager to increase the group's disc space.

Table A-8. IMAGE Utility Program Conditional Messages (Continued)

MESSAGE	MEANING	ACTION
FCHECK FAILURE FCLOSE FAILURE FCONTROL FAILURE FENTRY FAILURE FGETINFO FAILURE	These are exceptional errors indicating a hardware or software failure.	Notify your system manager of the failure.
FILE CODE IS NOT 0.#	The data base access file to be activated has a file code of other than 0.	Be sure you have the correct EDITOR file.
FILE EQUATES ARE ILLEGAL FOR DATA BASE AND DATA BASE ACCESS FILE	DBUTIL does not allow you to equate the name of the data base or data base access file to another file with the :FILE command.	Use the :RESET command to cancel the :FILE command. You can either break and resume execution or exit DBUTIL and run it again.
FILE filename RECSIZE TOO SMALL: LOG(S) TRUNCATED.	User recovery file size is too small to hold the largest log record.	Increase record size or use variable length records.
FILE NAME MAY NOT BE QUALIFIED	The file name may not be qualified with group and account.	Log on to the same group and account and reenter the command.
FILE NAME MORE THAN SIX CHARACTERS	The filename part of dbname2 is too long.	Use the EDITOR to change the FILE command.
FOPEN FAILURE ON logfilename:ferrmsg	You cannot open the logfile.	Examine the ferrmsg to determine the cause.
FREAD ERROR ON ASCII ACCESS FILE	Exceptional errors indicate a hardware or software failure.	Notify the system manager of the error.
FWRITE ERROR ON filename:ferrmsg	FWRITE failure on user recovery file.	Check ferrmsg to determine the cause.
GROUP NAME MORE THAN 8 CHARACTERS	The groupname part of dbname2 is too long.	Use the EDITOR to change the FILE command.

Table A-8. TurboIMAGE Utility Program Conditional Messages (Continued)

MESSAGE	MEANING	ACTION
ILR must be disabled before AUTODEFER can be enabled	Output deferred option (AUTODEFER) can not be used when the data base has been enabled for ILR or ROLLBACK.	Disable ILR or ROLLBACK and enable AUTODEFER using DBUTIL. (Refer to Sections 7 and 8 for more information on AUTODEFER.)
INCOMPLETE PURGE RELEASE SECURE	The operation was interrupted while in process.	Contact your system manager.
INCOMPLETE DBNAME2	The filename or or groupname is followed by a period and the rest of the record is empty. Or the next character is a semi-colon (;).	Use the EDITOR to change the FILE command.
INCOMPLETE LOCAL PART	The period following the local user name was missing, or the account name was missing, or a comma followed the account name and the group name was missing.	Use the EDITOR to change the HELLO command.
INCOMPLETE REMOTE PART	The period following the rusername was missing, or the raccounname was missing, or the raccounname was followed by a comma and the rgrouppname was missing.	Same as above.
INSUFFICIENT DISC SPACE	The amount of disc space required for the data base is not available on the system.	Consult with the system manager about disc space requirements.
INSUFFICIENT VIRTUAL MEMORY	There is not enough virtual memory available to open and access the database.	Try running the utility later when the system is not as busy.

Table A-8. TurboIMAGE Utility Program Conditional Messages (Continued)

MESSAGE	MEANING	ACTION
INVALID ACCESS CLASS NUMBER	The access class should be a number between 1 and 63.	Reenter with a valid access class number.
INVALID CHARACTER IN PASSWORD		Check the password specified and reenter the command.
INVALID COMMAND		Check spelling or see manual for legal commands.
INVALID CONTENTS OF ASCII ACCESS FILE activation failed deactivation failed	The file specified is either, 1) not a data-base-access file, or 2) it does not contain the valid format and parameters.	1) Verify the correct data-base-access file, or 2) use EDITOR to make the required changes and try accessing again.
INVALID DATA BASE CONTROL BLOCK	TurboIMAGE encountered an inconsistency in the control blocks.	Contact your system engineer.
INVALID DATA BASE ACCESS FILE NAME	The file name or data base name must be 1 to 6 characters starting with a letter.	Reenter the command with the correct name.

Table A-8. TurboIMAGE Utility Program Conditional Messages (Continued)

MESSAGE	MEANING	ACTION
INVALID DATA BASE NAME	Same as above.	Reenter the command with the correct name.
INVALID DATA BASE NAME OR ACCESS FILE NAME	Same as above.	Same as above.
INVALID DELIMITER	The command or a space is incorrectly positioned.	Use <b>HELP</b> to check command syntax and reenter the command.
INVALID IMAGE LOG RECORD DETECTED	DBRECOV encountered an undefined log record code.	Notify the system manager and HP support personnel.
INVALID MAINTENANCE WORD		Check the maintenance word specified and reenter the command.
INVALID NUMBER OF BUFFERS SPECIFIED	The number of buffers must be between 4 and 255.	Reenter <b>DBUTIL &gt;&gt;SET</b> command with a valid number.
INVALID NUMBER OF USERS SPECIFIED	The minimum number of users allowed is 1 and the maximum is 120. The ranges of users specified must be in ascending order.	Same as above.
INVALID SUBSYSTEM ACCESS FLAG IN ROOT FILE	TurboIMAGE detected an invalid value for the subsystem access in the root file. Valid accesses are <b>READ</b> , <b>R/W</b> , and <b>NONE</b> .	Reset the subsystem access flag using <b>DBUTIL &gt;&gt;SET</b> command.

Table A-8. TurboIMAGE Utility Program Conditional Messages (Continued)

MESSAGE	MEANING	ACTION
INVALID LANGUAGE	Language name or number has invalid characters.	Retype the correct language name or number.
INVALID PARAMETER	You specified an incorrect parameter.	Use HELP to check the command format and reenter the command.
INVALID PASSWORD FOR DATA BASE LOGGING IDENTIFIER		Use DBUTIL to set the valid log identifier password into the data base.
INVALID TRANSACTION SEQUENCE DETECTED	The transaction numbers are not consecutive.	Notify your system manager and HP support personnel.
INVALID (x) IN COLUMN y	The x represents the invalid character found in column y.	Use the EDITOR to change the data base access file record which returns the error message.
LANGUAGE MUST NOT BE LONGER THAN 16 CHARACTERS	The language name is too long and therefore, must be incorrect.	Retype the correct language name
LANGUAGE NOT SUPPORTED	The language is not supported on your system or is not a valid language name.	Check with your system manager for the configuration of the language, or enter a valid language name or number.
LESS THAN 3 RECORDS IN FILE!	The data base access file does not contain a FILE, DSLINE, and HELLO command.	Use the EDITOR to create the missing records.
LOCAL ACCT NAME IS TOO LONG	acctname is more than 8 characters.	Use the EDITOR to change the lacctname.



Table A-8. TurboIMAGE Utility Program Conditional Messages (Continued)

MESSAGE	MEANING	ACTION
LOCAL GROUP NAME IS TOO LONG	lgrouppname has more than 8 characters.	Use the EDITOR to change the lgrouppname.
LOCAL USER NAME TOO LONG	lusername has more than 8 characters.	Use EDITOR to change the lusername.
LOG BUFFER OVERFLOW	There is insufficient buffer space to build the log record.	Notify the system manager and HP support personnel.
LOGFILE AND DATA BASE LOGID'S DO NOT MATCH	The restored data base does not match the logfile.	Be sure the logfile and the backup data base are properly attached.
LOGFILE IS EMPTY	The logfile has no records.	Be sure you have correctly identified the logfile.
LOGFILE RECORD SIZE IS IMPROPER	The records are not 256 bytes with a fixed length.	Be sure this file is actually a logfile.
LOGID:logid IS INVALID	The data base logid has been removed from the MPE table.	Set the data base to an existing logid, or use the GETLOG command to establish the data base logid on the MPE table.

Table A-8. TurboIMAGE Utility Program Conditional Messages (Continued)

MESSAGE	MEANING	ACTION
LOGID MUST NOT BE LONGER THAN 8 CHARACTERS	The logid name is too long therefore, it is incorrect.	Retype the correct logid to set into the database.
LOGID PASSWORD IS INCORRECT	The password has been altered, or you are using the wrong password.	Set the logid with the correct password into the data base.
LOGGING IS ENABLED, CAN NOT CLEAR LOGID	The logid can not be changed while logging is in process in order to insure validity of recovery and logging.	Change the logid after the recovery process has completed.
MAINTENANCE WORD CANNOT BE CHANGED UNTIL RECOVERY HAS COMPLETED		After the recovery process has completed use the DBUTIL >>SET command to change the maintenance word.
MAINTENANCE WORD REQUIRED	The user invoking the utility is not the creator of the referenced data base and has not supplied a maintenance word.	If DBUTIL, reenter the command with the maintenance word. Otherwise correct the error or press return to terminate the program.
MAINTENANCE WORD TOO LONG	The specified maintenance word has more than 8 characters.	Reenter the command with the correct maintenance word.
max errorcount exceeded	Too many errors occurred in batch execution.	Correct the errors or use CONTROL ERRORS= command to ignore the errors.
MISSING!!	The key word FILE is not found in the first record; or the keyword DSLINE is not found in the second record; or the keyword HELLO is not found in any of the remaining records.	Use the EDITOR to modify the record in the data base access file that returned the error message.

Table A-8. TurboIMAGE Utility Program Conditional Messages (Continued)

MESSAGE	MEANING	ACTION
MISSING DBNAME 1	An equals sign (=), semi-colon (;), or end of line followed the keyword FILE.	Use the EDITOR to change the FILE command.
MISSING DBNAME 2	dbname1= followed by a semi-colon (;) or end of line.	Use EDITOR to change the FILE command.
MISSING DSDEVICE	DEV= was followed by a semi-colon (;), #, or the rest of the record was blank.	Use the EDITOR to change the FILE command.
MISSING LOCAL PART	No characters preceded the =HELLO.	Use the EDITOR to change the HELLO command.
MISSING REMOTE PART	The remote-id-sequence of the data-base-access file is missing.	Use the EDITOR to add the remote part.
MISSING REMOTE PASWD	HELLO was not followed by text, or was followed by a semi-colon (;).	Using the EDITOR, change the HELLO command.
MISSING SEMI-COLON	DEV= was not preceeded by a semi-colon (;)	Use the EDITOR to change the file command.
MISSING # SIGN	dsdevice was not followed by a #.	Same as above.
MULTIPLE LOG IDENTIFIERS NOT ALLOWED	You tried to simultaneously recover data bases that logged to different logfiles.	Run DBRECOV once for each logfile.
NLS RELATED ERROR	An error was returned by NLS/3000 on a DBOPEN on the data base.	Notify the system manager.

Table A-8. TurboIMAGE Utility Program Conditional Messages (Continued)

MESSAGE	MEANING	ACTION
NLINFO FAILURE	An error was returned by NLS/3000	Notify system manager.
NO DATA BASE HAS BEEN SPECIFIED FOR RECOVERY	RUN command has been entered and no data bases have been specified for recovery.	Use the RECOVER command to specify data base(s) or EXIT to terminate.
NON-CREATOR ACCESS NOT PERMITTED	You must be the data base creator to perform this function.	
NONEXISTENT LOGID	You are trying to set an unrecognized logid into the data base, or the data base logid has been removed from MPE.	Use GETLOG to put the data base logid into the MPE Table, or set an existing logid into the data base.
NO SUCH DATA BASE	Specified data base does not exist in users log on group.	Check base name and log on account and group. Press return to terminate the program.
NOT A DATA BASE ACCESS FILE	The specified file is not a data base access file.	Check the file name and try the command again.
NOT A DATA BASE OR DATA BASE ACCESS FILE		Check the data base or file name and try the command again.
NOT A DATA BASE ROOT FILE		Check the specified data base name and try the command again.
NOT A PRIVILEGED DATA BASE ACCESS FILE	The data base access file has not been activated.	Check the command and reenter it correctly.
NOT ALLOWED; MUST BE CREATOR	The user invoking the utility is not the creator of the data base and the data base has no maintenance word.	Log on with correct user name, account and group.

Table A-8. TurboIMAGE Utility Program Conditional Messages (Continued)

MESSAGE	MEANING	ACTION
NOT AN UNPRIVILEGED DATA BASE ACCESS FILE	The data base access file is already activated.	Check the command and reenter it if necessary.
OUTMODED ROOT	The root file of the specified data base corresponds to a different version of IMAGE software and is not compatible with the utility program currently executing.	Consult with your system manager to verify the correct version of the software. Ask HP support personnel about conversion process.
PARAMETER EXPECTED	The command you specified calls for another parameter.	Use <b>HELP</b> to find the correct command syntax.
PARAMETER MUST BE A COMMAND	Only command names may be specified with the <b>HELP</b> command.	Reenter the command.
PARAMETER SPECIFIED TWICE	The command specified requires only one parameter entry.	Reenter the command.
PASSWORD IS INCORRECT	The logid password in the data base is not the same as the password in the MPE table.	Set the logid and correct password into the data base. Or, use the <b>MPE ALTLOG</b> command to alter the password in the MPE table so it matches the one in the data base.
PASSWORD MUST NOT BE LONGER THAN 8 CHARACTERS	The password is too long so it is incorrect.	Retype the correct logid and password.
PREMATURE EOF ON ASCII ACCESS FILE	DBUTIL encountered an end of file mark before at least one =HELLO record in the data base access file.	Use the <b>EDITOR</b> to change the ASCII file.
PROCESS TABLE OVERFLOW	At some point more than 128 processes (or 180 in B.03.06 or later version) were logging at the same time.	Notify your system manager and HP support personnel.

Table A-8. TurboIMAGE Utility Program Conditional Messages (Continued)

MESSAGE	MEANING	ACTION
RECORD SIZE EXCEEDS 128 CHARACTERS!	The longest record in the data base access file exceeds the allowable length.	Use the EDITOR to modify the data base access file.
RECORDS ARE NUMBERED!	The data base access file is numbered.	Use the editor to keep the file unnumbered.
RECORD TABLE OVERFLOW	More than 1000 detail records have been added and they have been given relative record numbers different from those originally assigned. Wrong data base may have been restored, or not all modifications to the data base were logged.	Run DBRECOV without the NOABORTS option or recover to a point before the occurrence of the error. OR restore the correct data base, or restore data base and perform DBUNLOAD/DBLOAD.
RECOVERY TABLE OVERFLOW	You tried to specify more than 100 user recovery files.	Reduce the number of recovery files needed.
REMOTE ACCT name TOO LONG	racctname is more than 8 characters long.	Use the EDITOR to change the HELLO command.
REMOTE ACCT PASSWORD TOO LONG	rupasw is more than 8 characters.	Use the Editor to change the HELLO command.
REMOTE GROUP NAME TOO LONG	rgroupname is more than 8 characters.	Same as above.
REMOTE GROUP PASSWORD TOO LONG	rgpasw is more than 8 characters.	Same as above.
REMOTE USER NAME TOO LONG	rusername is more than 8 characters.	Same as above.

Table A-8. TurboIMAGE Utility Program Conditional Messages (Continued)

MESSAGE	MEANING	ACTION
REMOTE USER PASSWORD TOO LONG	rupasw is more than 8 characters.	Use the EDITOR to change the HELLO command.
ROLLBACK must be disabled before ILR can be disabled		Use DBUTIL >>DISABLE the data base for ROLLBACK. ILR will be automatically disabled.
ROLLBACK must be disabled before LOGGING can be disabled		Use DBUTIL >>DISABLE the data base for ROLLBACK, then LOGGING.
ROOT FILE DOES NOT EXIST	A root file with the name of the specified data base does not exist in the log on group and account.	Check the data base name you specified and reenter the command.
SEQUENCE ERROR ON LOG RECORD #n - EOF ASSUMED	Log record numbers are out of sequence.	Notify the system manager and HP support personnel.
STAGING FILE OVERFLOW	The staging file is not large enough.	Use a file equation to increase the file size.
TIMESTAMP OUT OF SEQUENCE ON LOG RECORD #n - EOF ASSUMED.	Log record timestamps are out of sequence.	Notify the system manager and HP support personnel.
TOO MANY PARAMETERS		Reenter the command with the correct number of parameters.
UNKNOWN COMMAND, TRY HELP	DBUTIL does not recognize the command you specified.	Enter the HELP command to get the list of DBUTIL commands.

Table A-8. TurboIMAGE Utility Program Conditional Messages (Continued)

MESSAGE	MEANING	ACTION
USER IS NOT CREATOR OF LOGGING IDENTIFIER		Log on to the system as the user who created the log identifier or as a user with system manager capability.
WARNING: DATA BASE MODIFIED AND NOT DBSTORED	The data base has been enabled for logging but a backup copy has not been made by executing DBSTORE.	Execute DBSTORE to make a backup copy of the data base.
WARNING: THE LANGUAGE OF THE DATA BASE IS DIFFERENT FROM THE LANGUAGE FOUND ON THE DBLOAD MEDIA. CONTINUE DBLOAD OPERATION / (Y/N):	<p>The user has changed the language of the data base between DBUNLOAD and DBLOAD. DBLOAD wants the user to be aware of potential differences in sorted chains of the collating sequence of the two languages. (The language of the data base on disc and the data base on tape are different).</p> <p>In session mode you are asked if you want to continue the operation. If you are in job mode, DBLOAD will terminate execution.</p>	After looking at the information DBLOAD returns and what the result on the sorted chains in the data base, continue the operation by answering YES.
WARNING: THE NUMBER OF DATA SETS DEFINED IN THE SCHEMA IS LESS THAN (OR GREATER THAN) THE NUMBER OF DATA SETS FOUND ON THE DBLOAD MEDIA.	You have added or deleted data sets from the data base between a DBUNLOAD and a DBLOAD. DBLOAD will continue with the load, but data may be lost or put into the wrong sets due to an invalid set change.	Stop the DBLOAD, correct the SCHEMA and DBLOAD again. Or, if you are sure the data base is not corrupt, allow DBLOAD to continue.



Table A-9. TurboIMAGE Utility Program Unconditional Messages

MESSAGE	MEANING	ACTION
AUTOMATIC MASTER IS FULL ON PATH #n	DBLOAD is unable to load a detail entry because the automatic master set associated with path n of the detail set is full.	Recreate the root file with a larger capacity for automatic master. Rerun the necessary utilities.
***BAD DATA BASE***	This message is issued by DBSTORE, DBRESTOR, and DBUNLOAD. It means the data base is flagged "bad" because of a known structural error due to an abnormal termination or to a system crash during DBLOAD or some other "output deferred" operation. The current operation (DBSTORE, DBRESTOR, DBUNLOAD) continues to function normally. DBLOAD additionally prints the message "SERIAL UNLOAD FOLLOWS" and automatically operates in serial mode. The data base on disc retains its "bad" flag and cannot be accessed through DBOPEN.	The data base is not usable in its current state. Purge it and restore a backup copy, or erase it and then load a tape or serial disc written by DBUNLOAD or another external copy of the data.
BROKEN FILE EQUATION CHAIN FOR TAPE FILE	Issued by DBSTORE if, in a chain of file equations, the actual device designator cannot be found.	Check your :FILE commands and reenter them correctly before running DBSTORE.
CANNOT GET EXTRA DATA SEGMENT NECESSARY FOR RESTORE OPERATION	DBRESTOR was unable to get the extra data segment it needed for the buffers used in the restore operation.	

Table A-9. TurboIMAGE Utility Program Unconditional Messages (Continued)

MESSAGE	MEANING	ACTION
CANNOT OPEN TERMINAL, TERMINATING	DBUTIL is unable to access the terminal.	Call HP systems engineer.
Can't create new copy of file <i>x</i> (FS error #)	MPE intrinsic FOPEN failed while creating a temporary file.	Check the file system error. Possible reasons: the specified device does not exist, or there may be insufficient space on the device specified.
Can't get specifications via FGETINFO for this file <i>x</i> (FS error #)	MPE intrinsic FGETINFO failed. DBUTIL can not get information for the specified file.	Check the file system error to determine cause.
Can't open file <i>x</i> (FS error #)	MPE intrinsic FOPEN failed.	Same as above.
Can't purge old copy of file <i>x</i> (FS error #)	MPE intrinsic FCLOSE failed while purging the "old" file.	Same as above.
Can't reset move flag for file <i>x</i> in the root file	Error occurred while writing root file label to reset the data base condition word.	Contact your system manager.
Can't save new copy of file <i>x</i> (FS error #)	MPE intrinsic FCLOSE failed while saving the temporary file.	Check file system error to determine the cause.
Can't set move flag for file <i>x</i> in root file	DBUTIL set the root file flag, and an error occurred while writing the root file label.	Contact your system manager.
CHAIN IS FULL ON PATH # <i>n</i>	DBLOAD is unable to load a detail entry because the chain count for path number <i>n</i> of detail set exceeds $2^{31} - 1$ (or 2,147,483,647 entries).	Delete some of the entries and reload or change the data base design.

Table A-9. TurboIMAGE Utility Program Unconditional Messages (Continued)

MESSAGE	MEANING	ACTION
<b>***COPY FAILED***</b>	An error occurred while copying the file. DBUTIL has encountered a problem reading or writing the files. (It is possible the disc pack is bad.)	Restore the data base and try the move again. Contact your system manager if the copy fails again.
Couldn't open the data base	There is no root file for the specified file. It is possible that the file specified is not part of the data base.	Check the file or data base. Try the operation again with the correct file name.
Data base state does not allow MOVE to be done	The data base is in a state in which MOVE cannot be performed.	Check the data base. If the data base has not been created, run DBUTIL >>CREATE. If the data base is damaged run recovery.
DATA SET FULL	Data set currently being loaded is full.	Recreate root file, increase the data set's capacity and run the utilities again.
<b>***DBSTORE FAILED - NO DATA BASE STORED***</b>	A file error or other message follows explaining the problem.	Contact your HP systems engineer.
EOF SEEN, PROGRAM TERMINATING	A :EOF has been entered.	Run DBUTIL again.
FGETINFO failure	DBUTIL received an error when calling MPE FGETINFO.	Contact your system manager.
FILE EQUATE FOR DBSTORE DBRESTOR DBLOAD DBUNLOAD ONLY MAY USE DEV	If you specify an input/output file with a :FILE command for any of these utility programs, only the file designators and DEV= parameters are allowed.	Enter :FILE command again and rerun program.

Table A-9. TurboIMAGE Utility Program Unconditional Messages (Continued)

MESSAGE	MEANING	ACTION
FILE NOT ON TAPE	Issued by DBRESTOR if the data base to be restored is not on the tape.	Check your :FILE command and/or the tape you mounted.
File <i>x</i> already resides on device <i>x</i>	The file specified already resides on the device.	
FREADDIR failure	This is an MPE exceptional failure and is returned when DBUTIL calls FREADDIR.	Contact your system manager.
HARD TERMINAL READ ERROR, TERMINATING	DBUTIL cannot read input from terminal.	Notify HP systems engineer
***INVALID SET COUNT***	TurboIMAGE found an inconsistency in the data set count.	Same as above.
MOVE of file <i>x</i> not allowed: file is not correct type (file code #)	The file specified is not a data base file.	Try the MOVE operation again with the correct file name.
NO MANUAL ENTRY FOR DETAIL ON PATH # <i>n</i>	DBLOAD is attempting to load detail data set entry <i>n</i> which is the number of the detail data set path referencing the manual master in question.	Add entry to manual master with application program or QUERY. Run DBLOAD again.
RECSIZE MUST BE MODULO 256 <=xxxx FOR THIS DEVICE	Issued if REC= recsize parameter for DBSTORE specified a record size not modulo 256 words, or greater than the configured record size of the device.	Check the REC= recsize parameter and correct it before running DBSTORE again.

**Table A-9. TurboIMAGE Utility Program Unconditional Messages (Continued)**

MESSAGE	MEANING	ACTION
UNABLE TO CONTINUE	DBUTIL program (operating in PURGE mode) cannot continue execution due to exceptional error in file system. This information is followed by MPE file information display.	Save the file information. Consult with system manager and HP support personnel.
Unable to obtain file label information for file <i>x</i>	MPE internal procedure failed. DBUTIL cannot obtain information for the specified file.	Contact your system manager.
Unexpected root file state	The MOVE process was unable to reset the data base flag. The data base may be in an inconsistent state.	Same as above.

A set of messages may be returned when an unusual condition causes TurboIMAGE to fail while an TurboIMAGE utility is executing. In each of the following messages in Table A-10, the value *xxxxx* is the octal location where TurboIMAGE failed.

Table A-10. TurboIMAGE Extended Utility Program Unconditional Messages

#	MESSAGE	MEANING	ACTION
-3	TurboIMAGE failure due to FREADDIR failure on root file at xxxxx	Root file was not readable. Problem may be in hardware.	Notify the system manager of the error.
-4	TurboIMAGE failure due to FREADLABEL failure on root file at xxxxx	User's label was not readable. Problem may be in hardware.	Same as above.
-64	SET NUMBERS ARE OUT OF SEQUENCE. TurboIMAGE FAILED AT xxxxx	The next set number on the backup volume was not what DBLOAD expected. Status word 0 has the expected set number; status word 1 has the next set number on the backup volume.	The schema may have been changed since the last DBUNLOAD and this DBLOAD. Add any new set the the end of the schema.
-66	BLOCK NUMBERS ARE OUT OF SEQUENCE. TurboIMAGE FAILED AT xxxxx.	The next block on the backup volume was not what DBLOAD expected. Status words 0,1 have the expected block number; words 2,3 have the actual next block number on the backup volume; word 4 has the set number on the backup volume.	An error for a known value that was written to tape with FWRITE was found. There may be a defect in the tape or the tape drive.
-70	ENTRY NUMBERS ARE OUT OF SEQUENCE. TurboIMAGE FAILED AT xxxxx	Same as above	Same as above.
-74	THE RECORD JUST READ IS UNRECOGNIZABLE TurboIMAGE FAILED AT xxxxx	The record just read from the backup volume was not what DBLOAD expected. The record could be an EOF, EOT, etc. The status word 0 has one of the following codes for the record expected: 0=tape head, 1=data file head, 2= data file block, 3=data file end, 4=tape end.	Same as above.

Table A-10. TurboIMAGE Extended Utility Program Unconditional Messages (Continued)

#	MESSAGE	MEANING	ACTION
97	ZSIZE ERROR WHILE CONTRACTING AT xxxx	A TurboIMAGE utility called ZSIZE to contract the Z to DB area of the stack. xxxxx is the octal location where TurboIMAGE failed.	
98	ZSIZE ERROR WHILE EXPANDING AT xxxxx	A TurboIMAGE utility called ZSIZE to expand the Z to DB area of the stack. xxxxx is the octal location where TurboIMAGE failed.	
101	DBRESTOR FAILURE IN DBRESTOR AT xxxxx	The DBRESTOR utility called MPE RESTORE which encountered problems while restoring the data base. A detailed error message should follow this message.	
401	DBOPEN FAILURE AT xxxxx	Error messages 401 through 414 are all TurboIMAGE utility failures caused by failure of the TurboIMAGE intrinsic specified in the message.  A detail error message can be retrieved by calling DBEXPLAIN.	Take appropriate action depending on the detail message. If it looks like: <b>FILE SYSTEM ERROR #nnn</b> , look up the error in <i>Error Messages and Recovery Manual</i> .
402	DBINFO FAILURE AT xxxxx	Same as above.	For more information see the TurboIMAGE Library Procedure Error Messages earlier in this appendix, or see the <i>Error Messages and Recovery Manual</i> .  Same as above.
403	DBCLOSE FAILURE AT xxxxx	Same as above.	Same as above.
404	DBFIND FAILURE AT xxxxx	Same as above.	Same as above.
405	DBGET FAILURE AT xxxxx	Same as above.	Same as above.

**Table A-10. TurboIMAGE Extended Utility Program Unconditional Messages (Continued)**

#	MESSAGE	MEANING	ACTION
406	DBUPDATE FAILURE AT xxxxx	Error messages 406 through 414 are all TurboIMAGE utility failures caused by failure of the TurboIMAGE intrinsic specified in the message.  A detail error message can be retrieved by calling DBEXPLAIN.	For more information see the TurboIMAGE Library Procedure Error Messages earlier in this appendix, or refer to the <i>Error Messages and Recovery Manual</i> .
407	DBPUT FAILURE AT xxxxx	Same as above.	Same as above.
408	DBDELETE FAILURE AT xxxxx	Same as above.	Same as above.
409	DBLOCK FAILURE AT xxxxx	Same as above.	Same as above.
410	DBUNLOCK FAILURE AT xxxxx	Same as above.	Same as above.
411	DBCONTROL FAILURE AT xxxxx	Same as above.	Same as above.
412	DBBEGIN FAILURE AT xxxxx	Same as above.	Same as above.
413	DBEND FAILURE AT xxxxx	Same as above.	Same as above.
414	DBMEMO FAILURE AT xxxxx	Same as above.	Same as above.



# RESULTS OF MULTIPLE ACCESS

APPENDIX

B

When opening a data base with DBOPEN, TurboIMAGE returns information in the status array describing the results of the procedure call. Table B-1 can be used to interpret these results when multiple processes are using the data base.

Each box in Table B-1 is associated with a requested mode or TurboIMAGE utility routine identified at the far left of the row in which the box appears. It is also associated with a possible current access mode or utility routine identified at the top of the column in which the box appears. The contents of the boxes can be used to determine the results of a DBOPEN call.

If access is granted, condition code CCE is returned and the first word of the status array contains a zero. The boxes containing "G" represent this situation.

If access is not granted, and the reason relates to current data base activity, the results are like those shown in the other boxes. There are two types of situations:

- If the first two words of status contain -1 and 0 respectively, the third word of status will contain a single number. This number is the MPE failure code returned from the FCHECK intrinsic. See the *MPE Intrinsic Reference Manual* for MPE failure code meanings. If that number is 48, 90, or 91, the failure occurred because current access to the data base does not permit it to be opened in the requested mode. Find the boxes in the requested mode row which contain a number equal to the third status word. The possible modes and utility routines which other processes may be using are the ones which label the columns containing these boxes. For example, if the third status word contains 48 and the requested mode is 2, the possible current modes are 1 and 5.

To find an alternate mode for accomplishing the task, look down the columns containing these boxes for one containing a "G". If the requested mode labeling the row in which the "G" resides can be used, try opening the data base with that mode. In the example above, alternate modes would be 1 or 5 since these rows contain "G" in columns 1 and 5.

If the box with contents matching the third status word is in a column associated with a utility, usually the only choice is to wait until execution terminates. When DBSTORE is being run, it is possible to open the data base with mode 6 or 8.

- If the first word in the status array contains -32, the failure occurred because the root file could be opened but not with the necessary AOPTIONS. This value can also be returned in situations not related to multiple access. See Appendix A in this manual and the description of the AOPTIONS parameter of the FOPEN intrinsic in the *MPE Intrinsics Reference Manual*. Use the same technique described above to determine the possible current modes or other activity and to select a course of action. For example, if the requested mode is 2 and the first word of status equals -32, possible current modes are 4 and 8, and the DBSTORE utility may be executing.

Messages enclosed in quotes are printed when the situation represented by the row and column headings occurs.

Table B-1. Actions Resulting From Multiple Access of Data Bases

CURRENT DBOPEN MODE												
REQUESTED DBOPEN MODE	1	2	3	4	5	6	7	8	BEING DBSTOREd	BEING DBRESTORed	BEING DBUNLOADed OR DBLOADed	
	1	G	48	91	48	G	48	91	48	48	91	91
	2	48	G	91	-32	48	G	91	-32	-32	91	91
	3	90	90	91	90	90	90	91	90	90	91	91
	4	90	90	91	90	48	G	91	-32	-32	91	91
	5	G	48	91	48	G	48	91	48	48	91	91
	6	48	G	91	G	48	G	91	G	G	91	91
	7	90	90	91	90	90	90	91	90	90	91	91
	8	90	90	91	90	48	G	91	G	G	91	91
	:RUN DBSTORE	'DATA BASE IN USE'					G	'DATA BASE IN USE'	G	G	'DATA BASE IN USE'	
:RUN DBRESTOR	'DUPLICATE FILE NAME'											
:RUN DBUNLOAD OR :RUN DBLOAD	'DATA BASE IN USE'											

Note: 48, 90, and 91 are values returned in the third *status* word and -32 is a value returned in the first *status* word. G indicates access is granted.

# SUMMARY OF DESIGN CONSIDERATIONS

APPENDIX

C

1. Keep one-of-a-kind information in master data sets, such as, unique identifiers. Keep duplicate information in detail data sets, such as, records of events (sales, purchases, shipments).
2. Define a search item in a detail data set if you want to retrieve all entries with a common value for that data item.
3. Use manual master data sets to prevent entry of invalid data in the detail search item linked to the master through a path. Use automatic master data sets to save time if the detail search items are unpredictable or too numerous to enter manually.
4. Limit the use of sort items to paths with relatively short chains in order to reduce the time required to add and delete entries.
5. Select the path most frequently accessed in chained order as the primary path.
6. Remember that data items must be an integral number of words in length.
7. When selecting the maximum block size, consider the environment in which the data base will be used. (Refer to Section 3, \$CONTROL command for more information.)
8. If you intend to use QUERY with your data base, refer to the *QUERY Reference Manual* for the data types that QUERY supports.
9. In application programs either reference data items and data sets by name or use DBINFO at the beginning of the program to initialize the data item and data set numbers in order to maintain data independence of the programs.

Refer to the discussion in Section 4 to decide on appropriate access modes to use for your application programs.

10. Analyze the time required to maintain the data base, for example, the time required to unload and load the data base.
11. The capacity of each data set should be defined as realistically as possible since a capacity that is too large wastes disc space. The capacity can be increased when necessary by restructuring the data base as described in Section 7.
12. A master data set capacity equal to a prime number or to the product of two or three primes may yield fewer synonyms than a master data set capacity of many prime factors. See Table C-1 for a partial list of prime numbers.
13. The account and group in which the data base resides must have enough file space available to contain all the data base files.
14. If your application uses sorted paths, plan to add or delete entries (DPUT, DBDELETE) to sorted chains when the system is not very busy. If it is very busy, limit the data base activity on sorted chains to reading and updating (DBUPDATE).
15. Do all or most of your locking at one level (data base, data set, or data entry).

## Summary of Design Considerations

16. If locking at the data entry level, do all or most of the locking using the same item in each data set. Otherwise, performance will be the same as if you were locking at the data set level.
17. Avoid holding locks around a terminal read.

Table C-1. Selected Prime Numbers

101	280,001	680,003	1,800,017	5,800,019
503	290,011	690,037	1,900,009	5,900,047
1,009	300,007	700,001	2,000,003	6,000,011
5,003	310,019	710,009	2,100,001	6,100,001
10,007	320,009	720,007	2,200,013	6,200,003
15,013	330,017	730,003	2,300,003	6,300,011
20,011	340,007	740,011	2,400,001	6,400,013
25,013	350,003	750,019	2,500,009	6,500,003
30,011	360,007	760,007	2,600,011	6,600,001
35,023	370,003	770,027	2,700,023	6,700,007
40,009	380,041	780,029	2,800,001	6,800,033
45,007	390,001	790,003	2,900,017	6,900,001
50,021	400,009	800,011	3,000,017	7,000,003
55,001	410,009	810,013	3,100,011	7,100,003
60,013	420,001	820,037	3,200,003	7,200,007
65,003	430,007	830,003	3,300,001	7,300,001
70,001	440,009	840,023	3,400,043	7,400,011
75,011	450,001	850,009	3,500,017	7,500,013
80,021	460,013	860,009	3,600,001	7,600,013
85,009	470,021	870,007	3,700,001	7,700,071
90,001	480,013	880,001	3,800,021	7,800,017
95,003	490,001	890,003	3,900,067	7,900,001
100,003	500,009	900,001	4,000,037	8,000,009
110,017	510,007	910,003	4,100,011	8,100,073
121,001	520,019	920,011	4,200,013	8,200,007
130,003	530,017	930,011	4,300,003	8,300,009
140,009	540,041	940,001	4,400,021	
150,001	550,007	950,009	4,500,007	
160,001	560,017	960,017	4,600,003	
170,003	570,001	970,027	4,700,021	
180,001	580,001	980,077	4,800,007	
190,027	590,021	990,001	4,900,001	
200,003	600,011	1,000,003	5,000,011	
210,011	610,031	1,100,009	5,100,071	
220,009	620,003	1,200,007	5,200,007	
230,003	630,017	1,300,021	5,300,003	
240,007	640,007	1,400,017	5,400,001	
250,007	650,011	1,500,007	5,500,003	
260,003	660,001	1,600,033	5,600,027	
270,001	670,001	1,700,021	5,700,007	

# MULTIPLE RIN SPECIAL CAPABILITY

APPENDIX

D

For the purpose of deadlock prevention, the system views any call to DBLOCK in which something is actually locked as a lock on a single resource, even though the call may have specified multiple lock descriptors. Any program which does not have the Multiple RIN (Resource Identification Number) capability (CAP=MR) can only have one resource locked at a time, and thus can only call DBLOCK once without an intervening call to DBUNLOCK.

It may be necessary for some applications to violate this rule. The purpose of this appendix is to tell you how to avoid problems that may arise if you prepare your application programs with MR capability (CAP=MR).

Some typical situations in which CAP=MR may be required are the following:

- A program has two or more data bases open and wishes to lock part or all of each data base simultaneously. (One or more of the data bases may be on a remote HP 3000.)
- A program wishes to lock an MPE file and a data base simultaneously.
- A program wishes to lock data entries in a data base and, after reading their contents, to apply further locks. This is very dangerous and is not recommended, since deadlocks can occur very easily.

The danger in all cases is that a deadlock may occur. For example, suppose process A has data set 1 locked and is trying to lock data set 9, and process B has data set 9 locked and is waiting for data set 1. In this case, a deadlock has occurred and the only way to break it is to restart the operating system.

TurboIMAGE avoids deadlocks within single calls to DBLOCK by first sorting the lock descriptors into an internally-defined sequence. It then applies the locks in ascending order sorted by data set number, then by the value provided for the lock item. You can use the same strategy in avoiding deadlocks. First define an order in which entities should be locked and then impose a rule on all programmers that this order be adhered to. The sequence of unlocking is not important. The rule that you establish should apply to all lockable entities:

- Data bases, data sets, and data entries.
- Remote data bases, data sets, and data entries.
- MPE files (FLOCK), global RINs (LOCKGLORIN), KSAM files (FLOCK), and files locked with the COBOLLOCK procedure.

When applying multiple DBLOCK calls to the same data base, extreme caution should be exercised since the deadlock situations can be very subtle. For example, if a process locks a data set and then attempts to lock the data base, the process will wait for itself forever.

If it is absolutely necessary to make multiple DBLOCK calls, the following information about how TurboIMAGE performs locking may be useful.

## **SORT SEQUENCE FOR LOCK DESCRIPTORS**

TurboIMAGE internally sorts the lock descriptors in the order as follows:

- ascending data set number
- lower bound of data item value for each data set number

If a lock descriptor's relop field contains  $\leq$ , it collates before any other lock descriptors for the data set since it has the lowest possible lower bound for its value. For example, a lock of descriptor of SALES:QUANTITY  $\leq$  10 collates before a lock descriptor of SALES:QUANTITY = 5, since the lower bound of the former is the lowest possible integer for an I-type data item.

## **CONDITIONAL LOCKS**

During a DBLOCK, if TurboIMAGE discovers a lock descriptor that is identical to one previously put into effect by the same user through the same access path, it ignores the latest lock descriptor. For example, the lock descriptor SALES:ACCOUNT = 89393899 is ignored if SALES:ACCOUNT = 89393899 was locked earlier on the same access path. However, it will not be ignored if a lock descriptor such as SALES:@ has been specified earlier.

If multiple lock descriptors are specified with mode 6 (conditional data entry locking), TurboIMAGE indicates how many locks have been applied when it returns (status word 1) to the calling process. It does not release the successful locks even though all the requested locks have not been applied. Since TurboIMAGE ignores identical lock descriptors specified a second time, it is possible to call DBLOCK again with the same descriptor list (if the program has MR capability). Those lock descriptors that are already in effect will be ignored and the others will be tried again. The second word of the status array contains the number of descriptors successfully locked in each call. This technique will not cause deadlocks provided the lock descriptor list is not altered.

It is not recommended that this technique be used in a tight program loop since system performance will degrade markedly. However, it can be used to retry the locks in a situation where the program prompts the user to determine whether the locks should be tried again, and the user indicates that they should. (If the user does not want to continue trying to lock all the entities, be sure to unlock the ones that succeeded.) DBUNLOCK need only be called once to unlock everything you have locked.

## REMOTE DATA BASES

Locking remote data base entities is the same as locking data base entities with the following exception. If the local system has a user-created process structure, and each process is locking a remote data base independently, the programs must have Multiple RIN capability since they are in the same job/session. The only effect using MR capability has on the local system is that the rule prohibiting multiple DBLOCK calls is not enforced. However, to access remote data bases each local process must issue a separate REMOTE HELLO to ensure that it has a corresponding process in the remote system.

The system does not force you to establish corresponding remote processes, but failure to do so can result in the remote session permanently suspending and requires a remote system restart to recover.

### WARNING

**Hewlett-Packard does not accept responsibility for system lockouts and deadlocks when Multiple RIN capability is in use with TurboIMAGE/3000.**

The DBUTIL command >>SHOW data base name LOCKS may be useful in tracing deadlocks that occur when CAP=MR is used.

# TurboIMAGE LOG RECORD FORMATS

APPENDIX

E

NOTE: All TurboIMAGE records are contained within MPE "WRITELOG" records. Consequently, all information contained in the header portion of each WRITELOG record is available, in addition to the information provided by TurboIMAGE.

## DBBEGIN

WORD(0-8)	-	MPE WRITELOG RECORD
WORD(9)	-	LOG RECORD LENGTH
WORD(10)	-	DBBEGIN LOG RECORD CODE ("BE")
WORD(11)	-	DATA SEGMENT NUMBER
WORD(12)	-	RECOVERY FLAG ("NO"-FAILED,"OK"-RECOVERED)*
WORD(13)	-	TRANSACTION NUMBER (2 WORDS)
WORD(15)	-	LENGTH OF USER BUFFER
WORD(16)	-	START OF USER BUFFER

## DBOPEN

WORD(0-8)	-	MPE WRITELOG RECORD
WORD(9)	-	LOG RECORD LENGTH
WORD(10)	-	DBOPEN LOG RECORD CODE ("OP")
WORD(11)	-	BASE ID
WORD(12)	-	USER NAME
WORD(16)	-	USER GROUP
WORD(20)	-	USER ACCOUNT
WORD(24)	-	USER IDENTIFIER
WORD(28)	-	DATA BASE NAME
WORD(31)	-	DATA BASE GROUP
WORD(35)	-	DATA BASE ACCOUNT
WORD(39)	-	SECURITY CLASS
WORD(40)	-	DBOPEN MODE PARAMETER
WORD(41)	-	LOGGING IDENTIFIER
WORD(45)	-	DBSTORE TIME STAMP (3 WORDS)
WORD(48)	-	USER PROGRAM NAME
WORD(52)	-	USER PROGRAM GROUP
WORD(56)	-	USER PROGRAM ACCOUNT
WORD(60)	-	MODE FROM WHO INTRINSIC
WORD(61)	-	CAPABILITY FROM WHO INTRINSIC
WORD(63)	-	LOCAL ATTRIBUTE FROM WHO INTRINSIC
WORD(65)	-	LOGICAL DEVICE OF JOB/SESSION INPUT
WORD(66)	-	PREVIOUS ROLLBACK TIME STAMP (3 WORDS)
WORD(69)	-	CURRENT ROLLBACK TIME STAMP (3 WORDS)
WORD(72)	-	RESERVED FOR DBRECOV RUN TIME USE
WORD(73)	-	RESERVED FOR DBRECOV RUN TIME USE



**DBPUT**

WORD(0-8) - MPE WRITELOG RECORD  
WORD(9) - LOG RECORD LENGTH  
WORD(10) - DBPUT LOG RECORD CODE ("PU")  
WORD(11) - DATA SEGMENT NUMBER  
WORD(12) - RECOVERY FLAG ("NO"-FAILED,"OK"-RECOVERED)\*  
WORD(13) - TRANSACTION NUMBER (2 WORDS)  
WORD(15) - DATA SET NUMBER  
WORD(16) - DATA SET TYPE ("MA"-MASTER,"DE"-DETAIL)  
WORD(17) - RECORD NUMBER (2 WORDS)  
WORD(19) - MODE PARAMETER  
WORD(20) - OFFSET TO KEY ITEM VALUE (IF MASTER TYPE)  
WORD(21) - OFFSET TO ITEM LIST  
WORD(22) - OFFSET TO DATA  
WORD(23) - BEGIN OF KEY,ITEM LIST,AND DATA BUFFER

**DBUPDATE**

WORD(0-8) - MPE WRITELOG RECORD  
WORD(9) - LOG RECORD LENGTH  
WORD(10) - DBUPDATE LOG RECORD CODE ("UP")  
WORD(11) - DATA SEGMENT NUMBER  
WORD(12) - RECOVERY FLAG ("NO"-FAILED,"OK"-RECOVERED)\*  
WORD(13) - TRANSACTION NUMBER (2 WORDS)  
WORD(15) - DATA SET NUMBER  
WORD(16) - DATA SET TYPE ("MA"-MASTER,"DE"-DETAIL)  
WORD(17) - RECORD NUMBER (2 WORDS)  
WORD(19) - MODE PARAMETER  
WORD(20) - OFFSET TO KEY ITEM VALUE (IF MASTER TYPE)  
WORD(21) - OFFSET TO ITEM LIST  
WORD(22) - OFFSET TO NEW DATA  
WORD(23) - OFFSET TO OLD DATA  
WORD(24) - BEGIN OF KEY,ITEM LIST,AND DATA BUFFER

**DBDELETE**

WORD(0-8) - MPE WRITELOG RECORD  
WORD(9) - LOG RECORD LENGTH  
WORD(10) - DBDELETE LOG RECORD CODE ("DE")  
WORD(11) - DATA SEGMENT NUMBER  
WORD(12) - RECOVERY FLAG ("NO"-FAILED,"OK"-RECOVERED)\*  
WORD(13) - TRANSACTION NUMBER (2 WORDS)  
WORD(15) - DATA SET NUMBER  
WORD(16) - DATA SET TYPE ("MA"-MASTER,"DE"-DETAIL)  
WORD(17) - RECORD NUMBER (2 WORDS)  
WORD(19) - MODE PARAMETER  
WORD(20) - OFFSET TO KEY ITEM VALUE (IF MASTER TYPE)  
WORD(21) - OFFSET TO DELETED DATA  
WORD(22) - START OF KEY AND DATA BUFFER

**DBMEMO**

WORD(0-8) - MPE WRITELOG RECORD  
 WORD(9) - LOG RECORD LENGTH  
 WORD(10) - DBMEMO LOG RECORD CODE ("ME")  
 WORD(11) - DATA SEGMENT NUMBER  
 WORD(12) - RECOVERY FLAG ("NO"-FAILED,"OK"-RECOVERED)\*  
 WORD(13) - TRANSACTION NUMBER (2 WORDS)  
 WORD(15) - LENGTH OF USER BUFFER  
 WORD(16) - START OF USER BUFFER

**DBCLOSE**

WORD(0-8) - MPE WRITELOG RECORD  
 WORD(9) - LOG RECORD LENGTH  
 WORD(10) - DBCLOSE LOG RECORD CODE ("CL")  
 WORD(11) - BASE ID  
 WORD(12) - USER PROCESS ABORT INDICATOR  
 WORD(13) - RESERVED FOR DBRECOV RUN TIME USE  
 WORD(14) - RESERVED FOR DBRECOV RUN TIME USE

**DBEND**

WORD(0-8) - MPE WRITELOG RECORD  
 WORD(9) - LOG RECORD LENGTH  
 WORD(10) - DBEND LOG RECORD CODE ("EN"), OR  
                   ("AE") IF ABORTED  
 WORD(11) - DATA SEGMENT NUMBER  
 WORD(12) - RECOVERY FLAG ("NO"-FAILED,"OK"-RECOVERED)\*  
 WORD(13) - TRANSACTION NUMBER (2 WORDS)  
 WORD(15) - LENGTH OF USER BUFFER  
 WORD(16) - START OF USER BUFFER

\* The recovery flag will always be zero in the log file records. It is used during recovery if user recovery files are created.

# MPE LOG RECORD FORMATS

APPENDIX

F

MPE Log Record Formats for log files and user recovery files.

## HEADER

WORD(0)	-	RECORD NUMBER (2 WORDS)
WORD(2)	-	CHECKSUM
WORD(3)	-	SUBSYSTEM IDENTIFIER (1ST BYTE)
WORD(3)	-	LOG RECORD CODE (2ND BYTE - 4)
WORD(4)	-	TIME
WORD(6)	-	DATE
WORD(7)	-	LOGGING IDENTIFIER

## OPENLOG

WORD(0)	-	RECORD NUMBER (2 WORDS)
WORD(2)	-	CHECKSUM
WORD(3)	-	SUBSYSTEM IDENTIFIER (1ST BYTE)
WORD(3)	-	LOG RECORD CODE (2ND BYTE - 1)
WORD(4)	-	TIME
WORD(6)	-	DATE
WORD(7)	-	LOGGING IDENTIFIER
WORD(11)	-	LOG NUMBER
WORD(12)	-	USER NAME, GROUP, ACCOUNT
WORD(24)	-	PIN#

## WRITELOG

WORD(0)	-	RECORD NUMBER (2 WORDS)
WORD(2)	-	CHECKSUM
WORD(3)	-	SUBSYSTEM IDENTIFIER (1ST BYTE)
WORD(3)	-	LOG RECORD CODE (2ND BYTE - 2)
WORD(4)	-	TIME
WORD(6)	-	DATE
WORD(7)	-	LOG NUMBER
WORD(8)	-	USER BUFFER LENGTH
WORD(9)	-	USER BUFFER AREA

# **WRITELOG CONTINUATION**

WORD(0)	-	RECORD NUMBER (2 WORDS)
WORD(2)	-	CHECKSUM
WORD(3)	-	SUBSYSTEM IDENTIFIER (1ST BYTE)
WORD(3)	-	LOG RECORD CODE (2ND BYTE - 7)
WORD(4)	-	TIME
WORD(6)	-	DATE
WORD(7)	-	LOG NUMBER
WORD(8)	-	USER BUFFER LENGTH
WORD(9)	-	USER BUFFER AREA

# **CLOSELOG**

WORD(0)	-	RECORD NUMBER (2 WORDS)
WORD(2)	-	CHECKSUM
WORD(3)	-	SUBSYSTEM IDENTIFIER (1ST BYTE)
WORD(3)	-	LOG RECORD CODE (2ND BYTE - 3)
WORD(4)	-	TIME
WORD(6)	-	DATE
WORD(7)	-	LOGGING IDENTIFIER
WORD(11)	-	LOG NUMBER
WORD(12)	-	USER NAME, GROUP, ACCOUNT
WORD(24)	-	PIN#

# **CHANGELOG**

WORD(0)	-	RECORD NUMBER (2 WORDS)
WORD(2)	-	CHECKSUM
WORD(3)	-	12 RECORD CONTAINS THE previous FILE IN THE SET
WORD(3)	-	13 RECORD CONTAINS THE next FILE IN THE SET
WORD(4)	-	TIME
WORD(6)	-	DATE
WORD(7)	-	LOGID
WORD(11)	-	SEQUENCE NUMBER OF THE CURRENT FILE
WORD(12)	-	CREATION TIME OF THE FIRST FILE
WORD(14)	-	CREATION DATE OF THE FIRST FILE
WORD(15)	-	NAME OF THE FIRST FILE IN THE SET
WORD(33)	-	LOG TYPE OF THE FIRST FILE IN THE SET
WORD(34)	-	NAME OF THE next FILE IN THE SET
WORD(34)	-	NAME OF THE previous FILE IN THE SET
WORD(52)	-	LOG TYPE OF THE next FILE IN THE SET
WORD(52)	-	LOG TYPE OF THE previous FILE IN THE SET
WORD(53)	-	NAME OF THE CURRENT FILE IN THE SET
WORD(71)	-	LOG TYPE OF THE CURRENT FILE IN THE SET

**RESTART**

WORD(0) - RECORD NUMBER (2 WORDS)  
WORD(2) - CHECKSUM  
WORD(3) - SUBSYSTEM IDENTIFIER (1ST BYTE)  
WORD(3) - LOG RECORD CODE (2ND BYTE - 6)  
WORD(4) - TIME  
WORD(6) - DATE  
WORD(7) - LOGGING IDENTIFIER

**CRASH**

WORD(0) - RECORD NUMBER (2 WORDS)  
WORD(2) - CHECKSUM  
WORD(3) - SUBSYSTEM IDENTIFIER (1ST BYTE)  
WORD(3) - LOG RECORD CODE (2ND BYTE - 9)  
WORD(4) - TIME  
WORD(6) - DATE

**NULL**

WORD(0) - RECORD NUMBER (2 WORDS)  
WORD(2) - CHECKSUM  
WORD(3) - RECORD CODE ("" (BLANK))

**TRAILER**

WORD(0) - RECORD NUMBER (2 WORDS)  
WORD(2) - CHECKSUM  
WORD(3) - SUBSYSTEM IDENTIFIER (1ST BYTE)  
WORD(3) - LOG RECORD CODE (2ND BYTE - 5)  
WORD(4) - TIME  
WORD(6) - DATE  
WORD(7) - LOGGING IDENTIFIER

## RECOVERY QUICK REFERENCE

The following two pages offer a very brief outline of the recovery options in TurboIMAGE. For more information regarding Intrinsic Level Recovery (ILR), Roll-Forward Recovery and Roll-Back Recovery refer to Section 7 "Maintaining the Data Base" and Section 8 "Using the Data Base Utilities". The data base administrator should determine which type of recovery to use based on the size of the data base, frequency of system failures, equipment available, and other considerations.

### Intrinsic Level Recovery

- is automatic and transparent to the user.
- returns the data base physical consistency by completing the last interrupted DBDELETE or DBPUT.
- when enabled, automatically logs each DBPUT and DBDELETE call to an internal ILR file (only the most recent call is noted). Data base is recovered automatically if recovery is needed next time the data base is opened.
- when enabled, the ILR file is stored and restored automatically along with the data base, using DBSTORE and DBRESTOR utilities.
- when disabled by a user, TurboIMAGE checks the ILR file to see if the data base needs recovery. If ILR is pending or required, TurboIMAGE uses the existing ILR file to recover the data base. The ILR file is purged and the flag in the data base root file is cleared.
- can't use output deferred mode (AUTODEFER). This ensures the structural integrity of the data base.
- can't defer writing modifications to the data base since output deferred can not be used.
- is intrinsic driven, transaction locking is not necessary.
- incurs a slight overhead on DBPUT's and DBDELETE's due to logging intrinsics to the ILR log file.
- after a system failure requires no more overhead than a single DBPUT or DBDELETE.
- can be used with user logging.

## Roll-Back Recovery

- provides rapid recovery of data base integrity following a soft system crash and restores the data base to a consistent state, physically and logically by backing out any incomplete transactions.
- requires logging, however only the current log file is required to restore data base integrity. DBSTORE is recommended but not required. (Logging is automatically enabled.)
- automatically enables ILR. ILR is required to ensure correctness of data base physical data links, to prevent the possibility of broken chains.
- is initiated by using DBRECOV, >ROLLBACK command.
- when disabled, both ILR and logging must be manually disabled using DBUTIL.
- requires all multiple-intrinsic data base transactions execute independently, using logical transaction locking.
- uses the time stamp during recovery to verify the correct log file for each data base being recovered. The time stamp is updated when the data base is first opened and is logged to the log file and the root file.
- should not be disabled if roll-back recovery must be used later because it will reset the logging time stamp, therefore recovery cannot be performed. The data base will be considered correct and cannot be rolled-back.
- transactions are not lost during a WARMSTART since they are not held in the memory buffer. (A WARMSTART must be performed after a system crash.)
- process finishes then the RESTART option in user logging can be used or a new logging cycle can be started. Remember to purge the log file before starting the new logging cycle.

## Roll-Forward Recovery

- provides recovery of a data base, both structurally and logically, to a likeness of its state at the time of a hard system failure.
- uses DBSTORE and DBRESTOR to restore the data base to a consistent state.
- requires logging be enabled. During recovery all log files since the last data base backup copy must be applied.
- and TurboIMAGE logging depend upon exact correspondence between the stored backup data base copy and the working data base on disc at the time logging was interrupted. The DBSTORE flag and log file time stamp will enforce this condition.
- does not require ILR be enabled, however, enabling ILR is recommended to eliminate the possibility of broken chains and to provide a log file for audit purposes. ILR must be manually enabled and disabled.
- and logging provide recovery of both intrinsics and transactions following a system failure.
- is initiated using DBRECOV, >RECOVER command. The data base must be purged and restored before recovery is initiated.
- during a WARMSTART all transactions in the memory buffer will be lost.
- does not require logical transaction locking, however it is recommended.
- process finishes the RESTART option in user logging can be used or a new logging cycle can be started. Remember to purge the log file before starting the new logging cycle.
- may not recover all transactions. If using MPE V/P or DBEND mode 2 transactions will be flushed to the log file at DBEND time, therefore transactions will not be lost.
- the data base administrator should consider how often to store the data base. The more frequent the DBSTORE the smaller the log files will be, thus cutting the time required for recovery.

## Recovery

- without a data base backup copy can be performed using DBUNLOAD (chained) and DBLOAD to salvage all or most of a data base.



## LOGGING DEVICE QUICK REFERENCE

The lists below outline both the benefits and disadvantages of logging to disc and logging to tape. Refer to Section 7 "Maintaining the Data Base" for more information on user logging. The data base administrator must determine which type of logging device to use based on equipment, operations staff, number of users and size of the data base, and other considerations listed on this page.

### LOGGING TO TAPE

- does not take up disc space.
- requires a dedicated tape drive.
- requires a reliable tape drive and a library of "good" tapes.
- is more secure in terms of a hard crash.
- is more time consuming. After a system failure the tape must be rewound and sequentially scanned until the end of file is detected. Remaining records are then appended to the file.
- the console operator must be available to respond to requests for tape mounts. If a request is ignored and you run out of memory buffer space, logging will stop. Applications requiring logging will get a WRITELOG error and will terminate.
- does not provide any security measures to prevent overwriting the current tape. The console operator should use care to mount a new tape before placing on-line.
- has overhead similar to logging to disc.

### LOGGING TO DISC

- files are susceptible to a hard crash.
- the integrity of the log file may be no better than the current data base state. The log file may contain inconsistencies, bad characters or other invalid data.
- if not using the AUTO option or :CHANGELOG command the data base administrator must make sure disc file space of the current log file is sufficient so that end of file is not reached. If end of file is reached logging will stop. Applications requiring logging will get a WRITELOG error and will terminate.
- during a WARMSTART the console operator can respond with an option to override or cancel the clean-up procedure on log files. Fewer log records written just prior to system failure are lost.
- has overhead comparable to logging to tape.

## SAMPLE JOB STREAMS

The following pages list sample job streams which can be used to initiate a logging cycle and to recover a data base using either roll-forward recovery or roll-back recovery. The data base administrator must decide which recovery method to use and how often the data base should be stored.

```

:JOB MGR.DATAMGT
:GETLOG ORDERLOG;LOG=ORDER001,DISC      <<Acquire log identifier>>

:BUILD ORDER001;DISC=200000,20,7        <<Build new log file>>

:RUN DBUTIL.PUB.SYS                      <<Set the data base flags>>
SET ORDERS LOGID=ORDERLOG                <<in the root file>>
ENABLE ORDERS FOR LOGGING                <<*>>
DISABLE ORDERS FOR ACCESS
ENABLE ORDERS FOR RECOVERY
EXIT
:RUN DBSTORE.PUB.SYS                     <<Store the data>>
ORDERS                                   <<base>>

:LOG ORDERLOG,START                      <<Start the logging process,>>
                                         <<logid is ORDERLOG>>

:RUN DBUTIL.PUB.SYS                      <<Set the data base>>
ENABLE ORDERS FOR ACCESS                 <<flags in the root file>>
DISABLE ORDERS FOR RECOVERY
EXIT
:EOJ

```

**Figure G-1. Sample Job Stream for Starting Logging Cycle**

The job stream above builds a new log file, in this case the log file resides on disc, and sets the data base flags. A backup copy of the data base is made (this sets the logid in the root file), logging is initiated with "START" and the data base is enabled for access and recovery is disabled.

\* Roll-forward recovery is being used in the above example. If using roll-back recovery, replace the line with >>ENABLE ORDERS FOR ROLLBACK. This enables the data base for logging, ILR and ROLLBACK recovery.

At the end of the logging cycle the data base administrator stops logging, stores the current log file on tape for back-up, purges the current log file, builds a new log file and stores a data base backup copy. To end the logging cycle, the steps in Figure G-1 are completed after initiating the following:

```

:LOG ORDERLOG,STOP      <<Stop logging.>>
                        <<:STORE the log file to tape.>>
:PURGE ORDER001         <<Purge the current log file.>>

```

:JOB MGR.DATAMGT	
:RUN DBUTIL.PUB.SYS	<<If the data base is to be>>
DISABLE ORDERS FOR ACCESS	<<stored prior to recovery,>>
ENABLE ORDERS FOR RECOVERY	<<set flags in the data>>
EXIT	<<base and run DBSTORE*>>
:RUN DBSTORE.PUB.SYS	
ORDERS	
:RUN DBUTIL.PUB.SYS	<<Purge the current data>>
PURGE ORDERS	<<base, and restore the>>
EXIT	<<backup copy of the>>
:RUN DBRESTOR.PUB.SYS	<<data base>>
ORDERS	
:RUN DBUTIL.PUB.SYS	<<Set the flags in the>>
DISABLE ORDERS FOR ACCESS	<<data base*>>
ENABLE ORDERS FOR RECOVERY	
EXIT	
:RUN DBRECOV.PUB.SYS	<<Use roll-forward>>
RECOVER ORDERS	<<recovery on data>>
FILE PART1,SYS/P1D1.MKTG,0,3	<<base ORDERS. File>>
FILE PART2,SYS/P1D1.MKTG,0,3	<<command is used to>>
FILE PART3,SYS/P1D1.MKTG,0,3	<<route log records>>
RUN	<<to individual log>>
EXIT	<<user files>>
:LOG ORDERLOG,RESTART	<<Restart current>>
:RUN DBUTIL.PUB.SYS	<<log file and set>>
ENABLE ORDERS FOR ACCESS	<<the data base>>
DISABLE ORDERS FOR RECOVERY	<<flags>>
EXIT	
:EOJ	

Figure G-2. Sample Job Stream for Roll-Forward Recovery

Storing the data base prior to purging it and restoring the data base backup copy is optional. After recovery has completed logging can either be restarted (from the current log file) or the log file can be purged and a new log file built.

\* If all recommended procedures have been followed, the data base backup copy will have flags set for enabling recovery and disabling access, therefore this step would be unnecessary. If this process is being done interactively a >>SHOW data base name FLAGS in DBUTIL will show if the flags for recovery and access are correctly set

:JOB MGR.DATAMGT	
:RUN DBUTIL.PUB.SYS	<<Set the flags in the data>>
DISABLE ORDERS FOR ACCESS	<<base root file*>>
ENABLE ORDERS FOR RECOVERY	
EXIT	
<<A DBSTORE of the data base at this time is recommended.	
:RUN DBRECOV.PUB.SYS	<<Use roll-back recovery>>
ROLLBACK ORDERS	<<on data base ORDERS. File>>
FILE PART1,SYS/P1D1.MKTG,0,3	<<command is used to route>>
FILE PART2,SYS/P1D1.MKTG,0,3	<<log records to individual>>
FILE PART3,SYS/P1D1.MKTG,0,3	<<user files>>
RUN	
EXIT	
:LOG ORDERLOG,RESTART	<<Restart the current log>>
:RUN DBUTIL.PUB.SYS	<<file and set the data>>
ENABLE ORDERS FOR ACCESS	<<base flags>>
DISABLE ORDERS FOR RECOVERY	
EXIT	
:EOJ	

**Figure G-3. Sample Job Stream for Roll-Back Recovery**

After recovery has completed logging can either be restarted (from the current log file) or the log file can be purged and a new log file built.

\* If all recommended procedures have been followed the data base will have flags set for enabling recovery and disabling access, therefore this step would be unnecessary. If A DBSTORE of the data base can be done at this time, this is optional however. If this process is being done interactively a >>SHOW data base name FLAGS in DBUTIL will show if the flags for recovery and access are correctly set.

## PRECONVERSION CONSIDERATIONS

After TurboIMAGE is installed, existing IMAGE/3000 data bases must be converted before they can be accessed. The DBCONV utility performs a conversion which involves restructuring the root file and all master sets. Detail data sets remain unchanged. It is recommended that you DBSTORE data bases and consider disc space requirements prior to converting the data bases with DBCONV.

There is a temporary and potential permanent increase in disc space requirements when converting from IMAGE/3000 to TurboIMAGE data bases. The extra space is needed to accommodate increasing the chain limit capabilities. DBCONV adds one word for each path in a master data set media record. The temporary extra disc space is equal to the largest master set in the data base. This space is returned when the conversion process completes.

Since the smallest amount of memory TurboIMAGE can allocate is a sector, the permanent extra disc space requirements could increase by as much as a sector per block. When DBCONV encounters such cases, the additional sectors are added automatically, and, in effect, the physical block size is increased.

The blocking effectiveness of each master data set is the major factor in determining this potential increase. In some cases no extra disc space is required to accommodate the added word per path. If you are currently nearing capacity on your discs, it is recommended that you evaluate the potential increase in disc space for any data bases to be converted to TurboIMAGE.

In addition to any permanent disc space requirements, some temporary disc space is needed for the conversion. During this process the master sets are converted, one at a time, into a temporary holding space of equal size. Therefore, the largest space needed is equal to the largest master set (plus any potential increase as determined above).

Before converting the data bases, you can use the DBCONV option, VERIFY, to analyze the following:

- Total extra disc space required by the converted database.
- Temporary disc space required by DBCONV for conversion of the database.
- The database root file for inconsistencies in data set definitions.

## TurboIMAGE Conversion (DBCONV)

An example of DBCONV with the VERIFY option is shown below.

```
:RUN DBCONV.PUB.SYS,VERIFY  
  
Database name? INVNTY  
  
Starting conversion disc space analysis and verification.  
  
For database INVNTY.  
  
Total extra disc space required for converted database: 1 sectors  
  
The temporary disc space required: 13 sectors  
  
Inconsistency is not detected in the root file.  
  
Conversion disc space analysis and verification is done !!!!  
  
END OF PROGRAM
```

If an inconsistency is detected, refer to the error messages at the end of this section.

It should be noted that a potential problem of less effective blocking only affects converted IMAGE/3000 data bases. New data bases created in a TurboIMAGE environment will be blocked automatically by DBSCHEMA, with the increased chain count taken into consideration. As a work-around to a potential blocking problem, you may perform a DBUNLOAD of the converted data base, run DBSCHEMA on the schema (which will block more effectively), and then perform a DBLOAD of the data.

### **CAUTION**

As a result of the potential increase in disc space requirements, if you are nearing capacity on your discs, conversion and migration to TurboIMAGE may not be advisable without the addition of disc space.

## CONVERTING FROM IMAGE/3000 TO TURBOIMAGE

TurboIMAGE enhancements provide the capability for growth in the data base by changing the limitations and adding new tables in the root file. Because TurboIMAGE data base internal structure is different from IMAGE/3000, the databases must be converted with DBCONV before they can be accessed. On the following pages are examples and information on running the DBCONV utility. Tables H-1 and H-2 may be referenced for any errors which occur during conversion.

A creator or user with AM or SM capability can run DBCONV to convert the data base. DBCONV will not run if ILR is enabled on the data base. A DBUTIL >>SHOW *data base name* FLAGS can be used to determine what is enabled or disabled for the data base. If ILR is enabled, a DBUTIL >>DISABLE should be performed to disable ILR before TurboIMAGE is installed and the conversion program is run. It is recommended that the data base administrator back up (DBSTORE or MPE STORE) the data base and schema file prior to the installation of TurboIMAGE.

An alternative method is available to disable ILR or convert each data base individually. Users can utilize an Editor use file which creates a master job stream that can be used to convert all the data bases on the system. Refer to "Converting Using JOB STREAMS" later in this appendix for more information.

If TurboIMAGE is installed and ILR has not been disabled, the user can run DBUTILB. DBUTILB works the same way DBUTIL does, however, it has been modified to disable flags in an IMAGE/3000 data base once TurboIMAGE has been installed. Example 2 on the following page shows how to use DBUTILB.

<b>CAUTION</b>
----------------

If it is necessary to return to IMAGE/3000 after data bases have been converted to TurboIMAGE, the DBCONV option, BACKWARD, is available. This backward conversion process will not be successful if the new TurboIMAGE data base limits have been applied to the data base. DBCONV, BACKWARD should be run while TurboIMAGE is installed on the system.

To convert an individual data base either interactively or in batch mode, use the following program:

## Syntax

```
:RUN DBCONV.PUB.SYS
```

```
DATABASE NAME? data base name [/maintenance word] [.group]
```

## Parameters

*data base name* is the name of the data base to be converted.

*maintenance word* is the maintenance word for the data base being converted. If the user is not the data base creator, then the maintenance word must be specified.

*group* is the group where the data base resides. This is an optional parameter.

## Example 1

```
:RUN DBCONV.PUB.SYS
```

```
HP32215C.00.00 TurboIMAGE:DBCONV (C) COPYRIGHT HEWLETT-PACKARD CO. 1984
```

```
DATABASE NAME? ORDERS
```

```
Database has not been modified since last DBSTORE.
```

```
Continue (YES/NO)? YES
```

```
ORDERS01 has been converted.
```

```
ORDERS02 has been converted.
```

```
ORDERS03 has been converted.
```

```
ORDERS05 has been converted.
```

```
ORDERS has been converted.
```

```
Detail sets don't need to be converted.
```

```
Conversion is done !!!!
```

```
END OF PROGRAM
```



## Example 2

```

:RUN DBCONV.PUB.SYS

HP32215C.00.00 TurboIMAGE: DBCONV (C) COPYRIGHT HEWLETT-PACKARD CO. 1984

DATABASE NAME? RETAIL

ILR is on, disable it first

END OF PROGRAM

:RUN DBUTILB.PUB.SYS

>>DISABLE RETAIL FOR ILR
  ILR is Disabled
>>EXIT

END OF PROGRAM

:RUN DBCONV.PUB.SYS

DATABASE NAME? RETAIL

Database has not been modified since last DBSTORE
Continue (YES/NO) ? YES

RETAIL01 has been converted.
RETAIL  has been converted.

Detail sets don't need to be converted.

Conversion is done !!!!

END OF PROGRAM

```

## Discussion

The first example shows conversion of the data base ORDERS. Since the user running DBCONV was the data base creator, no maintenance word was entered. The user is prompted for the data base name. DBCONV will terminate and print the message DATA BASE IN USE if it cannot gain exclusive access to the specified data base. The conversion program will check the data base "dirty" flag in the root file to determine if the data base has been modified since the last DBSTORE was made.

<b>NOTE</b>
-------------

It is highly recommended that the data base administrator perform a DBSTORE of data bases and MPE STORE of schema text files prior to updating the system to TurboIMAGE. Backup copies of data bases and schema files may save considerable time if any problems occur during conversion. DBCONV checks the root file flag to determine the state of the data base. If the data base is in an inconsistent state, DBCONV will terminate and the data base will not be converted.

If the data base has not recently been stored, and the user determines it is unnecessary, then DBCONV can be continued with the response "YES" and conversion will complete. The following message will appear after the data base name is entered:

Database has been modified since last DBSTORE.

The user will be prompted to continue whether or not the data base has been recently DBSTORE'd. The reason for this is to provide consistency so that conversion may be run in job (batch) mode. Running the conversion utility in job mode will facilitate data base conversion at night, or when best suited for system operation.

DBCONV will list the data set files converted, along with the data base root file. Notice that the detail data set files need no conversion, ORDERS04 and ORDERS06 were not converted because they are detail data set files.

The second example shows the conversion of the data base RETAIL. ILR was not disabled prior to running DBCONV, an error message is printed and DBCONV is terminated. Since the RETAIL data base is still an IMAGE/3000 data base, the DBUTILB program is run and ILR is disabled on the data base. This utility program accepts an IMAGE data base and allows the disabling of ILR when TurboIMAGE has been installed on the system. ILR is disabled and DBCONV is run again and completes successfully. Note that DBUTILB can be used to change the flags and to erase or purge a data base prior to running the conversion program after TurboIMAGE has been installed.

The following is a sample job stream for DBCONV using data bases ORDERS and RETAIL:

### Example (Job Mode)

:JOB MGR.ACCOUNTA,ADMIN	
:RUN DBCONV.PUB.SYS	<<Maintenance word and group are>>
ORDERS/SELL.DATAMGT	<<entered. ORDERS resides in a different>>
YES	<<group than the log on group.>>
:RUN DBCONV.PUB.SYS	
RETAIL	<<Creator's log on is used, so no>>
YES	<<maintenance word is required.>>
:EOJ	

## Converting from TurboIMAGE to IMAGE/3000

In case a problem occurs after the data base is converted to TurboIMAGE, the following utility may be used to return the data base to IMAGE/3000. The conversion utility is run similar to DBCONV, however, the internal structure of the data base will be converted from TurboIMAGE to IMAGE/3000. DBCONV, BACKWARD must be run on all data bases on the system. Then the system must be updated to reinstall IMAGE/3000.

### CAUTION

This backward conversion process may not be successful if the new TurboIMAGE data base limits have been used in the data base. Backward conversion should be done while TurboIMAGE is installed on the system.

## Syntax

```
:RUN DBCONV.PUB.SYS, BACKWARD
```

```
DATABASE NAME? data base name [/maintenance word] [.group]
```

## Parameters

BACKWARD	is the option which indicates the conversion process to convert a TurboIMAGE data base back to an IMAGE data base.
<i>data base name</i>	is the name of the data base to be converted.
<i>maintenance word</i>	is the maintenance word for the data base being converted. If the user is not the data base creator, then the maintenance word must be specified.
<i>group</i>	is the group where the data base resides. This is an optional parameter.

## Example

```
:RUN DBCONV.PUB.SYS,BACKWARD
```

```
HP32215C.00.00 TurboIMAGE/3000: DBCONV (C) COPYRIGHT HEWLETT-PACKARD CO. 1984
```

```
DATABASE NAME? ORDERS
```

```
Database has not been modified since last DBSTORE.  
Continue (YES/NO) ? YES
```

```
ORDERS01 has been converted back.  
ORDERS02 has been converted back.  
ORDERS03 has been converted back.  
ORDERS05 has been converted back.  
ORDERS   has been converted back.
```

```
Detail sets don't need to be converted.
```

```
Conversion is done !!!!
```

```
END OF PROGRAM
```

## Discussion

The above example shows conversion of the data base ORDERS back to IMAGE/3000. The user is prompted for the data base name. The conversion program will check the data base "dirty" flag in the root file to determine if the data base has been modified since the last DBSTORE. The user is prompted to continue with the conversion whether or not the data base has been DBSTORE'd. As with DBCONV, the data base administrator must determine whether a backup copy of the data base is necessary.

The backward conversion program works the same way the conversion program to convert from IMAGE to TurboIMAGE does. The program may be run in batch mode or interactively. The job streams to disable ILR and convert either all data bases on the system or data bases in each account can be used with backward conversion also.

## Converting Using Job Streams

Data bases can be prepared for and converted using an Editor use file. This use file creates a master job stream that converts all data bases on a system. This master job stream creates multiple job streams to run DBCONV on the data bases for each group and account on the system. In addition, a job stream that disables ILR for each data base before conversion can be done. An offline listing is produced which provides a listing of each data base converted after the job streams run.

This job stream method provides a convenient way to convert data bases without having to first check data base flags, disable ILR (if enabled), and then convert each data base separately. The package that builds the job streams to convert all IMAGE/3000 data bases to TurboIMAGE data bases consists of files CONVUDC, MASTUSE, TMPCONV1, TMPCONV3, TMPCONVX, and VSTREAM. As recommended earlier in this appendix, all data bases should be backed up prior to running DBCONV.

To start the conversion process using the Editor use file:

1. Log on as MANAGER.SYS,CONVALL.
2. MPE SETCATALOG CONVUDC. (Reset after using the job streams.)
3. Run Editor and enter USE MASTUSE.
4. Respond to the prompts issued by MASTUSE.

The MASTUSE file issues complete on-line instructions and prompts the user for information needed in the conversion process.

MASTUSE streams a job stream called JMASTCON. JMASTCON streams a job stream called JALLACCT which creates a job stream for each account on the system. Note that a job stream will be created for a data base on a private volume only if the private volume has been mounted before the use of MASTUSE. Each job stream has the same name as the account and converts the data bases in that account. The job stream JSTREAMS is also created. This is the master stream file that streams all the account job streams.

JALLACCT creates a job stream even if the account contains no data bases. In this case, the job stream is an empty file. When the user streams JSTREAMS to convert the data bases on the system, the following warning will be issued for each empty file encountered in JSTREAMS:

NO :JOB OR :DATA COMMAND ENCOUNTERED. (CIWARN 1406)

The user can ignore this warning.

## TurboIMAGE Conversion (DBCONV)

The user may either convert all the data bases on the system or convert data bases by account. To convert all the data bases on the system, stream the JSTREAMS job stream. To convert data bases by account, stream the job stream with the name of the account.

The following error messages may occur during the job stream *Jlaccount-name*:

```
END OF FILE (FSERR 0)
FILE SYSTEM ERROR 0 ENCOUNTERED ON LIST FILE. (CIERR 425)
```

If you receive these errors while converting the data bases, perform the following steps:

1. Logon as `MANAGER.SYS,CONVALL`
2. Run `EDITOR` and `TEXT TMPCONV3`
3. Change line 4 from `DISC=5000` to `DISC=X` where:

$$X = 5 + [(\text{\#groups in account}) * 7] + [(\text{\#databases in account}) * 18]$$

4. Keep `TMPCONV3` and exit from `EDITOR`
5. Use `MASTUSE` again to recreate the job streams

The job streams corresponding to account names include jobs to disable ILR. To insure the orderly processing of the jobs, it is recommended to allow one job to be executed at a time. (If converting each data base individually is preferred, then run the DBCONV utility as described in the beginning of this appendix.)

**CAUTION**

DBCONV requires disc space as large as the largest master data set and additional space for chain expansion. If the number of data bases on the system is large, there will be a large number of job streams. It may be necessary to partition the JSTREAMS file accordingly

These job streams may also be used for backward conversion, from TurboIMAGE to IMAGE/3000, with slight modifications. If you want to use the job streams for backward conversion the following minor modifications must be made to the TMPCONV3 file:

1. Change RUN DBUTILB.PUB.SYS to RUN DBUTIL.PUB.SYS (lines 88 and 122)
2. Change RUN DBCONV.PUB.SYS to RUN DBCONV.PUB.SYS,BACKWARD (line 131)

Once these modifications have been made, the job stream can be run and conversion of TurboIMAGE data bases back to IMAGE data bases will begin.

## Error Messages

Two types of error messages are generated by the DBCONV utility. Conditional errors occur before the utility program has begun execution. These errors are described in Table H-1. Unconditional errors occur after successful execution of the utility program has already begun. Unconditional errors are described in Table H-2.

**Table H-1. DBCONV Program Conditional Messages**

MESSAGE	MEANING	ACTION
Bad character for database name	The data base name specified contains an invalid character.	Run DBCONV again with the correct name.
Bad database group reference	The data base group specified is incorrect.	Same as above.
Bad database name reference	The data base name specified is incorrect.	Same as above.
Bad maintenance word	Maintenance word specified is incorrect.	Run DBCONV again and specify the correct maintenance word.
Database group too long	The group name specified is longer than 8 characters.	Run DBCONV again with the correct group name.
Database name too long	The data base name specified is longer than 6 characters.	Run DBCONV again with the correct name.
Maintenance word too long	The specified maintenance word has more than 8 characters.	Same as above.



Table H-2. DBCONV Program Unconditional Messages

MESSAGE	MEANING	ACTION
Database bad, conversion was in process	DBCONV process was interrupted while in progress. The data base may be damaged.	The data base creator should perform an MPE RESTORE of the data base, then run DBCONV again.
Database bad, creation was in process	The data base was damaged while being created. The data base is in a state where it cannot be converted.	The data base creator should purge the data base using DBUTILB. Run DBSCHEMA and then DBUTIL >>CREATE. This will create a TurboIMAGE data base - conversion is not necessary.
Database bad, data set file moving was in process	Process was interrupted while a data set file was being moved (via DBUTIL >>MOVE command). The data base is in a state where it cannot be converted.	The data base creator should do an MPE RESTORE of the data base, then run DBCONV again.
Database bad, erase was in process	Data base was damaged while being erased. The data base is in a state where it cannot be converted.	The data base creator should erase the data base using DBUTILB, then run DBCONV. OR the data base creator can purge the data base using DBUTILB, run DBSCHEMA and then DBUTIL >>CREATE. (Converting the data base will not be necessary if the latter option is used.)
Database bad, ILR disable was in process	Process was interrupted while ILR was being disabled. The data base is in a state where it cannot be converted.	The data base creator should do an MPE RESTORE of the data base, then run DBCONV again.
Database bad, ILR enable was in process	Process was interrupted while ILR was being enabled. The data base is in a state where it cannot be converted.	Same as above.
Database bad, ILR was in process	Process was interrupted while Intrinsic Level Recovery was in progress. The data base may be damaged.	Same as above.

**Table H-2. DBCONV Program Unconditional Messages (Continued)**

MESSAGE	MEANING	ACTION
Database bad, modification was in process	Data base was damaged while being modified in output deferred mode.	Data base creator should 1) MPE RESTORE and run recovery on the data base or, 2) reinstall IMAGE/3000 and DBUNLOAD and erase the data base, install TurboIMAGE, run DBCONV then DBLOAD the data base.
Database has not been created	The data base specified has a root file but no data set files. IMAGE DBSCHEMA has been run, but DBUTIL >>CREATE has not.	The data base creator should purge the data base root file using DBUTILB. Run DBSCHEMA and DBUTIL >>CREATE. Conversion will not be necessary.
Database incompatible	The data base referenced is not a version of IMAGE which can be converted to TurboIMAGE. OR the data base referenced is not a version of TurboIMAGE which can be converted back to IMAGE.	Determine if the version is "HP32215B" OR "HP32215C".
Database is already in the right format	The specified data base has already been converted.	
Database is in an unknown state	The data base is in a state where conversion cannot be performed. The data base may be damaged.	Contact your system manager.
Database maintenance word is required for non-creator	The user invoking the DBCONV program is not the creator and has not supplied the correct maintenance word.	Run the conversion program again and supply the correct maintenance word.
Data entries per chain exceeds IMAGE limit, 65536 entries per chain	Conversion was in process before it terminated. The data base is in an inconsistent state.	The data base creator should perform an MPE RESTORE of the data base, then contact your system manager.
Data items per database exceeds IMAGE limit, 255 items	The data base is in a consistent state, however, it cannot be converted back to IMAGE.	Contact your system manager.
Data items per set exceeds IMAGE limit, 127 items per set	Conversion was in process before it terminated. The data base is in an inconsistent state.	The data base creator should perform an MPE RESTORE of the data base, then contact your system manager.

Table H-2. DBCONV Program Unconditional Messages (Continued)

MESSAGE	MEANING	ACTION
Data sets per database exceeds IMAGE limit, 99 sets	The data base is in a consistent state, however, it cannot be converted back to IMAGE.	Contact your system manager.
File system internal error	This is an MPE file system error.	Contact your system manager.
ILR is on, disable it first		Disable ILR using DBUTILB >>DISABLE command. Run DBCONV again.
Inconsistent Blocking Factor, Media Length or Block Length in Root file for <datasetname>	DBCONV has detected root file corruption.	The root file must be corrected or recreated. Contact your system manager for assistance in correcting the root file, then RUN DBCONV.PUB.SYS again.
Inconsistent Media Length, Path Count, Entry Length in the Root file for <datasetname>	DBCONV has detected root file corruption.	Same as above.
Incorrect database maintenance word	The user invoking the program is not the creator of the data base and has not supplied the correct maintenance word.	Run the conversion program again and supply the correct maintenance word.
Non-creator access is not permitted	The user invoking the program is not the creator of the data base.	Log on with the creator's user name, account and group and run DBCONV again.

## A

Abnormal termination of a procedure, 4-30  
 Abort conditions, A-13, A-31  
 Aborts and recovery, 7-12  
 ABORTS parameter, 8-11  
 Absent list, 2-14  
 Access  
   Calculated, 4-12  
   Chained, 4-12  
   Directed, 4-11  
   Granting, B-1  
   Mode, 2-14  
   Modes and user class number, 4-8, 4-14, 4-16  
   Modes, 4-2  
   Option, 8-43  
   Path, 4-9  
   Serial, 4-11  
 Access class password, modify, 8-56  
 ACCESS option, 8-43, 8-45  
 Account manager, 2-12  
 Account member, 2-12  
 Account protection, 2-12  
 ACTIVATE command, 8-38  
 Activate DBA file, 8-38  
 Actual file designators, 3-14  
 Adding entries, sequence for, 4-7  
 Address, Primary, 10-2  
 Algorithms, primary address calculation, 10-8  
 Array, 2-2  
 ASCII characters, 2-2  
 At sign in DBA file, 9-8  
 Audit trail, 7-12  
 AUTODEFER, 2-19, 8-45  
 Automatic masters, 2-6, 4-19

## B

Backup copies, 1-6, 7-4  
 Backup files, 8-3  
 BASIC, 1-6  
   Examples, 6-61  
 BIMAGE, 6-64  
   Procedure parameters, 6-64  
 BIMAGE interface procedures, 6-61  
 Bit map, 10-4  
 Blocking factor, 2-11  
 BLOCKMAX=option, 3-20  
 Blocks, 2-11, 10-4

Blocks and bit maps, 10-4  
 Broken chains, 8-32  
   Message returned, 8-32  
 Buffer length, 3-23  
 Buffer management, 10-10  
 BUFFSPECS=option, 8-56  
 BUILD command, 7-20  
 Byte, 3-5

## C

Calculated access, 4-12  
 Call statements, Procedure, 5-3  
 Calling errors, A-13  
 Calls to BIMAGE procedures, 6-61  
 Capacity, 2-2  
   Data set, 3-10  
   Master data set, C-1  
   Maximum, 3-11  
 CCE, 4-29, A-12  
 CCG, 4-29, A-12  
 CCL, 4-29, A-12  
 Chain head, 2-4, 10-2  
 Chained access, 4-12  
   And current path, 4-9  
   And locking, 4-13  
 Chains  
   Data, 10-1  
   Definition, 2-4  
   Sorted, C-1  
   Synonym, 10-2  
 CHANGELOG command, 7-23  
 Checking status of a procedure, 4-29  
 Checking the subsystem flag, 4-28  
 Class number, user, 4-2  
 Cleanup mode, MPE, 7-14  
 Closing the data base or data set, 4-28  
 COBOL, 1-6  
   Sample program, 6-14  
 COBOL examples, 6-2  
 COMMAND intrinsic and DS, 9-3  
 Commands, Schema Processor, 3-17  
 Comments in schema, 3-2  
 Communication area of DBG, 10-5  
 Complex numbers, 3-8  
 Compound data items, 2-2  
 Concurrent access modes, 4-3  
 Condition code, 4-29  
 Condition word, 4-29

## Index

- Condition word values, BIMAGE, 6-67
- Conditional locks, D-2
- Continuation records, 3-17
  - Schema Processor, 3-17
- CONTROL, DBRECOV command, 8-11
- CONTROL, Schema Processor Command, 3-20
- Control blocks, 10-7
  - Size of, 10-7
- Control Block, User Local, 4-2
- Control blocks, Overview of, 4-1
- Conventions, language, 3-1
- Converting data base, H-1, H-5
  - Back to IMAGE/3000, H-7
  - Considerations, H-1
  - Disc space requirements, H-1
  - To TurboIMAGE, H-3
  - Using job streams, H-11
- Copying data entries
  - From serial device, 8-4
  - To serial disc, 8-31
- Copying entire data base
  - From serial disc, 8-26
  - To serial disc, 8-28
- Correspondence, Backup copy and logfile, 7-35
- CREATE command, 8-40
- Creating data base, 7-1
- Creating the textfile, 3-15
- Creator, 2-11
  - Data base, 1-8, 3-16
- Creator's log-on group, 2-11
- Critical mode, 2-19
- Current path, 4-9
  - And DBGET, 5-32
  - Definition, 4-12
  - Number and DBCLOSE, 5-7
  - Number and DBFIND, 5-29
- Current record
  - Number and DBFIND, 5-29
  - Rereading, 4-13

## D

- Data, reading the, 4-9
- Data base, 1-3, 2-1
  - Access mode and user class number, 4-8
  - Access modes, 4-2
  - Adding entries, 4-7
  - Backup copy, 7-4
  - Closing the, 4-28
  - Control blocks, 4-1
  - Conversion, H-1, H-3, H-7
  - Creating, 7-1
  - Creator, 1-8, 3-16

- Definition of, 2-1
- Description language, conventions, 3-1
- Designer, 1-8
- Elements, 2-1
- Entering data in the, 4-7
- Manager, 1-8
- Name, syntax, 3-1
- Opening the, 4-1
- Personnel, 1-8
- Protection, 2-12, 5-3
- Recovery options, 7-5
- Remote, 9-1, D-3
- Restructure, 7-2
- Statistics, 7-15
- Structure, 2-1
- Structure, obtaining information, 4-27
- Utilities, 8-1
- Data base administrator, 1-8
  - Access to logfile, 8-24
- Data base backup, prior to conversion, H-6
- Data base creator, 1-8, 2-13
- Data base designer, 1-8
- Data base manager, 1-8
- Data base structure, 2-1
- Data chains, 10-1
- Data entries, 2-2
- Data entry, 2-1
  - And access mode, 4-9
  - And user class number, 4-8
  - Deleting, 4-15
  - Length, 2-2
  - Locking, 4-16, 4-21
  - Numbers, 4-9
  - Sequence for adding, 4-7
- Data files, 2-11
- Data integrity, 2-19
- Data item, 2-1
  - Compound, 2-2
  - Identifiers, 3-9
  - Information about, 4-27
  - Length, 3-5
  - Numbers, 3-9
  - Packed, 3-8
  - Types, 3-6
  - Validating checking, 5-46
- Data names, 3-10
- Data set, 2-1
  - Capacity, 2-2
  - Closing the, 4-28
  - Control block, 10-7
  - Create, 8-40
  - Definition of, 2-2
  - Detail, 2-4
  - Detail, adding entries to, 5-60

- Detail, deleting, 5-12
- Identified, 2-2
- Identifiers, 3-13
- Location, 2-4
- Locking, 4-16
- Master, 2-2
- Master, adding entries to, 5-58
- Master, deleting, 5-12
- Maximum, 3-10
- Purge, 8-52
- Reinitialize, 8-47
- Space allocation, 10-9
- Storage, 2-4
- Summary table, 3-22
- Write list, 2-14
- Data sets, 2-2
  - Detail, 2-4
  - Rewinding, 5-7
- Data types, 2-2
- Data-base-access (DBA) file
  - Activate, 8-38
  - Changing, 9-11
  - Code, 9-11
  - Content, 9-5
  - Deactivate, 8-42
  - Name, 9-8
  - Reporting, 8-65
  - Syntax rules, 9-10
  - Syntax verification, 9-10
  - Using, 9-4
- DBB
  - Allocating buffers, 8-56
  - Use of, 10-5
- DBBEGIN
  - Calling sequence, 5-4
  - COBOL example, 6-14
  - Description, 4-26
- DBCLOSE
  - Calling sequence, 5-6
  - COBOL example, 6-13
  - FORTTRAN example, 6-33
  - Pascal example, 6-46
  - SPL example, 6-52
- DBCONTROL, Description, 5-9
- DBCONV program, H-1
  - BACKWARD, H-7
- DBDELETE
  - Calling sequence, 5-11
  - COBOL example, 6-10
  - FORTTRAN example, 6-29
  - Pascal example, 6-45
- DBEND
  - Calling sequence, 5-14
  - COBOL example, 6-14
- Description, 4-26
- Mode 2 option, 5-15
- DBERROR
  - Calling sequence, 5-16, 6-62
  - COBOL example, 6-13
  - Description, 4-30
  - FORTTRAN example, 6-34
  - Pascal example, 6-47
  - SPL example, 6-52
- DBEXPLAIN
  - Calling sequence, 5-25
  - COBOL example, 6-13
  - Description, 4-30
  - FORTTRAN example, 6-34
  - Pascal example, 6-47
  - SPL example, 6-52
- DBFIND
  - Calling sequence, 5-28
  - Description, 4-9, 4-12
  - Pascal example, 6-43
- DBG, Use of, 10-5
- DBGET
  - Calculated, COBOL example, 6-6
  - Calculated, FORTRAN example, 6-25
  - Calculated, Pascal example, 6-42
  - Calculated, SPL example, 6-53
  - Calling sequence, 5-30
  - Chained, COBOL example, 6-7
  - Chained, FORTRAN example, 6-26
  - Chained, Pascal example, 6-43
  - Chained, SPL example, 6-53
  - Description, 4-9, 4-12
  - Direct, COBOL example, 6-5
  - Direct, FORTRAN example, 6-24
  - Direct, Pascal example, 6-41
  - Direct, SPL example, 6-53
  - Serial, COBOL example, 6-4
  - Serial, FORTRAN example, 6-22
  - Serial, Pascal example, 6-40
  - Serial, SPL example, 6-53
- DBINFO
  - Calling sequence, 5-34
  - COBOL example, 6-12
  - Description, 4-27
  - FORTTRAN example, 6-32
  - Pascal example, 6-46
  - Special uses of, 4-28
  - SPL example, 6-52
- DBLOAD utility, 8-4
- DBLOAD, 2-7
- DBLOCK
  - Calling sequence, 5-40
  - COBOL example, 6-11
  - Description, 4-16

## Index

- FORTTRAN example, 6-30
- Pascal example, 6-45
- SPL example, 6-52
- DBMEMO, Calling sequence, 5-48
- DBOPEN
  - Calling sequence, 5-50
  - COBOL example, 6-2
  - Description, 4-1
  - FORTTRAN example, 6-20
  - Logging, 5-52
  - Pascal example, 6-38
  - SPL example, 6-52
- DBPUT
  - Calling sequence, 5-56
  - COBOL example, 6-3
  - Description, 4-7
  - FORTTRAN example, 6-21
  - Logging, 5-60
  - Pascal example, 6-39
  - SPL example, 6-52
- DBRECOV utility, 8-8
  - ABORT option, 8-8
  - CONTROL command, 8-11
  - EXIT command, 8-15
  - FILE command, 8-16
  - PRINT command, 8-19
  - PURGE option, 8-8
  - RECOVER command, 8-20
  - RESTART option, 8-8
  - ROLLBACK command, 8-22
  - RUN command, 8-24
- DBRECOV, STOP-RESTART feature, 7-41
- DBRESTOR utility, 8-26
- DBS, Use of, 10-5
- DBSTORE flag, 7-4, 7-35, 8-28
  - Override, 8-12
- DBSTORE utility, 8-28
- DBTABLE option, 8-19
- DBU, Use of, 10-5
- DBU (Control Block), 4-2
- DBUNLOAD, 2-7
- DBUNLOAD utility, 8-31
  - And broken chains, 8-32
- DBUNLOAD utility program, 2-7
- DBUNLOCK
  - Calling sequence, 5-61
  - COBOL example, 6-11
  - FORTTRAN example, 6-30
  - Pascal example, 6-45
  - SPL example, 6-52
- DBUPDATE, 5-63
  - Calling sequence, 5-63
  - COBOL example, 6-9
  - Description, 4-14
  - FORTTRAN example, 6-28
  - Locking, 5-63
  - Logging and locking, 5-64
  - Pascal example, 6-44
  - SPL example, 6-52
- DBUTIL utility, 8-37
  - ACTIVATE command, 8-38
  - CREATE command, 8-40
  - DEACTIVATE command, 8-42
  - DISABLE command, 8-43
  - ENABLE command, 8-45
  - ERASE command, 8-47
  - EXIT command, 8-48
  - HELP command, 8-49
  - MOVE command, 8-50
  - PURGE command, 8-47, 8-52
  - RELEASE command, 8-54
  - SECURE command, 8-55
  - SET command, 8-56
  - SHOW command, 8-59
  - VERIFY command, 8-65
- DBUTIL, 1-6
- DEACTIVATE command, 8-42
- Deadlocks, 4-24
  - Prevention, D-1
- Debugging, A-12
  - DBERROR, 5-16
- Defaults
  - CONTROL command, 7-31, 7-39
  - \$CONTROL command, 3-20
- Delete chain, 10-9
- Deleting data, 4-15
- Description language conventions, 3-1
- Design considerations, C-1
- Designer, data base, 1-8
- Detail data sets
  - Adding entries to, 5-58
  - Deleting, 5-12
  - Media records, 10-1
  - Space allocation, 10-9
- Device class, assigning the, 3-11
- Device list, 8-62
- DEVICE option, 8-59
- Directed access, 4-11
- Directed access, reading data, 4-11
- Disable
  - Access, 8-43
  - AUTODEFER, 8-43
  - Dumping, 8-43
  - ILR, 8-43
  - Logging, 8-43
  - Recovery, 8-43
  - ROLLBACK, 8-43
- DISABLE, DBUTIL command, 8-43

DISABLE command, 8-43  
 Display DBUTIL commands, 8-49  
 Displaying information  
   About data base, 8-59  
   About locks, 8-62  
 Distributed Systems (DS/3000), 1-8, 9-1  
 Doubleword integer parameters, BASIC, 6-66  
 DS messages, suppress, 9-6  
 DS user identification, 9-5  
 Dset DBFIND, 5-28  
 Dummy parameters, 5-3  
 Dumping option  
   Disable, 8-43  
   Enable, 8-45  
   Show, 8-59  
 Dynamic locking, 4-16

## E

Enable  
   Access, 8-45  
   AUTODEFER, 8-45  
   Dumping, 8-45  
   ILR, 8-45  
   Logging, 8-45  
   Recovery, 8-45  
   ROLLBACK, 8-45  
 ENABLE command, 8-45  
 Entering data in the data base, 4-7  
 Entries  
   Migrating secondaries, 10-9  
   Primary, 10-2  
   Secondary, 10-2  
 EOF=parameter, 8-11  
 ERASE command, 8-47  
 Erasing the data base, and logging, 7-26  
 Error messages, 1-8  
   DBCONV program, H-11  
 Errors  
   Abort condition messages, A-31  
   Calling, A-15  
   Conditional utility messages, A-32  
   DBCONV messages, H-11  
   Exceptional conditions, A-26  
   File system, A-14  
   Library procedures, A-12, A-14  
   Memory management, A-14  
   Unconditional utility msgs, A-32, A-52, A-57  
 Errors, interpreting, 4-30  
 ERRORS=option (DBSCHEMA), 3-20  
 ERRORS=parameter, 8-11  
 Examples  
   BASIC, 6-61

COBOL, 6-2  
 FORTRAN, 6-20  
 Locking facility, 4-22  
 Pascal, 6-35  
 RPG, 6-80  
 Schema Processor, 3-24  
 SPL, 6-48  
 EXIT command, 8-48  
 Expressions, BASIC, 6-66  
 Extended sort field, 2-9  
 Extents, file, 2-11

## F

Failure to recover transactions, 7-33  
 Failures, system, 7-27, 7-33  
 FILE command, 8-16  
   DBRECOV, 8-16  
   In DBA file, 9-5  
   MPE, 3-14  
 File designators, actual, 3-14  
 File designators, formal, 3-14  
 File errors, Schema Processor, A-1  
 File name, 2-11  
 File system and memory management  
   Error messages, A-14  
 Files, data, 2-11  
 FILETABLE option, 8-19  
 Flags  
   Checking the subsystem, 4-28  
   DBSTORE, 7-4, 7-26, 8-29  
   Disable, 8-43  
   Displaying, 7-35  
   Enable, 8-45  
   Installing logging, 7-19  
   Maintaining logging, 7-25  
 FLAGS option, 8-59  
 Floating-point numbers, 2-2  
 Flow chart, security, 2-17  
 Fmode, 8-16  
 Formal file designator, 3-14  
 FORTRAN examples, 6-20  
 FORTRAN, 1-6

## G

GETLOG command, 7-18  
 Granting a user class access, 2-14  
 Group protection, 2-12  
 Group user, 2-12



## H

Hardware condition code, A-12  
 HELP command, 8-49  
 Host language access, 6-1

## I

I files, 8-45, A-13  
 Identifiers, data item, 3-9  
 ILR for audit purposes, 7-34  
 ILR Quick Reference, G-1  
 ILR, special considerations, 7-7  
 IMAGE/3000 and TurboIMAGE  
   Compatibility, 1-1, H-1  
 Inconsistency, 7-8  
 Information, data base structure, 4-27  
 Initiating data sets, 8-40  
 Integers, 2-2  
 Integrity, 2-19  
 Interactive locking, 4-21  
 Internal structure, 1-8  
   And techniques, 10-1  
 Internal techniques, 10-8  
 Interpreting errors, 4-30  
 Intrinsic Level Recovery, 7-6  
 Intrinsic numbers, 5-3  
 Item part, 3-4

## L

Language conventions, 3-1  
 Languages, examples of, 6-1  
 Language, native, 1-5  
 Library procedure, 1-6  
 Library procedures  
   Abort condition messages, A-31  
   Calling error messages, A-15, A-30  
   Error messages, A-12  
   Exceptional conditions, A-26  
   File system error messages, A-14  
   Memory management error messages, A-14  
   Summary of, 5-2  
 LINES= option, 3-20  
 LIST option and \$PAGE command, 3-18  
 LIST option, 3-20, 3-22  
 Listfile, Schema Processor, 3-14  
 Lock area, 10-10  
 Lock descriptors, 4-17  
   Array format of, 5-44  
   Sort sequence, D-2  
 Locking and transactions, 7-13

Locking levels, 4-19  
 Locking requirements, 7-9  
 Locking/unlocking  
   Access modes, 4-19  
   And chained access, 4-13  
   And direct access, 4-11  
   And logging, 7-12  
   And serial access, 4-12  
   And user classes, 2-12  
   Conditional and unconditional, 4-18  
   Conditional, D-2  
   DBUPDATE, 5-64  
   Deadlocks, 4-24  
   Deciding on a strategy, 4-20  
   During user dialog, 4-21  
   Dynamic, 4-16  
   Examples of, 4-22  
   Interactive dialog, 4-21  
   Internal tables, 4-18  
   Internals, 10-10  
   Issuing multiple locks, 4-24  
   Levels, 4-19  
   Overview, 4-16  
   Performance, C-1  
   Release, 4-24  
   Remote data bases, D-3  
   Shared access, 4-24  
 Locking, 1-8, 4-11  
   Choosing a level, 4-20  
 LOCKS option, 8-59  
 LOG command, 7-22  
 Log file time stamps, 7-26  
 Log identifier  
   Acquiring, 7-18  
   Setting in root file, 7-19, 8-56  
 Log process, Initiate, 7-22  
 Log record, 7-26  
   Formats, MPE, F-1  
   Formats, TurboIMAGE, E-1  
 Logfile, building, 7-20  
 Logging  
   And access modes, 4-7  
   And locking, 7-12  
   And logical transactions, 4-26  
   And process suspension, 4-26  
   Capability, 7-17  
   CHANGELOG capability, 7-23  
   COBOL example, 6-14  
   Cycle, 7-25  
   Cycle, sample job stream, G-5  
   DBCLOSE, 5-7  
   DBDELETE, 5-12  
   DBOPEN, 5-52  
   DBPUT, 5-60

DBSTORE flag, 8-29  
 DBUPDATE and locking, 5-64  
 Description, 7-12  
 Disable, 8-43  
 Disabling, 7-26  
 Displaying status, 7-21  
 Enable, 8-45  
 Erasing the data base, 7-26  
 Flags, 7-19, 7-25  
 Format records, 7-26  
 How it works, 4-25  
 Installation, 7-17  
 Intrinsic, 4-25  
 Intrinsic, calling sequence, 4-26  
 Locking and unlocking, 7-26  
 Maintaining, 7-22  
 Overview, 7-12  
 Recovery blocks, 7-13  
 Sequence of operations, 7-13  
 Special DBEND, 5-15  
 Statistics, 7-15  
 Tape or disc, 7-26  
 Timestamp, 7-26, 8-12  
 To disc, quick reference, G-4  
 To tape, quick reference, G-4  
 Transaction numbers, 4-26  
 What it does, 4-25  
 Logging Device Quick Reference, G-4  
 LOGGING option, 8-43  
 Logical transactions and locking, 7-8  
 Logical transactions, 4-26  
 Logical transactions, defined, 7-8  
 LOGID=option, 8-56, 8-59

## M

Magnetic tape, 2-9  
 MAINT=option, 8-56, 8-59  
 Maintenance word, 2-19  
   Changing or removing, 8-56  
 Making a data base backup copy, 7-4  
 Manager, data base, 1-8  
 Manual master, 2-6  
 Master and detail search items, 3-12, 3-13  
 Master data set, 2-4, 2-6  
   Adding entries to, 5-58  
   Automatic, 2-6  
   Deleting, 5-12  
   Index, 2-4  
   Manual, 2-6  
   Media records, 10-3  
   Space allocation for, 10-9  
 Maximum records in data set, 2-11

Maximum, Data items, 3-4  
 Maximum, data set capacity, 3-11  
 Media records, 10-1  
 Memory management, error messages, A-14  
 Methods, Remote data-base-access, 9-2  
 Migrating secondaries, Entries, 10-9  
 Mirror data base, 7-41  
 MODE4 parameter, 8-13  
 MODEX parameter, 8-13  
 MOVE command, 8-50  
 MPE account and log-on group, 2-11  
 MPE cleanup mode, 7-14  
 MPE disc files, 2-11  
 MPE log record formats, F-1  
 MPE RESTORE command, 2-12  
 MPE subsystem, SORT/3000, 2-9  
 MPE STORE command, 2-12  
 MPE SYSDUMP command, 2-12  
 MPE WRITELOG records, E-1  
 MPE :FILE command, 3-14  
 MR capability, D-1  
 Multiple access, B-1  
 Multiple data base transactions, 7-35  
 Multiple RIN capability, D-1  
 Multiple data base transactions, 7-13

## N

Names  
   data set, 3-10  
   Data base, syntax, 3-1  
   Data, 3-10  
 Native Language Support (NLS), 1-5  
 Nibble, 3-5  
 NOABORTS parameter, 8-12  
 NOLIST option, 3-20  
 NOROOT option, 3-20  
 NOSTAMP parameter, 8-12  
 NOSTATS parameter, 8-13  
 NOSTORE parameter, 8-12  
 NOTABLE option, 3-20  
 Notes on logging, 7-43  
 NOUNEND parameter, 8-11  
 Null list, 2-14  
 Null password, 2-13  
 Null read/write class lists, 2-13  
 Numbers, data item, 3-8

## O

Opening data base more than once, 5-52  
 Opening the data base, 4-1

## Index

### Operating instruction

- Recovery options, 7-5

- Utility programs, 7-1

ORDERS data base, 2-9

Output deferred, 8-7

Output, Schema Processor, 3-22

Overview, TurboIMAGE, 1-1

## P

PAGE command, Schema Processor, 3-18

### Parameters

- Procedure, 5-3

- Unused, 5-3

Pascal examples, 6-35

PASSWORD option, 8-56, 8-59

Password part, 3-3

Password syntax, 3-2

Password, access class, modify, 8-57

Passwords, 1-8, 4-2

Paths, 2-4

- Access, 4-9

- Current, 4-9

- Obtaining information about, 4-27

Performance analysis, TurboIMAGE, C-1

Pointers, 1-3, 10-1

- Data set, 5-32

Post-recovery procedures, 7-37

Primary address, 10-2

- Calculation, 10-8

Primary entries, 10-2

Primary path DBGET procedure, 2-7

Primary path, 2-7, C-1

Prime numbers, C-2

PRINT command, 8-19

Privileged file protection, 2-12

### Procedure

- Abort conditions, A-31

- Call statement, 5-3

- Calls, BIMAGE, 6-62

- Calls, status information, 4-29

- Errors messages, A-12

- Exceptional conditions, A-26

- Parameters, 5-3

- Parameters, BIMAGE, 6-64

Procedure status, checking, 4-29

Procedures, Summary of, 5-2

Process statistics, recovery program, 7-15

Process suspension, logging and, 4-26

Program aborts and recovery, 7-12

### Protection

- Account, 2-12

- Data base, 5-3

- Group, 2-12

- Library procedure, 2-19

- Privileged file, 2-12

- Utilities, 2-19

PURGE command, 8-47, 8-52

Purging the data base, 8-52

## Q

### QUERY

- Data types, 3-8

- Remote data base access, 9-13

QUERY, definition of, 1-3

Quick Reference, Logging Device, G-4

Quiet periods, 7-13

## R

Read and write class list, 2-12, 2-13

### Reading

- Data, calculated access, 4-12

- Data, chained access, 4-12

- Methods, 4-9

- The data, 4-9

Real numbers, 2-2

Record table overflow, 7-27, 8-14

Records, 2-2

- In data set, maximum, 2-11

- Media, 10-1

- Rereading, 4-13

- Size, 2-11

RECOVER command, 8-20

Recovering without a backup copy, 7-36

Recovering, 1-6

### Recovery

- Blocks, 7-13

- Considerations, 7-12

- CONTROL command, 7-31, 7-39

- DBRECOV STOP-RESTART, 7-41

- Determining success of, 7-39

- Determining the success of, 7-31

- Disable, 8-43

- Disabling roll-back, 7-29

- Enable, 8-45

- Enabling roll-back, 7-28

- Enabling roll-forward, 7-34

- FILE command, 7-32, 7-40

- Files, 7-32, 7-40

- Files, staging disc size, 7-13

- From a stream file, 7-12

- ILR, 7-6

- MPE cleanup mode, 7-13

- Multiple data base transactions, 7-13
- Overview, 7-5
- Performing DBRECOV STOP-RESTART, 7-46
- Performing roll-back, 7-29
- Performing roll-forward, 7-36
- Post recovery procedures, 7-37
- PRINT command, 7-32, 7-40
- Quick reference, G-1
- Record numbers, 8-14
- RECOVER command, 7-38
- Roll-back and ILR, 7-28
- Roll-back timestamp, 7-28
- Roll-back, 7-27
- Roll-forward and ILR, 7-34
- Roll-forward, 7-33
- Rollback, 7-27
- RUN command, 7-30, 7-38
- Statistics, 8-12, 8-24
- Stream file, 7-12
- Suppress transactions, 7-13
- Table overflow, 8-14
- Tables, 7-15
- Transferring log files, 7-44
- Recovery file, Statistics, 7-15
- Recovery files, 8-17
- RECOVERY option, 8-43, 8-45
- Reinitialize data sets, 8-47
- RELEASE command, 8-54
- Release data base, 8-54
- Releasing locks, 4-24
- Remote data base, D-3
- Remote data base access, 10-6
  - IMAGE/3000 and TurboIMAGE, 9-1
  - Local application, 9-1
  - Log-on identification, 9-8
  - Methods, 9-1
  - Referencing, 9-11
  - Using QUERY, 9-13
  - Utility programs, 7-1
- Remote Data Base Control Block, 9-1, 10-6
- Rereading data, 4-13
- Resetting data sets, 5-7
- Resource identification number (RIN), D-1
- RESTART option, 8-8
- RESTORE command, MPE, 2-12
- Restoring from backup data base copy, 7-35
- Restrictions, RPG, 6-80
- Restructuring, 1-6
  - Allowed structural changes, 7-2
  - Unsupported structural changes, 7-3
- Rewinding data sets, 5-7
- RIN, 1-8, D-1
- Rmode, 8-16
- ROLLBACK command, 8-22

- Roll-Back Recovery, Quick Reference, G-2
- Roll-Forward Recovery, Quick Reference, G-3
- Root file, 2-11
  - Purge, 8-52
- ROOT option, 3-20
- RPG examples, 6-80
- RPG, 1-6
  - relation to TurboIMAGE, 6-80
- RUN command, 8-24

## S

- Sample job stream
  - DBCONV, H-6
  - Recovery, G-5
  - Roll-Back Recovery, G-7
  - Roll-Forward Recovery, G-6
  - Starting Logging Cycle, G-5
- Sample job streams, recovery and logging, G-5
- Schema
  - Changes, 7-2
  - Comments, 3-2
  - Definition of, 1-5, 3-1
  - Structure, 3-2
- Schema Processor, 2-11, 3-14
  - Command error messages, A-3
  - Commands, 3-17
  - Continuation records, 3-17
  - Creating the textfile, 3-15
  - Errors, 3-24
  - Example, 3-24
  - File error messages, A-2
  - Messages, A-1
  - Operating instructions, 3-14
  - Operation, 3-14
  - Output, 3-22
  - Summary information, 3-22
- Search items, 2-3, 2-4
  - Adding entries, 4-8
  - Creating, 3-13
  - Definition, 2-3
  - Design considerations, C-1
  - Number per detail, 2-4
  - Updating, 4-15
- Secondary address, 10-2
- Secondary entries, 10-2
  - Deleting data, 5-12
- SECURE command, 8-55
- Secure data base, 8-55
- Security flow chart, 2-17
- Security provisions, 2-12
- Security, file system
  - Release, 8-54

## Index

- Secure, 8-55
- Selecting the block size, 3-21
- Sequence for adding entries, 4-7
- Sequence of entries, DBUNLOAD, 8-36
- Serial access, reading the data, 4-11
- SET command, 8-56
- Set part
  - Details, 3-12
  - Masters, 3-10
- Sharing data base, B-1
- SHOW command, 8-59
- Show locks, 8-62
- SHOWLOGSTATUS command, 7-23
- Sort items, 2-7, 3-13
  - Design considerations, C-1
  - Updating, 4-15
- Sort sequence for lock descriptors, D-2
- Sorted chains, C-1
- Sorted entries, maintenance of, 2-7
- Space allocation for
  - Detail data sets, 10-9
  - Master data sets, 10-9
- Special capability, Multiple RIN, D-1
- SPL examples, 6-48
- SPL, 1-6
- Staging disc
  - Altering size of, 7-13
  - Definition, 7-13
  - Recovery file, 7-13
- STAMP parameter, 8-12
- STATS parameter, 8-12
- Status area
  - Information and multiple access, B-1
  - Information, A-12
  - Information, procedure calls, 4-29
  - Register, 4-29
- Status array, 5-3
- Status parameter, 6-67
- STOPTIME parameter, 8-12
- Storage locations, 2-2
- STORE command, MPE, 2-12
- STORE parameter, 8-12
- Storing entire data base, 8-28
- String variables, BASIC, 6-66
- Sub-items, 2-2
  - Count, 3-4
  - Length, 3-4
- Subsystem flag, checking the, 4-28
- Summary
  - Data set table, schema processor, 3-22
  - Description, 3-23
  - Of access modes, 4-3
  - Of library procedures, 5-2
  - Of utilities, 7-1

- Summary information, 3-22
- Summary table description, 3-22, 3-23
- Suppress DS messages, 9-6
- Synonym chains, 10-2
- Synonyms, 10-2
- Syntax, Schema Processor Commands, 3-17
- Syntax errors, Schema Processor messages, A-1
- SYSDUMP command, 2-12
- System failures, 2-19
- System manager, 2-12

## T

- TABLE option, 3-20
- Textfile, Schema Processor, 3-14
- Timestamp
  - Data base, 8-11
  - Log records, 8-11
- Transaction block, 7-8
- Transaction logging, 4-7
- Transaction numbers, logging, 4-26
- Transactions, logical, defined, 7-8
- TurboIMAGE conversion, H-1
- TurboIMAGE enhancements, 1-2
- TurboIMAGE utility program, DBUTIL, 1-6, 2-11
- TurboIMAGE, effect on programs, 1-6
- Type designators, 3-4
  - And programming languages, 3-6
  - Pascal table of, 6-36
  - Table of, 3-5
- Type integer expressions, BASIC, 6-66
- Type integer variable, BASIC, 6-66
- Types, data, 2-2
- Types, uses of, 3-5, 3-8

## U

- Unconditional utility error messages, A-52
- UNEND parameter, 8-12
- Unrecoverable disc, 2-19
- Unrecoverable tape, 2-19
- Unused parameters, 5-3
- Updating data, 4-14
  - Access modes, user class number, 4-14
  - Search and sort items, 4-15
- User class, 2-12, 2-19
  - Locking, 2-19
- User class number, 4-8
  - Deleting data, 4-16
  - Opening the data base, 4-2
  - Setting, 3-3
  - Updating data, 4-14

User class password, 2-13  
 User file label, 2-11  
 User Local Control Block, 2-19, 4-2  
 User type AC or GU, 2-12  
 USERS option, 8-59  
 Utilities, protection, 2-16  
 Utility program  
   Conditional error messages, A-33  
   Error messages, A-32  
   Extended unconditional messages, A-57  
   Protection, 2-16  
   Unconditional error msgs, A-52  
 Utility programs, 1-6  
   Operation, 7-1, 8-3  
   Summary, 7-1

## V

Validity checking, 5-46  
 VERIFY command, 8-65  
 Volume, 8-4

## W

WARMSTART, 7-14  
 Word, 3-5  
 Write class list, 2-12, 2-13  
 WRITELOG records, MPE, E-1

## X

XDBBEGIN, calling sequence, 6-62  
 XDBCLOSE  
   BASIC example, 6-78  
   Calling sequence, 6-62  
 XDBDELETE

  BASIC example, 6-74  
   Calling sequence, 6-62  
 XDBEND, calling sequence, 6-62  
 XDBERROR, BASIC example, 6-79  
 XDBEXPLAIN  
   BASIC example, 6-78  
   Calling sequence, 6-62  
 XDBFIND, calling sequence, 6-62  
 XDBGGET  
   Calculated, BASIC example, 6-71  
   Calling sequence, 6-62  
   Chained, BASIC example, 6-72  
   Serial, BASIC example, 6-70  
 XDBINFO  
   BASIC example, 6-76  
   Calling sequence, 6-62  
 XDBLOCK  
   BASIC example, 6-74  
   Calling sequence, 6-62  
 XDBMEMO, calling sequence, 6-62  
 XDBOPEN  
   BASIC example, 6-68  
   Calling sequence, 6-62  
 XDBPUT  
   BASIC example, 6-69  
   Calling sequence, 6-62  
 XDBUNLOCK  
   BASIC example, 6-74  
   Calling sequence, 6-62  
 XDBUPDATE  
   BASIC example, 6-73  
   Calling sequence, 6-62

## SPECIAL CHARACTERS

\$CONTROL, 3-20  
 \$PAGE, 3-18  
 \$TITLE, 3-19

# READER COMMENT SHEET

HP 3000 Computer Systems

TurboIMAGE  
Reference Manual

32215-90050 December 1985

We welcome your evaluation of this manual. Your comments and suggestions help us to improve our publications. Please explain your answers under Comments, below, and use additional pages if necessary.

Is this manual technically accurate?

☐ Yes ☐ No

Are the concepts and wording easy to understand?

☐ Yes ☐ No

Is the format of this manual convenient in size, arrangement, and readability?

☐ Yes ☐ No

Comments:

This form requires no postage stamp if mailed in the U.S. For locations outside the U.S., your local HP representative will ensure that your comments are forwarded.

FROM:

Date \_\_\_\_\_

Name \_\_\_\_\_

Company \_\_\_\_\_

Address \_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

FOLD

FOLD



NO POSTAGE  
NECESSARY  
IF MAILED  
IN THE  
UNITED STATES

**BUSINESS REPLY MAIL**

FIRST CLASS PERMIT NO. 1070 CUPERTINO, CALIFORNIA

POSTAGE WILL BE PAID BY ADDRESSEE

Publications Manager  
Hewlett-Packard Company  
Information Technology Group  
19447 Pruneridge Avenue  
Cupertino, California 95014

FOLD

FOLD



Part No. 32215-90050  
Printed in U.S.A. 12/85  
E1285

