TurboIMAGE/XL
Database Management System
Reference Manual

HP 3000 MPE/iX Computer Systems

Edition 7

Manufacturing Part Number: 30391-90011
E0300

U.S.A. March 2000

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 or fitness for a particular purpose. Hewlett-Packard shall not be liable for errors contained herein or for direct, indirect, special, incidental or consequential damages in connection with the furnishing 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.

Restricted Rights Legend

Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013. Rights for non-DOD U.S. Government Departments and Agencies are as set forth in FAR 52.227-19 (c) (1,2).

Acknowledgments

UNIX is a registered trademark of The Open Group.

Hewlett-Packard Company
3000 Hanover Street
Palo Alto, CA 94304 U.S.A.

Contents

Contents

Contents

Contents

7. Logging and Recovery

Contents

Figures

Figures

Tables

Tables

Preface

This manual describes the TurboIMAGE/XL Database Management System for the HP 3000 Series 900 computers. It is the reference document for anyone designing, creating, and maintaining a database and for application programmers writing database access programs.

TurboIMAGE/V users will find information and instructions on how to move from TurboIMAGE/V to TurboIMAGE/XL in appendix H.

Designers of TurboIMAGE/XL databases will find knowledge of the HP 3000 MPE/iX operating and file systems useful in determining the amount of system resources, such as disk space and computation time, needed to maintain a specific database. Because access to TurboIMAGE/XL databases 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: BBASIC, C, COBOL II, FORTRAN 77, Pascal, or RPG.

NOTE In this manual, a **word** is a 32-bit storage unit and a **halfword** is a 16-bit storage unit. One **byte** is 8 bits.

MPE/iX

MPE/iX, Multiprogramming Executive with Integrated POSIX, is the latest in a series of forward-compatible operating systems for the HP 3000 line of computers.

In Hewlett-Packard documentation and in talking with HP 3000 users, you will encounter references to MPE XL, the direct predecessor of MPE/iX. MPE/iX is a superset of MPE XL. All programs written for MPE XL will run without change under MPE/iX, and you can continue to use MPE XL system documentation.

You may encounter references to MPE V, an HP 3000 operating system that is not based on the PA-RISC architecture. MPE V software can be run on the PA-RISC (Series 900) HP 3000 computers in what is known as compatibility mode (CM).

What's New in this Edition

TurboIMAGE/XL version C.07.04 or later includes the following enhancements:

Jumbo Data Sets

By using the JUMBO option of $CONTROL, data sets can be greater than 4 GBytes.

B-Tree Indices

Provides the capability for wildcard and range searches. A new chapter 11 was added on this topic.

Dynamic Roll-Back of Multiple Database Transactions

This enhancement is an extension of the dynamic roll-back (DBX) feature to include multiple databases.

Dynamic Master Data Set Expansion

Dynamic master data set expansion allows a master data set to be expanded dynamically (up to a new maximum capacity specified in the root file) during DBPUT when the data set space is exhausted. To facilitate the dynamic master data set expansion enhancement, DBSCHEMA, DBUTIL, and some of the TurboIMAGE/XL intrinsics are enhanced.

Support for TurboSTORE/iX 7x24 True-On-line Backup in DBRECOV

TurboSTORE/iX 7x24 True-On-line Backup can be used to backup the database even when open if the option ONLINE=START or ONLINE=END is used. DBRECOV will recover such a database using logfiles.

Native-mode Utilities

DBUTIL and DBSCHEMA are now native-mode utility programs.

New Modes for DBINFO and DBFIND

DBINFO, DBFIND, and DBCONTROL have additional modes for handling B-Tree index files and master data set expansion.

New DBUTIL Commands and Options

These new DBUTIL commands are added: DETACH, REDO, DO, LISTREDO, ADDINDEX, DROPINDEX, and REBUILDINDEX. A new INDEXED option is added for the SHOW command.

Enhanced Database Integrity

The DBPUT and DBUPDATE intrinsics check the integrity of neighboring entries to detect possible data corruption before inserting into a chain.

Scalability

TurboIMAGE/XL is enhanced to increase the concurrency of modification intrinsics, DBPUT, DBDELETE, and DBUPDATE (Critical Item Update feature ON).

How to Use This Manual

The information in this manual is presented in the order you will use the various TurboIMAGE/XL modules. A text discussion of the overall purpose of a module and definitions of terms used to describe the module precede the reference specifications. Each chapter assumes a knowledge of the material presented in preceding chapters.

Chapter 1 Introduces the TurboIMAGE/XL Database Management System.

Chapter 2 Describes database concepts useful for new users of the TurboIMAGE/XL database structure.

Chapter 3 Discusses design implementations and includes a schema listing for the sample database used throughout the book.

Chapter 4 Provides a discussion on using the database. It is useful for both new and existing TurboIMAGE/XL users.

Chapter 5 Contains the TurboIMAGE/XL procedures with syntax and examples.

Chapter 6 Provides executable sample programs in C, COBOL II, and RPG; provides sample routines in Pascal and FORTRAN 77.

Chapter 7 Discusses database recovery and logging options.

Chapter 8 Contains the TurboIMAGE/XL utilities with syntax and examples.

Chapter 9 Provides information about accessing a database residing on another MPE/iX or MPE V system. Use this chapter if your system has Network Services (NS3000 or NS3000/XL) capability.

Chapter 10 Presents the internal structure of TurboIMAGE/XL elements and methods used to perform certain functions. You need not understand all the material in this chapter to use TurboIMAGE/XL, but refer to it as needed.

Chapter 11 Discusses the key points for the B-Tree index enhancement that is new with this edition. It addresses the changes in TurboIMAGE/XL utilities and intrinsics. It explains how to create and maintain B-Tree indices and perform searches with DBFIND.

Appendix A Contains a description of the error messages issued by the various TurboIMAGE/XL modules.

Appendix B Provides additional information about sharing the database.

Appendix C Contains a summary of important considerations for designing a database.

Appendix D Contains information about the special locking (multiple RIN) capability.

Appendix E Contains TurboIMAGE/XL log record formats to aid in interpreting log and user recovery files.

Appendix F Contains MPE/iX log record formats to aid in interpreting log and user recovery files.

Appendix G Provides a quick reference guide of recovery and logging processes.

Appendix H Contains a detailed discussion of the differences between TurboIMAGE/V and TurboIMAGE/XL.

Other Information Sources

You may need to consult the following manuals:

Migration Process Guide

MPE/iX Intrinsics Reference Manual

MPE/iX Commands Reference Manual

Native Language Programmer's Guide

NLS/3000 Reference Manual

Query/V Reference Manual

NS3000/XL User/Programmer Reference Manual

TurboIMAGE/XL Database Management System DBChange Plus User's

TurboIMAGE/XL Database Management System DBChange Plus

Volume Management Reference Manual

Training

For current information on available training courses, see the *HP Education Catalog*.

1 Introduction

TurboIMAGE/XL is a set of programs and procedures that you can use to define, create, access, and maintain a database. A **database** is a collection of logically-related files containing both data and structural information. Pointers within the database allow you to gain access to related data and to index data across files.

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

- Data security
- Rapid data retrieval and formatting
- Program development
- Program maintenance
- Program file independence
- File consolidation
- Special information needs

This chapter describes each of these topics in detail.

Effective use of TurboIMAGE/XL can remove a large portion of the overhead associated with integrated system design from application analysts and programmers. TurboIMAGE/XL can channel system design talents into functional rather than structurally-supportive design tasks.

Data Security

Conventional file management systems have limited data security provisions. Access to computer readable data can only be denied to individuals with system access by providing physical protection for the media upon which the file is stored, such as using a data vault for storing sensitive data stored on magnetic tape or disk.

TurboIMAGE/XL, in conjunction with MPE/iX, provides security at account, group, data item, and data set levels. Implementing security at the item level allows sensitive data to be stored on-line under the control of TurboIMAGE/XL, a database manager or designer, and system manager, with minimal regard for additional security provisions. TurboIMAGE/XL security provisions can limit programmer or operator access to sensitive information.

Rapid Data Retrieval and Formatting

Conventional file organization frequently requires using multiple file extracts, sorts, and report programs to produce meaningful output. Information requests frequently require weeks to implement, during which time the usefulness of the requested data can decrease.

QUERY/3000, the Hewlett-Packard database inquiry facility, or user-written inquiry programs that use the TurboIMAGE/XL procedures, allow instant interrogation of the database by individuals with access to the system. Other ad-hoc reporting packages are also available.

Program Development

The database structure can be defined and built without using special purpose application level programming. Because control of the linkage portion of the database is under TurboIMAGE/XL software control, the programmer does not need to be concerned with testing the structure and can concentrate on the functional programming task. QUERY/3000 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. More modules of a given system can be tested at the same time.

A specific benefit in the COBOL environment is program coding time. The programmer need only define File Division entries for those files that exist outside the control of TurboIMAGE/XL. 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/XL is implicitly defined in every program that accesses the database. 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 database affect only those programs that process the changed data; no other programs in the system need to be recompiled to reflect the new database structure.

The evolution of the database 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 is not limited by design decisions made during initial system design. The structure of a database can evolve with the needs of the application user. The application designer no longer has to anticipate the needs of the user across the full life of the system.

Program File Independence

Conventional file structures tend to be rigid and inflexible. The nature of conventional file management systems requires that the logic of application programs be intricately interwoven with file design. When it is 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. Because change is the rule rather than the exception in data processing, a large percentage of total time and manpower is spent reprogramming.

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

File Consolidation

Most information processing systems that serve more than one application area contain duplicate data. For example, a vendor's name can 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 database 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. Using logical associations, the data can be used by any program needing it. Because 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.

Special Information Needs

The requirement for one-time information in a format that has never been requested is no longer a problem for data processing users. The user with a special data requirement can get to any subset of information on the database, 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) can be used with TurboIMAGE/XL, which allows character sets other than US ASCII to be used in defining a database and allows data to be sorted in a database according to the local alphabet. NLS enhancements are recognized by four TurboIMAGE/XL utilities: `DBSCHEMA`, `DBUTIL`, `DBUNLOAD`, and `DBLOAD`. For more information on these utilities, refer to the *NLS/3000 Reference Manual* and chapters 6 and 8 in this manual.

Database Personnel

The terms database administrator or database manager, database creator, and database designer can refer to one or more persons involved in administering, creating, or designing the database. The **database administrator** coordinates database use. This person knows the passwords and can authorize others to use the database by making a password available if it is needed for a particular application. The database administrator is also responsible for system backup and recovery. The **database creator** is defined by the MPE/iX 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 database files. The database creator and administrator can be the same person. If not, the administrator will probably have access to the user name and account in which the database resides or to the maintenance word that is defined in chapter 8.

How to Use TurboIMAGE/XL

The following steps summarize how to use TurboIMAGE/XL. Refer to Figure 1-1. for an illustration of each of the steps.

1. **Design the Database.**

   A **database designer** (system analyst) or team of designers determine what data is required by all the application projects that will share the database. 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 database content and structure.

2. **Describe the Database.**

   After the design is complete, it is described using the TurboIMAGE/XL Database Definition Language (DDL). This external definition is called a **schema**. The database creator processes the schema using the TurboIMAGE/XL Schema Processor which creates an internal definition of the database called a root file. The person who creates the root file is identified as the **database creator** and can subsequently create and initialize the database. Chapter 3 contains the description language syntax and operating instructions for the Schema Processor.

3. **Create the Database Files.**

   DBUTIL, a TurboIMAGE/XL utility program, builds the database files according to requirements of the database structure specified in the root file. The files contain no data initially.

4. **Store and Retrieve the Data.**

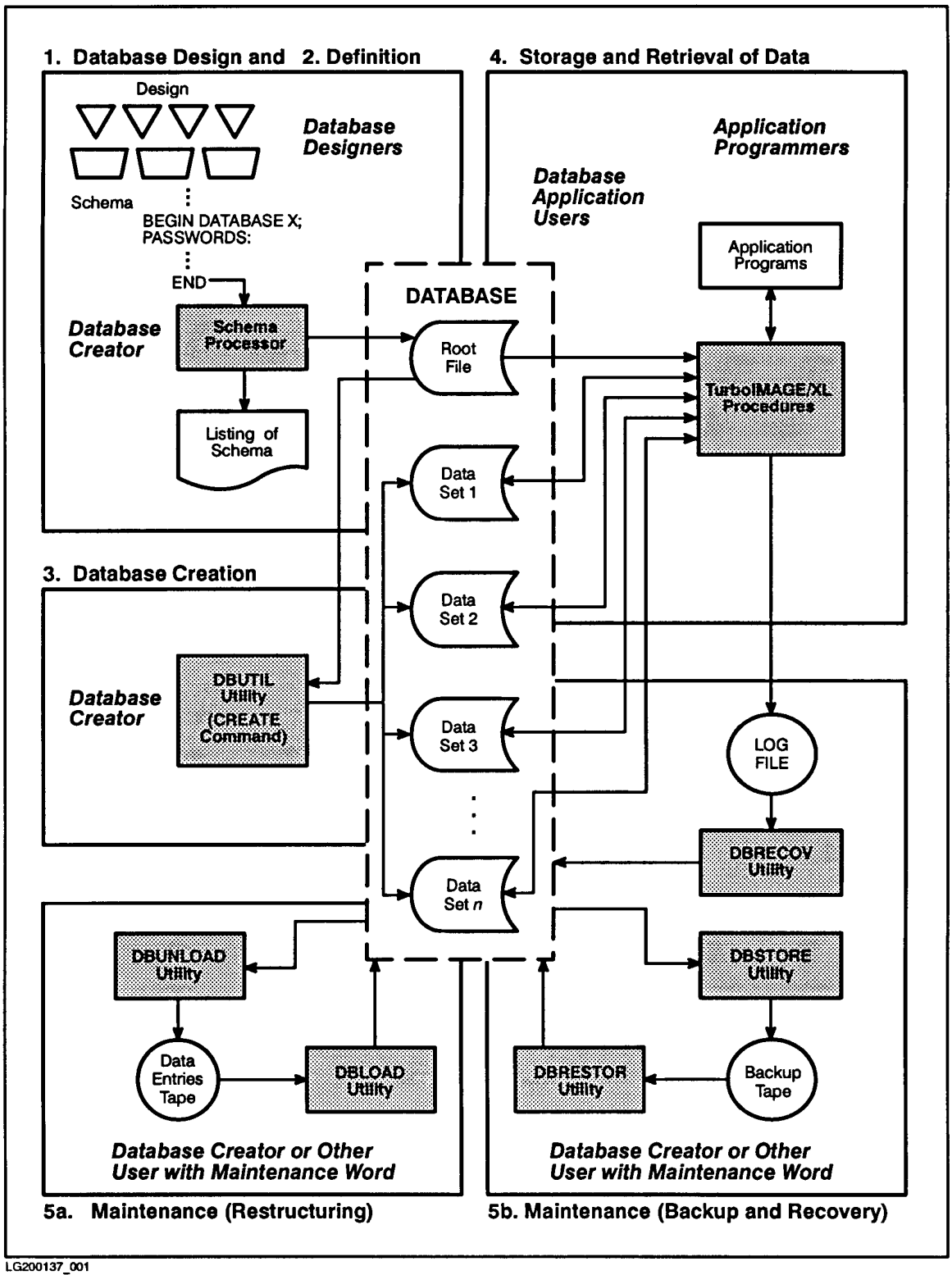
   TurboIMAGE/XL provides a set of library procedures that can be called from BBASIC, C, COBOL II, FORTRAN 77, Pascal, or TRANSACT/V language application programs. The database can also be used with RPG programs but the Report Program Generator issues the calls to TurboIMAGE/XL procedures.

   The application project members can design and write programs in the programming language which best suits their needs and call the TurboIMAGE/XL 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 chapter 5, and examples of calling several procedures from the different languages are given in chapter 6.

5. **Maintain the Database.**

   The TurboIMAGE/XL utility programs can be used to maintain backup copies of the database and perform other utility functions such as logging, recovering, or restructuring the database. These programs are described in chapters 7 and 8. You can also use the TurboIMAGE/XL procedures to write your own maintenance programs, or you can acquire a database restructuring facility.

Figure 1-1. How to Use TurboIMAGE/XL



2 Database Structure and Protection

This chapter describes the structure of a TurboIMAGE/XL database. The discussion defines the elements of a database and describes how they are related and how they can be accessed. If you are designing a TurboIMAGE/XL database, use this chapter with chapter 3 which describes how to define a database.

Database Elements

A **database** is a named collection of related data. The formal description of this data is called a **schema**. The database is defined in terms of **data items**, **data entries**, and **data sets** which are described in the text below.

Data Items

A **data item** is the smallest accessible data element in a database. Each data item consists of a value referenced by a data item name. The name is usually selected to describe the data value. Many data item values can be referenced by the same data item name with each value existing in a different data entry.

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 can have a value. A compound item is similar to an array in programming languages such as FORTRAN 77 and BBASIC. For example, 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.

Critical items are defined as the **key item** in a master data set and the **search** and **sort items** in detail data sets. These are described later in this chapter.

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

Data Entries

A **data entry** is an ordered set of related data items (sometimes referred to as a record). Specify the order of data items in an entry when you define the database. Data entries can be defined with at most 255 data item names; none can be repeated. The length of the data entry is the combined length of the data items it contains and cannot exceed 2348 half-words or 4696 bytes.

Data Sets

A **data set** is a collection of data entries where each entry contains values for the same data items. For example, a customer data set can contain 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.

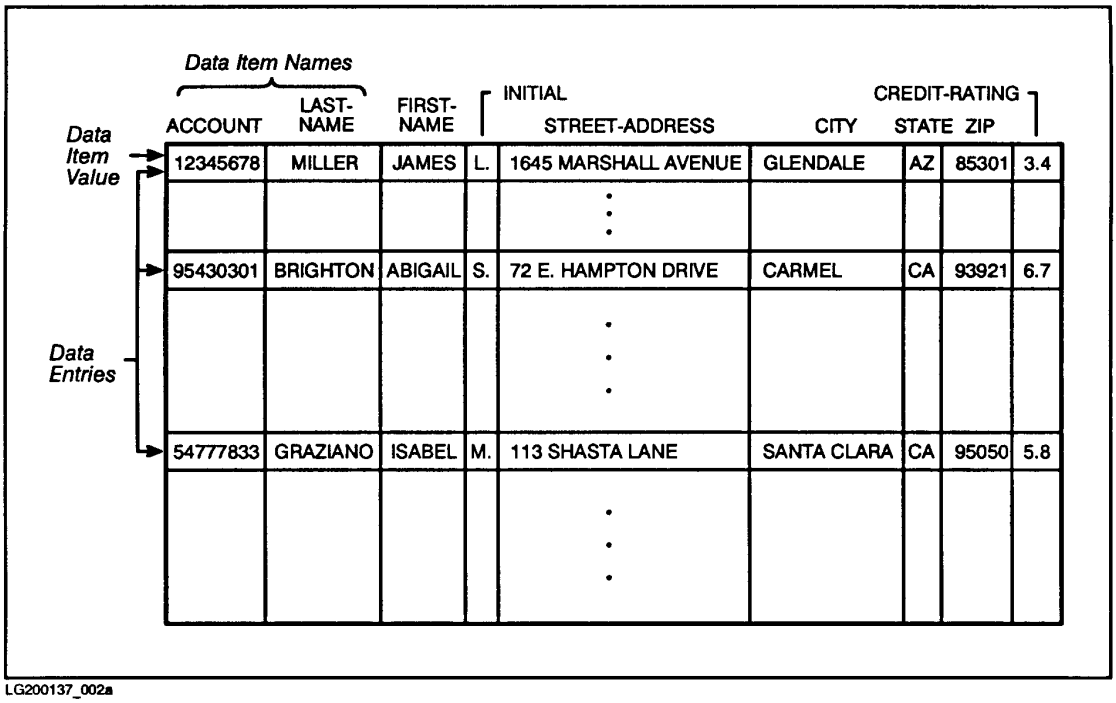
A database can contain up to 199 data sets. Each data set is referenced by a unique data set name that follows certain naming conventions. The data set names are made up of the root file name appended by two characters. For example, if the root file is named *XXXX*, the first data set defined in the schema is named *XXXX*01, the second data set is named *XXXX*02, and so on. To name the maximum of 199 data sets per database, names are incremented from *XXXX*01-99, *XXXX*A0-A9, *XXXX*B0-B9, up to *XXXX*J9.

If the data set is a jumbo data set, the additional files created for the data set are named by appending POSIX extensions such as .001, .002, and so on to the data set name. For example, if the jumbo data set XXXX02 requires two additional files, they are named XXXX02.001 and XXXX02.002. The additional files are called **chunks**.

Each data set is stored in one disk file, or more if a jumbo data set, consisting of storage locations called records. When you describe the database with the database definition language, you specify the maximum **capacity**, or number of records, of each data set. Each record is identified by a record number that can be used to retrieve the entry within it.

Figure 2-1. shows a sample of one data set from a database 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. Many data item values can be referenced by the same data item name if each value exists in a different data entry. For example, the data item FIRST-NAME has the value JAMES in one data entry and ABIGAIL in another data entry.

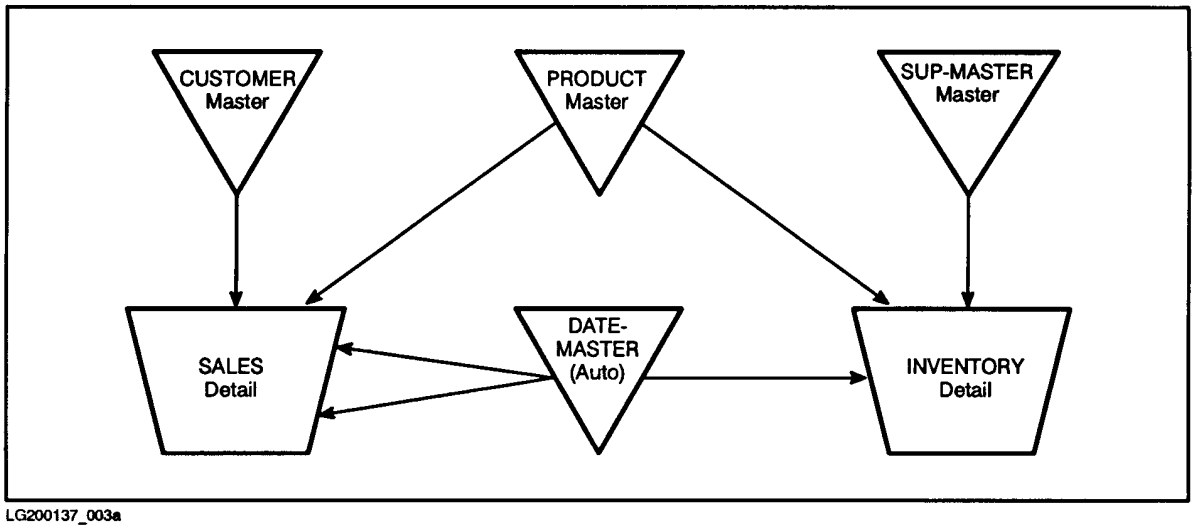
Figure 2-1. CUSTOMER Data Set Sample



Data Set Types and Relationships

A TurboIMAGE/XL data set is either a **master** or a **detail** data set; these data sets are described in this section. Figure 2-2. illustrates the relationships and the types of six data sets in the ORDERS database. Master data sets are identified by triangles and detail data sets by trapezoids.

Figure 2-2. Master and Detail Data Set Relationships



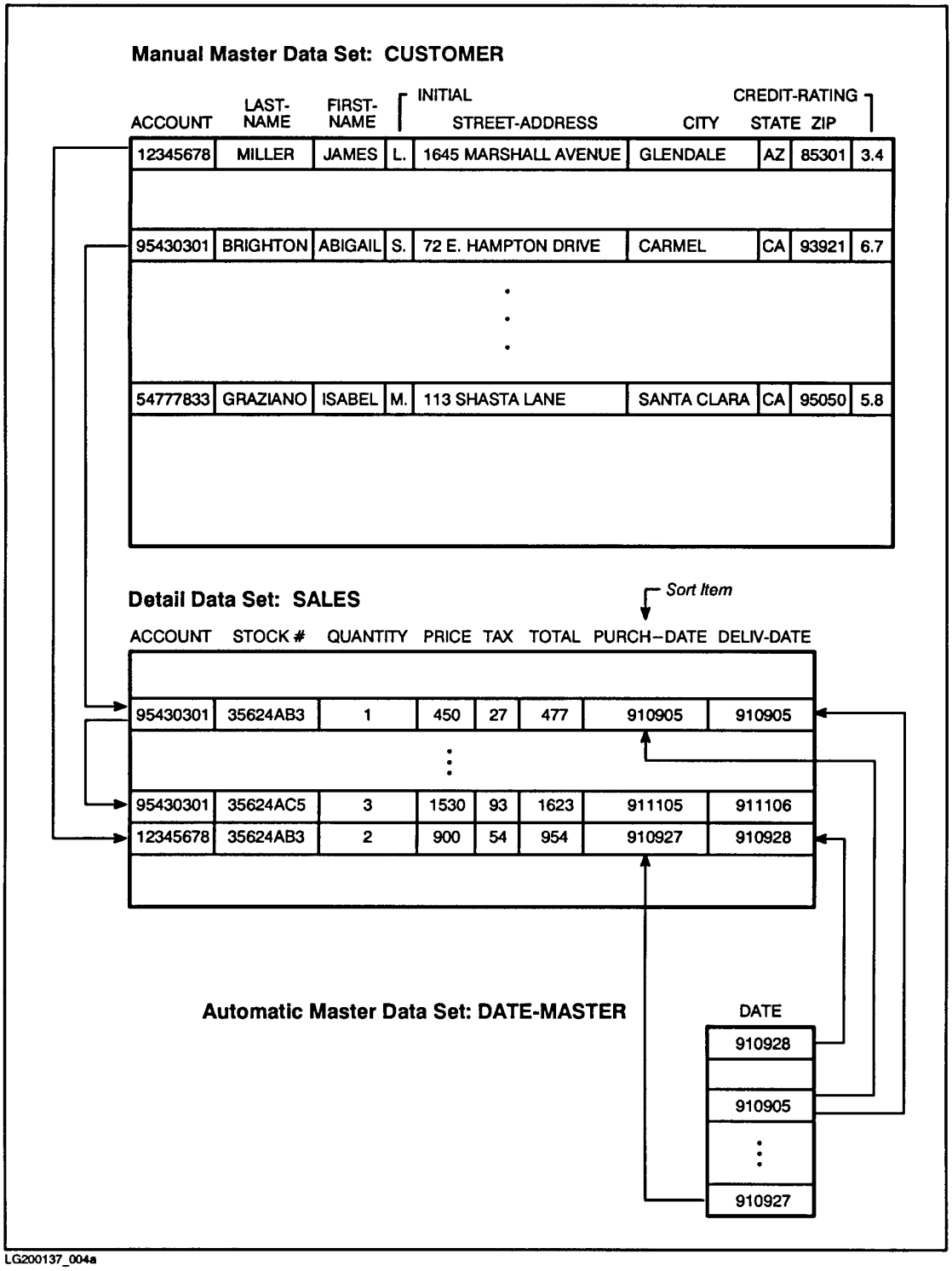
Master Data Sets

Master data sets have the following characteristics:

- They are used to keep information relating to a uniquely identifiable entity. For example, the CUSTOMER data set contains information describing customers.
- They allow for rapid retrieval of a data entry because one of the data items in the entry, called the key item, determines the location of the data entry. A key item *cannot* be a compound item. In Figure 2-3. on the next page, the CUSTOMER data set contains a key item named ACCOUNT. The location of each entry is determined by the value of the customer's account number.
- They can serve as indexes to the detail data set (that is, they can be related to one or more detail data sets). The ACCOUNT key item in the CUSTOMER master data set is related to the ACCOUNT search item in the SALES detail data set in Figure 2-3. The entry for a customer named Abigail Brighton with account number 95430301 serves as an index to two entries in the SALES detail data set which contain information about purchases she made.

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

Figure 2-3. Master and Detail Data Sets Example



Automatic and Manual Masters

A master data set is defined as either automatic or manual. The characteristics of both are described below:

Table 2-1. Manual and Automatic Masters

Manual Master	Automatic Master
Can be standalone (not related to any detail data set) or can serve as an index for one or more detail data sets.	Must serve as an index to one or more detail data sets.
Must contain a key item and can contain other data items.	Must contain only one data item, the key item.
The user must explicitly add or delete all entries. A related detail data set entry cannot be added until a manual master data set entry with matching key item value has been added. When the last detail entry related to a master entry is deleted, the manual master entry still remains in the data set. Before a master entry can be deleted, all related detail entries must be deleted.	TurboIMAGE/XL 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 key item values, a master entry with matching key 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 detail entries have been deleted.
The key item values of existing master entries serve as a table of legitimate search item values for all related detail data sets. A nonstandalone manual master can be used to prevent the entry of invalid data into the related detail data sets.	

In Figure 2-3., CUSTOMER is a manual master data set, and DATE-MASTER is an automatic master data set. 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 910927 and 910928 are automatically added by TurboIMAGE/XL when the detail entry is added to SALES unless they are already in the DATE-MASTER data set.

DATE-MASTER, an automatic master, contains only one data item which is the key item DATE. CUSTOMER, a manual master, contains several data items in addition to the key 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 910905, the DATE-MASTER entry with that value is automatically deleted by TurboIMAGE/XL.

Manual versus Automatic Data Sets

Database designers can choose a manual or automatic master data set depending on the following:

- Manual masters help ensure that valid search item values are entered for related detail entries. They can also serve as indexes to detail data sets.
- Automatic masters serve as indexes to detail data sets and 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 database designer must decide between the control of data entry available through manual masters and the program simplicity offered by automatic masters. For example, because DATE-MASTER is an automatic data set, erroneous dates such as 331299 can be accidentally entered.

Detail Data Sets

Detail data sets have the following characteristics:

- They are used to record information about related events such as 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.
- They can be defined with from zero to 16 search items (unlike a master data set which contains at most one key 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.
- They can be defined for automatic expansion of their capacity.

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/XL stores pointer information with each detail data entry that links all entries with the same search item value. Entries linked in this way form a **chain**. A search item is defined for a detail data set to retrieve all entries with a common search item value (that is, all entries in a chain). The SALES entries with ACCOUNT equal to 95430301 form a two-entry chain. The number of entries in a single chain is limited only by the maximum number of entries in a data set.

Paths

A master data set key 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 key item in CUSTOMER and the ACCOUNT search item in SALES form a path to link the CUSTOMER master data set to the SALES detail data set. 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.

Because a detail data set can contain as many as 16 search items, it can be related to at most 16 master data sets. 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 can be multi-indexed by a single 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 can serve as an index to one or more detail data sets. No master data set can be related to more than 16 detail data sets. For each such relationship, TurboIMAGE/XL 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 data set entry's key 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 chapter 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. TurboIMAGE/XL automatically maintain the chain heads.

Primary Paths

One of the paths of each detail data set can be designated by the database designer as the **primary path**. The main reason for designating a primary path is to maintain the entries of each chain of the path in contiguous storage locations. To maintain contiguous locations, occasionally use the DBUNLOAD utility program to copy the database to tape, the DBUTIL utility program to erase the database, and the DBLOAD program to reload the database from the tape. When the database is reloaded, contiguous storage locations are assigned to entries of each primary path chain. Therefore, the database designer should designate the path most frequently accessed in chained order as the primary path. This type of access is discussed in chapter 5.

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 chapter 5.

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 can have different sort items, and one path's sort item can 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. PURCH-DATE is a meaningful sort item because the dates are stored in a properly form for collating (year-month-day).

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 that TurboIMAGE/XL maintains sorted paths. 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.

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. Note that Native Language Support does not support extended sort items. The only database language that supports extended sort fields is Native-3000 which uses US ASCII. If an extended sort field is used, the sort is done in ASCII collating sequence (negative integers sort higher than positive).

If you depend on extended sort fields to sort a chain, do not call DBUPDATE to modify any of the values in the extended sort fields because the chain will not be automatically resorted according to the new extended sort data values. Instead, call DBDELETE and DBPUT to re-enter the records with modified values. DBUPDATE only recognizes extended sort items when the actual search or sort item is changed. Chapter 5 describes DBUPDATE, DBDELETE, and DBPUT in detail.

If you do not want TurboIMAGE/XL 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 database content is copied to magnetic tape using the TurboIMAGE/XL 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 database, the chains are re-created. Therefore, entries that were previously ordered chronologically will not necessarily be in that same order. The new chronological ordering is based on the order 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. See chapter 8 for more information about DBUNLOAD.

NOTE It is important to limit the use of sorted chains to paths consisting of relatively short chains or chronological sort items that are usually added to the end of chain (for example, date). Sorted paths should not 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/iX HP Sort subsystem.

Jumbo Data Sets

You can create data sets greater than 4 GB in size. A data set of this type, called a **jumbo data set**, can span more than one MPE file. The naming convention uses POSIX extensions. For example, a dataset named SALES03 with four multiple files results in a total of five files with the following names and filecodes:

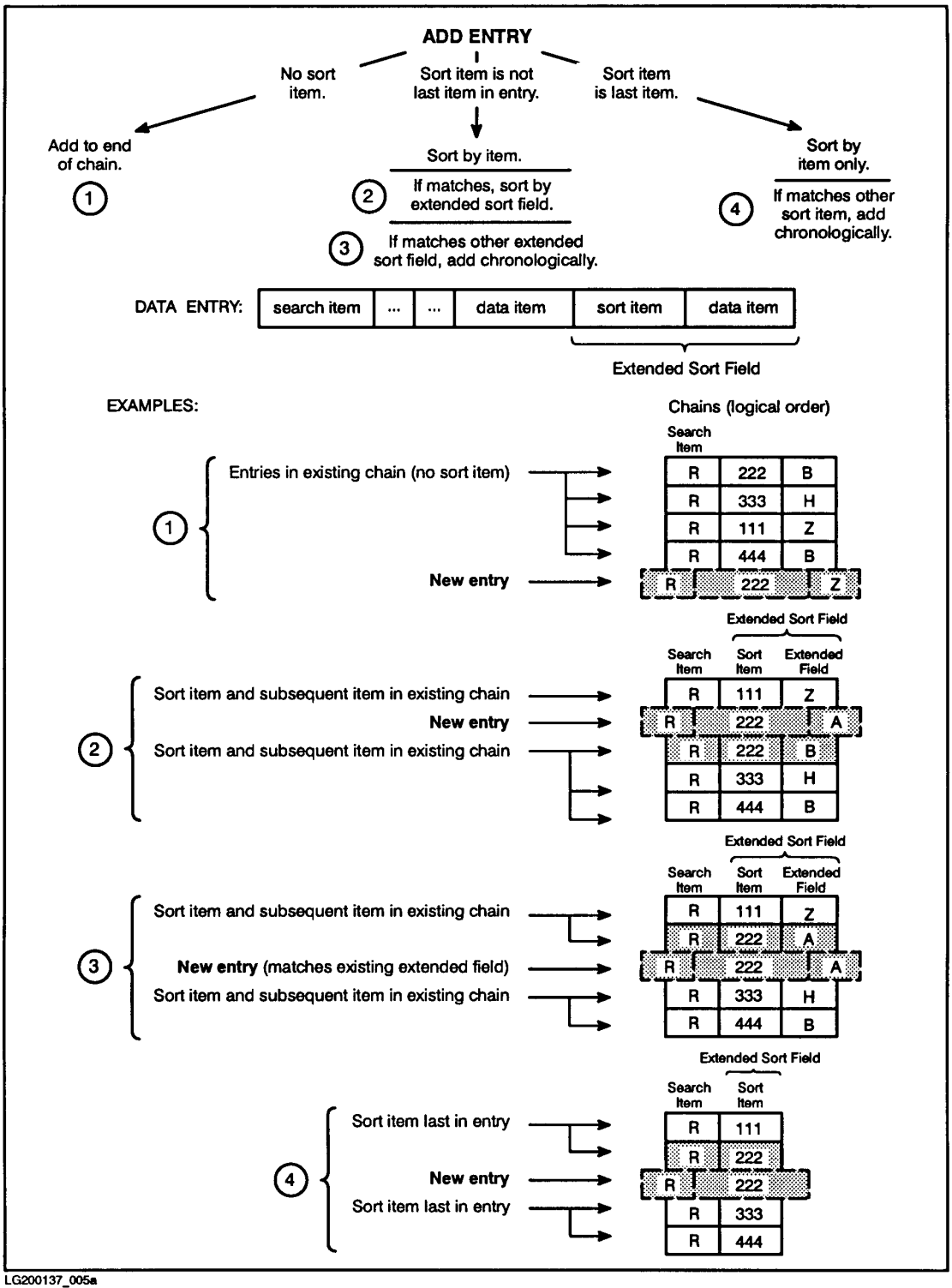
```
SALES03          filecode -408
SALES03.001      filecode -409
SALES03.002      filecode -409
SALES03.003      filecode -409
SALES03.004      filecode -409
```

SALES03 is the **Chunk Control file**. It has information about the other files, but no user data of its own. The remaining files in the data set are called **Chunk Data files** or **chunks**.

To specify a jumbo data set, the JUMBO option must be included in the schema before defining a jumbo data set. Then any data set whose capacity is greater than 4 GB automatically becomes a jumbo data set. Use the NOJUMBO option of DBSCHEMA to turn-off the jumbo option. Then any data set whose capacity is greater than 4 GB will generate an error. For existing databases, you must use a third-party tool to create jumbo data sets.

NOTE Any third-party and diagnostic tools used must support jumbo data sets. DBSTORE does not store a jumbo data set; to store a jumbo data set, use the STORE command, and specify POSIX names.

Figure 2-4. Adding Entries to a Sorted Chain



The ORDERS Database

Figures 2-5 and 2-6 illustrate the complete ORDERS database. Figure 2-5. lists the data items within each data set as defined in the schematic of the ORDERS database shown in Figure 2-2. The data types (in parentheses) are described in chapter 3 with the item part of the schema. Paths are indicated by arrows. CUSTOMER, SUP-MASTER, PRODUCT, and DATE-MASTER are master data sets and SALES and INVENTORY are detail data sets. Figure 2-6. shows a sample entry from each data set and two sample entries for DATE-MASTER.

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 SUP-MASTER and the primary path for SALES is the one defined by PRODUCT.

Figure 2-5. ORDERS Data Sets and Paths

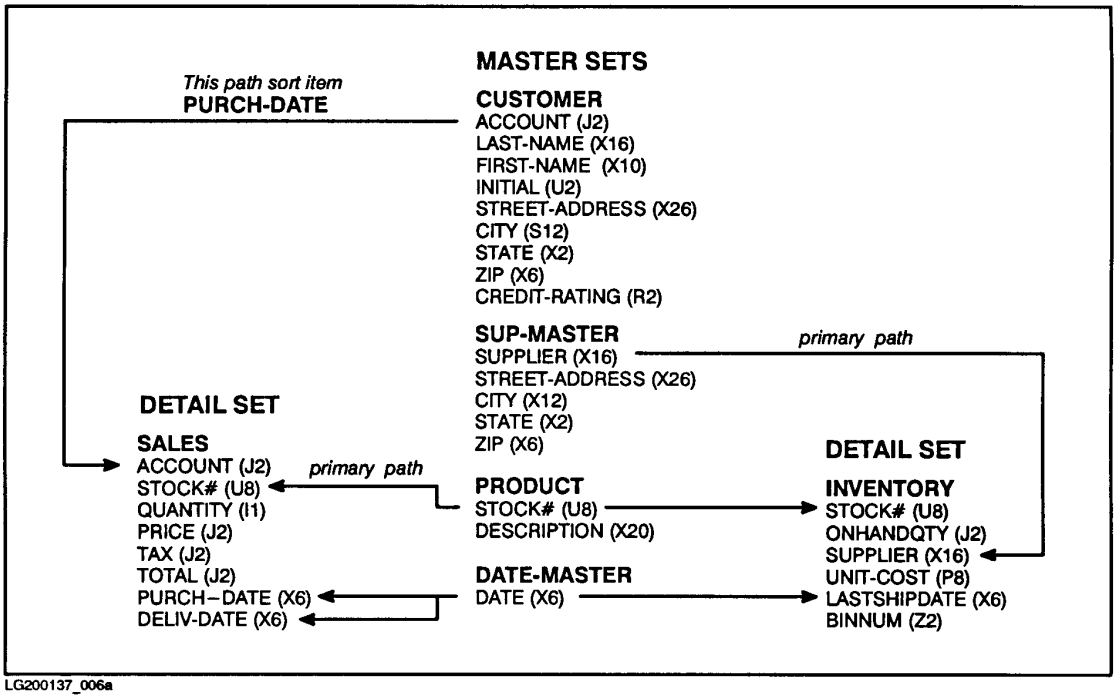
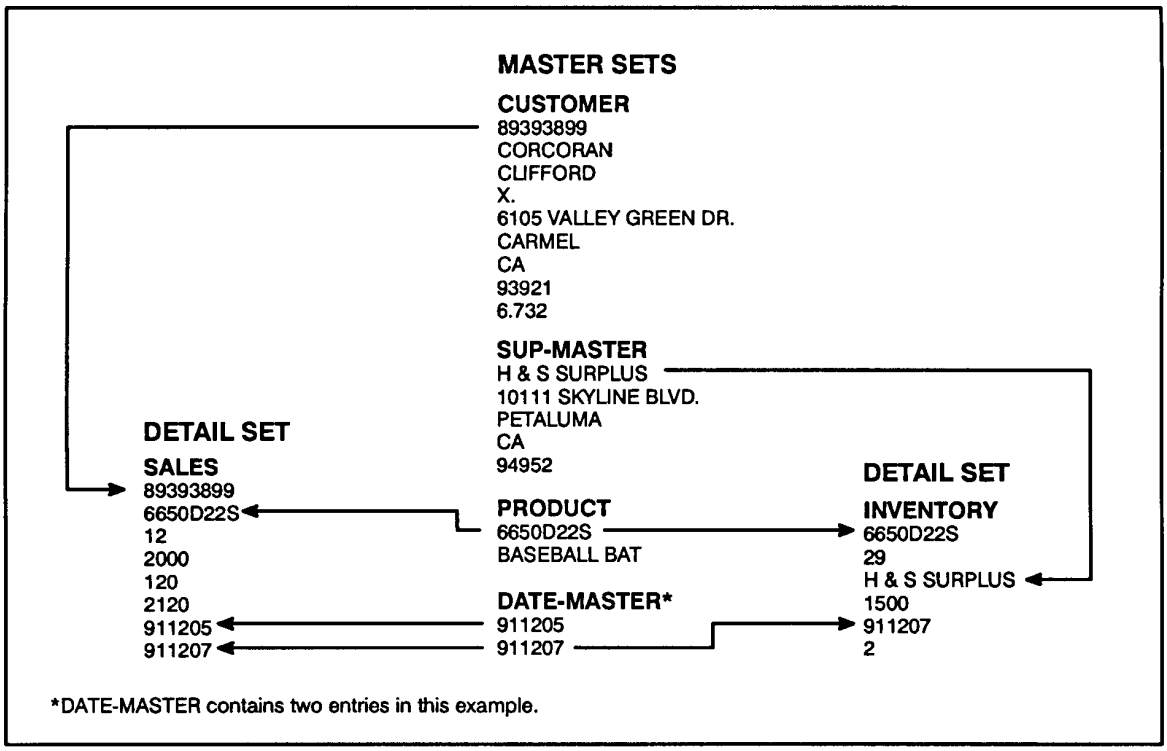


Figure 2-6. A Sample Entry for Each Data Set in the ORDERS Database



LG200137_007a

Database Files

Database elements are stored in privileged MPE/iX disk files. In addition to the **root file** which contains the database definition, other files that contain data are called data sets.

Root File

The **root file** serves as a common point of entry to, and a source of information about, the database. The root file is a single-extent MPE/iX disk file; that is, the entire file occupies contiguous sectors on the disk.

The person who creates the root file is the **database creator** and can subsequently create and initialize the database. The root file is created within the database creator's logon group and account when the Schema Processor is executed. The root file has a local file name identical to the database name. Thus, the name of the root file for the ORDERS database is ORDERS. Refer to the *MPE/iX Commands Reference Manual* for more information about MPE/iX account and logon groups.

Data Files

For non-jumbo data sets, there is one data file for each data set of a database. The size of each record and number of records in the file are defined by the database schema, and that information is recorded in the root file. The data files are created and initialized by the DBUTIL utility.

Each data file is created 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 defined in the schema. For example, the ORDERS database is defined with DATE-MASTER and CUSTOMER as the first two data sets. These data sets are in data files ORDERS01 and ORDERS02. For jumbo sets and sets with B-Tree indices, the chunks or index files are created using POSIX file format. For chunks, the local file name is appended with ".001", ".002", and so on. For example, ORDERS01.001 and ORDERS01.002 for two chunks of ORDERS01. For index files, the local file name of the master is appended with ".idx" (lowercase). For example, ORDERS02.idx for the master set CUSTOMER with a B-Tree index. (Refer to the >>CREATE command of DBUTIL in chapter 8.)

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

Media Record Length

Media record lengths vary between data sets but are constant within each file. Each record is large enough to contain a data entry and the associated TurboIMAGE/XL pointer information. The amount of pointer information depends on the way the data set is defined. Pointer information is described in chapter 10. The maximum number of records in a data set file depends on the media record size, the available disk space, and the

MPE/iX file system constraints.

Blocks

The media records in a data file physically reside together in a group called a **block**. Each block corresponds to one MPE file record. The number of media records in each block is called the **blocking factor**. The Schema Processor determines the blocking factor during creation of the root file. Chapter 3 contains more information about block size and blocking factors. The format of blocks is given in chapter 10.

Protecting the Database

TurboIMAGE/XL prevents unauthorized persons from gaining access to the database. It provides external protection through the MPE/iX privileged file, account, and group constructs and, in addition, provides the database designer and database manager with methods to refine security within the database.

Privileged File Protection

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

Using MPE/iX commands that permit copying TurboIMAGE/XL files to tape represents a potential breach of database privacy, and their use should be controlled. In particular, anyone who uses the MPE/iX `SYSGEN`, `STORE`, or `RESTORE` commands should notify the database manager. The `SYSGEN` and `STORE` commands permit system supervisors, system managers, and other privileged users to copy files to tape. The `RESTORE` command can purge and replace a database file with a different file from tape if the files have the same name.

Account and Group Protection

To gain access to a TurboIMAGE/XL database, you must be able to access the files in the account and group in which the database resides. The system manager and account manager administer the security levels for accounts and groups. The system manager creates accounts, and either the system or account manager creates new groups and users.

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

Defining Database Security

After the data items, data sets, and paths for the database have been defined, database security can be addressed. Defining security involves the following two steps:

1. Defining user classes and passwords
2. Setting up read and write class lists

User Classes and Passwords

Consider who will be using the database. Do all users perform the same tasks or are the tasks varied? Do all users need to read and update the same data items? The answers to these questions will help define how many user classes are needed.

For each type of user, define a password and user class number. Each **user class** is identified by an integer from 1 to 63. Because more than one user at a time can use the same password, you may only need to define a few passwords for your database. You may want to relate the user class number to the user's job position; for example, the ORDERS database is defined with these user classes and passwords:

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

When you initiate access to the database, you must supply a password to establish the user class. If the password is null or does not match any password defined for the database, the user class assigned is zero which has read access to unprotected data sets.

NOTE Because user class 0 has read access, calls requiring read access to an item or set will complete successfully (condition word 0) even if an invalid password was supplied.

The database creator does not need to supply a password. If you are the logged on as the database creator and enter a semicolon in place of a password, you are granted full access to all data sets in the database. TurboIMAGE/XL uses the number 64 to identify the database creator and the numbers 0 to 63 to identify all others.

Read and Write Class Lists

After you have defined user classes and passwords, define the type of access allowed by each password to the data items and data sets in the database. Establish security in the schema by including or excluding the user class numbers in the read or write class list of the data items and data sets, or by omitting a user class list entirely. Omitting a user class list (known as an absent list) has the same effect as including all user classes, including user class 0, in the read class list.

The combinations of the data set and data item user class lists result in one of the following five types of access:

- Write access
- Update access
- Read access
- No access
- Creator-only access

Write Access A user class that has write access can add entries to or delete from the data set. *Write access means that update and read access are also granted*, and is sometimes referred to as *full data access*. To grant write access to a user class for a data set, include the user class number in the write list of the data set. The specified user class then needs to open the database using mode 1, 3, or 4 to take advantage of write access. The user class is ignored if it appears in the user class lists of data items that belong to the data set because write access to a user class at the data set level supersedes that at the item level.

NOTE Database access modes 2, 5, 6, 7, and 8 do not allow write access. Programs that open the database in these modes must pass data set and data item level security. For additional information, refer to chapter 4 and to "Database Access Modes and Data Set Write Lists" later in this chapter.

Update Access A user class that has update access can change the values of a particular data item in an existing data entry. However, the user class cannot add or delete entries from the data set. *Update access means that read access is also granted*. To grant update access to a user class for a data item, include the user class in the read list of the data set and in the write list of the data item. The specified user class then needs to open the database in mode 1, 2, 3, or 4 to take advantage of this type of access. The user class can have update, read, or no access to other data items in the data set depending on the user class lists of the other data items.

NOTE TurboIMAGE/XL provides an option called critical item update (`CIUPDATE`) which lets you update the values of detail data set search and sort items if the database access mode is 1, 3, or 4 and if permitted for the current process. You can restrict update of these data items by assigning read-only access at the set level and controlling write or update access at the item level. See chapter 4 for more information on `CIUPDATE`.

Read Access A user class that has read access can only view the values of a particular data item. To grant read access to a user class for a data item, include the user class in the read class list of the data set and in the read class list of the data item. This user class can have update, read, or no access to other data items in the data set depending on the user class lists of the other items.

No Access A user class that has no access cannot read data item values. No access to a user class can be defined for an entire data set or for specific data items in a data set.

Specify no access for a user class to an entire data set by excluding the user class from the read and write class lists of the data set. To allow no access to a specific data item, include the user class in the read class list of the data set and exclude the user class from the read and write class lists of the data item.

Note that the read or write portions of the user class list can be left empty; this is known as a **null list**. In the example below, only the database creator has write access:

`(11,14/)` The write class list is null.

Creator-Only Access If you specify an empty data set user class list, as shown below, only the database creator has access:

`(/)` This is called a null list.

Sample Read and Write Class Lists Table 2-2. contains sample lists for the CUSTOMER data set and CREDIT-RATING data item in the ORDERS database.

Table 2-2. Sample Read/Write Class Lists

	Read Class List	Write Class List
CUSTOMER (*data set*)	11,14	11,18
CREDIT-RATING (*data item*)	14	14

Because 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 could be included as a reminder in the schema of the total capability granted to user class 14.

Table 2-4. later in this chapter contains examples of the effects of read and write class lists. Note that the examples take into account how the database access mode affects the data set write list.

Null and Absent Lists

A distinction is made between the absence of both read and write class lists (which by default allows read access) 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 can be one of the following:

(/) Both read and write class lists are null.

(11,14/) The write class list is null.

Because the existence of a write class list implies a read class list, even if no user classes are listed in the read list and at least one user class is specified in the write list, the read class list is not considered null.

An absent list and the following null write list, in which the read portion contains all user classes and the write portion is null, yield the same result:

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

The effect of null and absent lists is illustrated in Figure 2-8. later in this chapter.

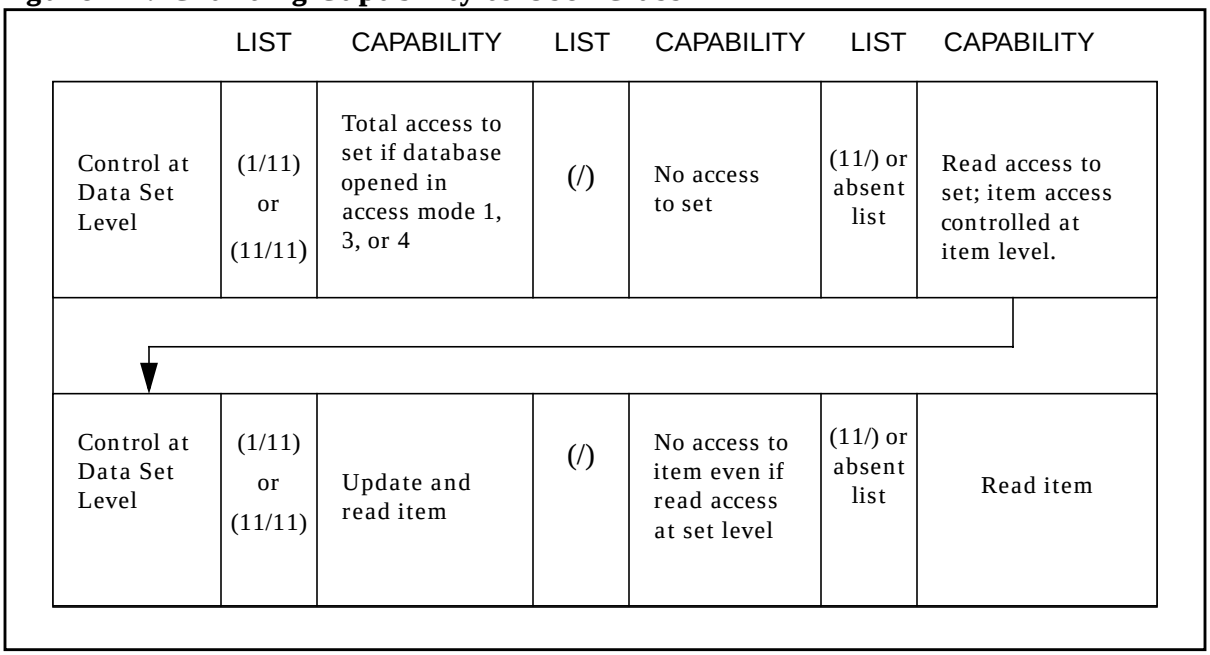
Database Access Modes and Data Set Write Lists

Before you can gain access to a database, you must open it specifying a password that establishes your user class number and an **access mode** that defines the type of database tasks you want to perform. Access modes are described in chapter 4 with the instructions for opening a database. At this time it is necessary only to note that some of the eight available access modes do not allow write or update access even if the user class is allowed these capabilities through the user class lists. If the database is opened in access mode 2, 5, 6, 7, or 8, all data set write class lists are merged into the read class lists, and the merged read class lists are used for all data sets.

Granting a User Class Access

Figure 2-7. and Table 2-3. illustrate the use of read and write class lists from two different perspectives. Figure 2-7. 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.

Figure 2-7. Granting Capability to User Class 11



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

Table 2-3. presents the same rules organized by the task that 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 omitted in the schema (absent lists).

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

Task	Database Access Mode	Data Set\Security Rules	Data Item\Security Rules
Read Data Item	1, 3, or 4	User class must be in data set write list, or	
		User class must be in data set read list and pass data item security.	User class must be in read or write list.
	2, 5, 6, 7, or 8	User class must be in data set read or write list and pass data item security.	User class must be in read or write list.
Update Data Item	1, 3, or 4	User class must be in data set write list, or	
		User class must be in data set read list and pass data item security.	User class must be in write list.
	2	User class must be in data set read or write list and pass data item security.	User class must be in write list.
Add or Delete Data Entries	1, 3, 4	User class must be in data set write list.	

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

- **Specify the user class number in the data set read class list** (or omit both read and write lists entirely). This grants the user class read access to the data set that is controlled at the data item level as described later. If both read and write class lists are absent, the user class is granted this type of access because the lists are (0,1,2,...63/) by default. Opening the database in access mode 2, 5, 6, 7, or 8 is the same as specifying the user class number in the data set read class list only.
- **Specify the user class number in the data set write class list.** If the database 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, and read any item, regardless of the data item read and write class lists. Master data set key item values cannot be updated. Detail data set search or sort item values can be updated if permitted by the critical item update (CIUPDATE) option settings for the database and the current process. A user class number must be in the data set write list in order to add and delete entries. For information about critical item update (CIUPDATE), refer to chapter 4.
- **Exclude the user class number from both the specified read and write class lists of the data set.** This denies the user class any type of access to the data set.

Assuming the database designer has granted only read access 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 read and write class 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. Master data set key item values cannot be updated. Detail data set search or sort item values can be updated in database access mode 1, 3, or 4 if permitted by the critical item update (`CIUPDATE`) option settings for the database and the current process. Because 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 update the value.
- **Exclude the user class number from both the specified read and write class lists of the data item.** 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 database designer must explicitly specify the user class number to allow that class to make any type of change to the database. Read access can be granted by default in some situations, for example, by omitting the lists entirely (also known as absent lists). To deny read access to a data set or data item, the database designer must specify a list and deliberately exclude the user class number.

Figure 2-8. provides a security flowchart. The database has been opened in modify access mode 1, 3, or 4; these are the only allowable access modes for `CIUPDATE` which allows update of detail data set search and sort items. As you read the flowchart, consider the following examples based on the sample ORDERS database:

- Only user classes 11 and 18 can add and delete CUSTOMER data entries because these are the only user class numbers in the data set write list as shown earlier in Table 2-2.. To do so, they must open the database 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. To do so, the database must be opened in access mode 1, 2, 3, or 4.

Figure 2-8. Security Flowchart

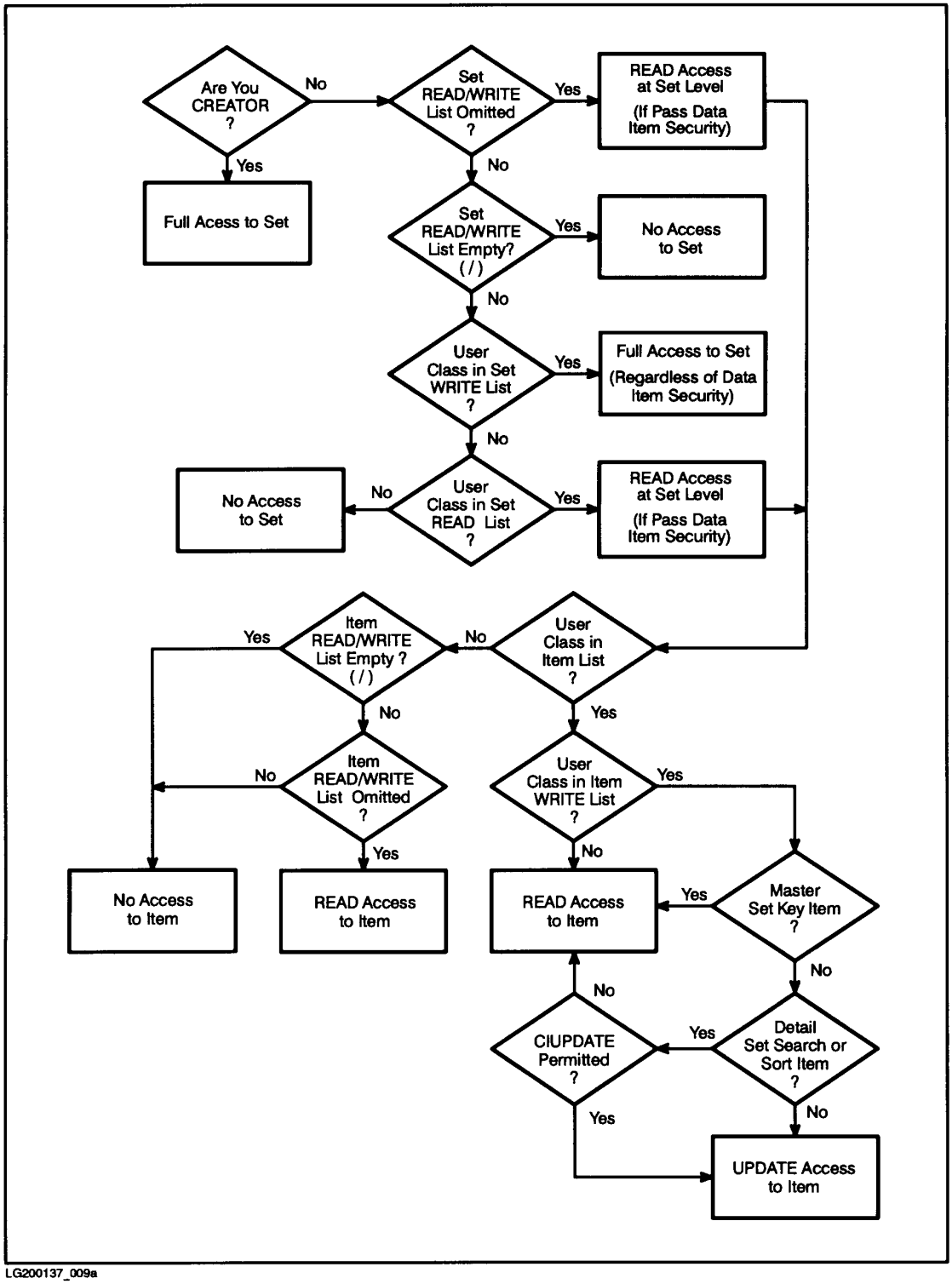


Table 2-4. contains more illustrations of the effects of read and write class lists. These are general examples that are not based on the ORDERS database shown in this manual.

Note that these examples take into account the effects of the access mode in which the database is opened. The database 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 add and delete entries, read all items, and update the values of all items with the following exceptions. Master data set key item values cannot be updated. Detail data set search and sort item values can be updated only in database access mode 1, 3, or 4, and only if permitted by the CIUPDATE option settings for the database and the current process. Note that except where user class 9 is specifically identified in a read and write class list, user class 9 has complete access to data set 1 only when the database access mode is 1, 3, or 4.

Table 2-4. Sample Read and Write Class Lists

Data Item	Read/Write List	Item Read Access	Item Update Access[a]
Data Set 1 (0,18,13/9)			
A		0,13,18,9	9[b]
B	(13/)	13,9	9
C	(/)	9	9
D	(/19)	9	9
E	(18/13)	13,18,9	13,9
F	(/13,18)	13,18,9	13,18,9
G	(12/0)	0,9	0,9
H	(13/)	13,9	9
Data Set 2[c]			
A		0,1,. . .,63	
I	(13/9)	13,9	9

a. User has access only if the database access mode is 1, 2, 3, or 4. For access modes 1, 3, and 4, the CIUPDATE option settings for the database and the current process need to permit updates of any items that are detail data set search or sort items.
b. User has access only if the database access mode is 1, 3, or 4. For update access, the CIUPDATE option settings for the database and the current process need to permit updates of any items that are detail data set search or sort items.
c. Data set 2 has an absent list. The database creator has full access to the data set if the database access mode is 1, 3, or 4. If the database is opened in access mode 2, the user has item update access to all items, except master data set key items or detail data set search and sort items even if the CIUPDATE option settings for the database and the current process permit updates of search and sort item values.

User Classes and Locking

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

Protection in Relation to Library Procedures

All access to a database is achieved through database control blocks that reside in privileged MPE/iX files which are not directly accessible to database users. Because no user process can read or modify these control blocks, TurboIMAGE/XL guarantees protection of the database from unauthorized programmatic access. Refer to the detailed description of these control blocks in chapter 10. For more information about MPE/iX files and privileged mode, refer to the *MPE/iX Intrinsics Reference Manual*.

All TurboIMAGE/XL library procedures that structurally modify the database execute in critical mode. This defers any requested process termination while modifications are in progress. If any file system failures occur during such database modification, TurboIMAGE/XL causes process termination because the database integrity is questionable.

The Database Buffer Area Control Block (DBB) contains pointers to the data set blocks that are used to transfer data (see chapter 10 for additional information). All data set blocks whose contents are changed, reflecting a modification of the database, are always logged by an internal MPE/iX service called Transaction Management (XM) before the library procedure returns to the calling program. This guarantees database integrity despite any program termination that might occur between successive procedure calls. However, deferred output (AUTODEFER) allows the user to override this scheme. When AUTODEFER is enabled, the database does not use MPE/iX Transaction Management. Instead, AUTODEFER uses the MPE/iX file system default recovery mode. This mode keeps data pages in memory for as long as possible until file close time. In this mode, a system failure can cause the loss of database integrity. For more information about AUTODEFER, refer to the >>ENABLE command of DBUTIL in chapter 8.

Protection Provided by the TurboIMAGE/XL Utilities

The TurboIMAGE/XL utilities perform various checks to ensure database integrity:

- They acquire exclusive or semi-exclusive access to the database being processed. (Chapter 4 contains more information about types of access in the discussion of opening a database.)
- Only the database creator or a user supplying the correct **maintenance word** can execute the utilities. The database creator defines the maintenance word when the database is created with the DBUTIL utility (refer to chapter 8). In addition, anyone without system manager (SM) capability intending to use the DBUTIL >>SHOW command or anyone running the utilities other than DBRECOV must be logged on to the group in which the database resides (refer to chapter 8).

  If no maintenance word is defined, only the database creator can execute the utilities. The exception to this rule is that a user with system manager (SM) capability can use the DBUTIL >>SHOW command on any database without having to supply the

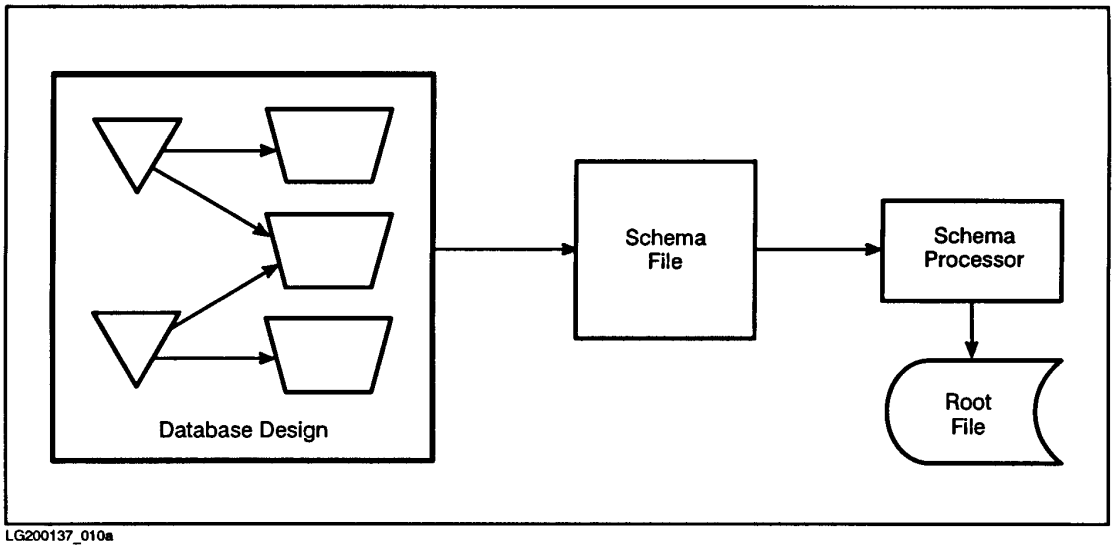
maintenance word.

- Unrecoverable disk or tape problems are treated as functional failures rather than limited successes and result in program termination.

3 Defining a Database

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

Figure 3-1. Database Definition Process



Database Description Language

The database description, called a **schema**, can exist in the MPE/iX system as an ASCII 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 database 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 database language are the same as those described on the conventions page at the beginning of this manual. In addition, these conventions apply.

Table 3-1. Additional Conventions

Convention	Description
Punctuation	All punctuation appearing in format statements must appear exactly as shown.
Comments	Comments take this form: <<comment>> Comments can contain any characters and can 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.
Data Names	Data names can consist of from 1 to 16 alphanumeric characters, the first of which must be alphabetic. Characters after the first must be chosen from this set: Letters A through Z, digits 0 through 9, or + – * / ? ‘ # % & @
Upshifting	All alphabetic input to the Schema Processor is upshifted (converted to uppercase) with the exception of passwords which can contain lowercase characters. Because the Schema Processor upshifts alphabetic characters, programs must specify data set and data item names in all uppercase characters. Take note of this if the programming language you use does not require uppercase characters.

Schema Structure

The overall schema structure is:

```
BEGIN DATABASE database name[,LANGUAGE: language];
PASSWORDS: password part
ITEMS: item part
SETS: set part
```

```
END.
```

The *database 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 database. Refer to the *Native Language Support Programmer's Guide* for further information. The default language is NATIVE-3000 which uses the US ASCII character set.

NOTE When using DBLOAD, the database language must match the language ID stored in the backup media. If they do not match, DBLOAD will give you a warning message in a session, and if you reply **Y**, the DBLOAD will continue. However, in a job, DBLOAD does not load the database.

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

Password Part

The password part defines user classes and passwords. Chapter 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

user class number is an integer between 1 and 63, inclusive. User class numbers must be unique within the password part.

password contains from 1 to 8 ASCII characters including lowercase characters and excluding carriage return, slash, semicolon, and blank. Blanks are removed by the Schema Processor and are not counted in the password length. If you include blanks in your password, the following message is displayed:

```
BLANKS HAVE BEEN REMOVED
```

Example

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

Description

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

item name is the data item name. It must be a valid TurboIMAGE/XL data name as described earlier in Table 3-1.. It must be unique within the item part.

sub-item count is an integer from 1 to 255 that denotes the number of sub-items within an item. If omitted, the sub-item-count equals one by default. A data item whose sub-item count is 1 is a simple item. If the sub-item count is greater than 1, it is a **compound item**.

type designator defines the form in which a sub-item value is represented in the computer. The type designators E, I, J, K, P, R, U, X, and Z are described in Table 3-2. in the section "Data Item Length."

sub-item length is an integer from 1 to 255. It is the number of halfwords, bytes, or nibbles (depending on the type designator) in a sub-item. If omitted, it is equal to 1 by default.

read class list is a group of user class numbers between 0 and 63, inclusive, separated by commas. User class numbers are described in chapter 2.

write class list is a group of user class numbers between 0 and 63, inclusive, separated by commas. User class numbers are described in chapter 2.

Example

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

Description

There can be no more than 1023 data items in a database. 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 database. The complete ORDERS database schema appears at the end of this chapter.

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 can be a **halfword**, a **byte**, or a **nibble**. A halfword is 16 bits, a byte is eight bits, and a nibble is four bits or a half-byte.

NOTE In this manual, a **word** is a 32-bit storage unit and a **halfword** is a 16-bit storage unit. One **byte** is 8-bits.

Table 3-2. describes the type designators and the unit of measure used for each.

Table 3-2. Type Designators

Unit	Bits	Type\ Designator	Description
Halfword	16-bit	E	A real (IEEE floating point) number.
		I	A signed binary integer in 2's complement form.
		J	Same as I, but QUERY/3000 allows only numbers conforming to specifications for COBOL II COMPUTATIONAL data to be entered.
		K	An absolute binary quantity (no negative values).
		R	A real (HP 3000 floating point) number.
Nibble	4-bit	P	A packed decimal number.
Byte	8-bit	U	An ASCII character string containing no lowercase alphabetic characters.
		X	An unrestricted ASCII character string.
		Z	A zoned decimal format number.

A data item must be an integral number of halfwords in length regardless of the type designator and its unit of measure. In other 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, because 4 nibbles equal 1 halfword. 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 halfwords.

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

TurboIMAGE/XL 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/XL does not perform any conversions of data or examine the item to check its validity as it is being added to the database. The only data item values that TurboIMAGE/XL checks are those specified as part of a lock descriptor in calls to the DBLOCK procedure. Refer to the discussion on DBLOCK in chapter 5. There are no rules requiring that a specific type of data defined by a programming language must be stored in

a specific type of TurboIMAGE/XL data item. However, for consistency, R-type items are recommended for storing item values in HP 3000 real format and E-type items are recommended for storing values in IEEE format.

Table 3-3. relates TurboIMAGE/XL type designators and sub-item lengths to the data types typically used to process them in the available programming languages.

Note that the UNIT-COST item (P8) in the INVENTORY data set is easier to process with COBOL II or RPG programs than with the other languages because packed data is a standard data type in COBOL II and RPG. An actual database can 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 can be conveniently processed by programs written in any of the languages.

Table 3-3. TurboIMAGE/XL Data Types and Programming Languages

Data\ Type	BBASIC	C	COBOL II	FORTRAN 77	Pascal	RPG
E2	Short real	float		Real	Real	
E4	Real	double		Double precision	Longreal[a]	
I	Short integer	short int	Computational S9 to S9(4)	Integer*2	–32768..32767 [*subrange*]	Binary
I2	Integer	int	Computational S9(5) to S9(9)	Integer*4	Integer	Binary
I4			Computational S9(10) to S9(18)		Longint	Binary
J	Short integer	short int	Computational S9 to S9(4)	Integer*2	–32768..32767 [*subrange*]	Binary
J2	Integer	int	Computational S9(5) to S9(9)	Integer*4	Integer	Binary
J4			Computational S9(10) to S9(18)		Longint	Binary
K1		unsigned short		Logical	0..65535 [*subrange*]	
K*n*[b]				Logical	Integer[c]	
P4		char[2]	Computational-3 S9(3)		Packed array [1..2] of char[c]	
P8		char[4]	Computational-3 S9(7)		Packed array [1..4] of char[c]	
P*n*			Computational-3 S9(n-1)			Numeric

Table 3-3. TurboIMAGE/XL Data Types and Programming Languages

Data\ Type	BBASIC	C	COBOL II	FORTRAN 77	Pascal	RPG
R2[d]	Short real			Real	Real	
R4[d]	Real			Double precision	Longreal[a]	
U	String	char	Display Picture A	Character	Char [*subrange*]	Character
X	String	char	Display Picture X	Character	Char	Character
X*n*		char[*n*]	Display Picture X(*n*)		Packed array [1..*n*] of char	
Z			Display Picture S9			Character
Z*n*			Display Picture S9(*n*)		Packed array [1..*n*] of char[c]	

a. The Pascal long real is an extension for double-word floating point.
b. For data type K*n*, *n* is a length qualifier ranging from 2 to 255, inclusive. Note that, although the Schema Processor, DBSCHEMA, allows you to define a K*n* data type, not all languages support it. Be sure to select an appropriate data type based on your programming language and report writer requirements.
c. These Pascal declarations provide correct storage allocation.
d. The Schema Processor, DBSCHEMA, allows you to define an E or R data type ranging in length from 1 to 255, inclusive. However, not all languages support these data types. Be sure to select an appropriate data type based on your programming language and report writer requirements. Compiler options are usually required to choose between HP 3000 floating point and IEEE reals. Be sure your program is compiled correctly and check your data values with QUERY/3000.

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/3000, a COBOL II program, or an RPG program. Here is a summary of what happens in each case:

- For values entered using QUERY/3000:

  NO sign specified: Sign is 1111_2

  PLUS sign specified: Sign is 1100_2

  MINUS sign specified: Sign is 1101_2

- For values entered using COBOL II:

  PICTURE clause specifies NO sign: Sign is 1111_2

  PICTURE clause specifies PLUS sign: Sign is 1100_2

PICTURE clause specifies MINUS sign: Sign is 1101_2

- For values entered using RPG:

  NO sign or PLUS specified: Sign is 1100_2

  MINUS sign specified: Sign is 1101_2

When using TurboIMAGE/XL to locate all packed data items with a particular value (as described under "DBLOCK" in chapter 5), you must be aware that TurboIMAGE/XL 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/XL will not retrieve any items with the unsigned value 2.

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

Complex Numbers

Applications programmed in BBASIC or FORTRAN 77 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.

Business BASIC Decimal Numbers

BBASIC decimal numbers should be stored as K2 or K4 data types if QUERY/3000 is to be used on the decimal numbers. QUERY/3000 processes these decimals via a run-time parameter.

QUERY/3000 and Data Types

QUERY/3000 supports only a subset of the available data item types. If you intend to use QUERY/3000, you should consult the *QUERY/V Reference Manual* for specific information about the way QUERY/3000 handles the various TurboIMAGE/XL data types, including compound data items.

Data Item Identifiers

When using the TurboIMAGE/XL procedures described in the chapter 5, 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 because a change in the order of the item definitions or deleting 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.

Table 3-4. shows examples of item parts.

Table 3-4. Examples of an Item Part

Item Example	Description
`A,I2;`	32-bit signed integer.
`MELVIN,3I(1,20/44);`	Compound item. Three single halfword signed integers. Read classes are 1 and 20; write class is 44. (Write classes can also read.)
`BLEVET,J;`	Single halfword signed integer between –9999 αvδ 9999 (XOBOΛ II).
`COSTS,2X10;`	Compound item. Two 10-character ASCII strings.
`DATE,X6;`	Six-character ASCII string.
`VALUES,20R2(1/8);`	Compound item. 20 two-halfword real (HP 3000 floating point) numbers. Read class is 1; write class is 8. (Write classes can also read.)
`PURCHASE-MONTH,U8;`	Eight-character ASCII string with no lowercase alphabetics.
`MASK,K2;`	32-bit absolute binary quantity.
`TEMPERATURE,17R4;`	Compound item. 17 four-halfword real (HP 3000 floating point) numbers.
`SNOW*#@,Z4;`	Four-digit zoned decimal (numeric display) number.
`POPULATION,P12;`	Eleven decimal digits plus a sign in the low order nibble. Occupies three halfwords.
`ATOMIC-WEIGHT,E4;`	64-bit IEEE real.

Set Part

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 key and search items. In addition, it defines security at the set level and at the item level within each data set.

Master Data Sets

The set part syntax and parameters for master data sets are provided below.

Syntax

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

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

{CAPACITY:
 C:        }maximum capacity [(blocking factor)][,initial capacity
                                                 [,increment]      ];
```

Parameters

set name is the data set name. It must be a valid TurboIMAGE/XL data name as described earlier in "Data Sets" in chapter 2 and in the discussion of the DBUTIL >>CREATE command in chapter 8. A maximum of 199 data sets, including both masters and details, is allowed.

MANUAL (or M) denotes a manual master data set. Each entry within a manual master must be created manually and can 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/XL and contains only one data item.

/INDEXED denotes that a B-Tree index is desired for the master data set key item.

read class list is a group of user class numbers between 0 and 63, inclusive, separated by commas. User class numbers are described in chapter 2.

write class list is a group of user class numbers between 0 and 63, inclusive, separated by commas. User class numbers are described in chapter 2.

device class is the class name of the MPE/iX device on which the data set resides. The device must be a member of the volume set on which the database resides.

item name is the name of a data item defined in the item part.

path count is an integer between 0 and 16, inclusive, which is used with the key item only. It indicates the number of paths that will be established to various detail data sets. Refer to chapter 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 can be used with a manual master data item to indicate the key 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.

maximum capacity is the maximum number of entries the data set can contain: that is, the data set's capacity. It must be less than or equal to 2^{31} –1 (2,147,483,647) and is limited by the size of the entry as well as the maximum size of an MPE/iX file, jumbo or non-jumbo.

For non-jumbo data sets, if the capacity expansion parameter, initial capacity, is specified and is less (not zero) than the maximum capacity, the data set is enabled for dynamic expansion. Note that specifying initial capacity for a jumbo data set will generate an error. When enabled for dynamic expansion, the maximum capacity is adjusted by TurboIMAGE/XL to represent an even multiple of the blocking factor. Otherwise, it remains unchanged.

blocking factor is the number of data set media records in one block. If a value is not specified, it is calculated by DBSCHEMA.

initial capacity is the initial capacity for the data set, that is, the number of entries for which space will be allocated and initialized when the data set is created. This number must be between 1 and 2^{31} –1 inclusive but must be less than or equal to the maximum capacity. Specifying initial capacity for a jumbo data set will generate an error. This parameter should be used to closely approximate the current volume of data. If it is very low, there can be frequent expansions leading to severe disk fragmentation. If it is very high, DBPUT may take a long time to complete which could impact other database users. The initial capacity is adjusted to represent an even multiple of the blocking factor. Initial capacity is an optional parameter. If initial capacity is not specified, or if initial capacity is either zero or equal to the maximum capacity, then dynamic capacity expansion is not enabled for the data set, and maximum capacity is used for the data set file creation.

increment is either the number of entries or the percentage of the initial capacity by which the data set will be expanded each time its initialized space is exhausted. If a percentage is used, the percent sign (%) must follow the incremental amount. This increment parameter can only be used if the initial capacity parameter is also specified. This number must be 1 to 32767 inclusive for percent, or 1 to 2^{31} –1 (2,147,483,647) inclusive for number of entries. If it is very low, there can be frequent expansions leading to severe disk fragmentation. If it is very high, DBPUT may take a long time to complete which could impact other database users.

The number of entries defined, or the entries calculated from the percent, cannot exceed the maximum entry count minus the initial allocation. The increment is adjusted to represent an even multiple of the blocking factor. The increment is an optional parameter. If the increment is not specified for the data set, or is zero, but the initial capacity is greater than zero, then the increment for each expansion is defaulted to ten percent (10%) of the initial capacity for the data set. If the initial capacity is equal to the maximum capacity, or the initial capacity is zero, then this indicates the data set cannot be expanded and increment is ignored.

Examples

```
NAME:       SUP-MASTER,MANUAL /INDEXED (13/12,18),DISC1;
ENTRY:      SUPPLIER(1),
            STREET-ADD,
            CITY,
            STATE,
            ZIP;
CAPACITY:   2001,501,20%;
```

Description

The example also 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/XL database and can aid in better use of system resources. An understanding of how to spread the data sets over multiple disk devices can 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 can have up to eight names).

To retrieve information on where each data set resides after specifying device classes in the schema, you can use the MPE/iX command LISTF,3 (after the database is created). This command lists the device type, logical device number, and the device class name for each data set in the database. The `DBUTIL >>SHOW` command can also be used to display the devices on which data sets reside.

The data set SUP-MASTER will have a B-Tree index created on the key item, SUPPLIER. B-Tree `DBFIND`s can be done using SUPPLIER for SUP-MASTER. Also, B-Tree `DBFIND`s will be allowed using all of its corresponding search items in the detail sets. In the above example, SUPPLIER has only one path. Hence, B-Tree `DBFIND` can be done using the related search item and the detail data set. The presence of capacity expansion parameters indicate that it is enabled for dynamic data set expansion.

The example also shows that the maximum capacity is 2001, the initial capacity is 501, and the increment is 20%.

Another example is capacity defined as follows:

```
CAPACITY: 2001,501,25;
```

The maximum capacity is set at 2001, has an initial capacity of 501, and will be automatically expanded by 25 when the initialized space is exhausted.

To allow dynamic expansion for a master data set, specify the maximum capacity and the initial capacity when defining the data set. If dynamic expansion is not needed for the data

set, the maximum capacity is the only required parameter.

Verify that there is enough disk space for a data set to be expanded. Performance may be impacted by the number of entries incremented when a master data set is dynamically expanded. The number of disk extents used for the data set file may also impact the performance of TurboIMAGE/XL.

NOTE For existing databases, use DBChange Plus or a third-party utility to specify the expansion parameters.

Detail Data Sets

The set part syntax and parameters for detail data sets are provided below. The maximum capacity, the initial capacity, and the incremental amount are new parameters for the detail data set CAPACITY definition. These parameters allow a detail data set to be expanded dynamically (up to a new maximum capacity specified in the root file) during DBPUT when the detail data set space is exhausted.

Syntax

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

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

{CAPACITY:
 C:        }maximum capacity [(blocking factor)][,initial capacity
                                                  [,increment]      ];
```

Parameters

set name is the data set name. It must be a valid TurboIMAGE/XL data name as defined in "Data Sets" in chapter 2 and in the discussion of the DBUTIL >>CREATE command in chapter 8. A maximum of 199 data sets, including both masters and details, is allowed.

DETAIL (or D) denotes a detail data set.

read class list is a group of user class numbers between 0 and 63, inclusive, separated by commas. User class numbers are described in chapter 2.

write class list is a group of user class numbers between 0 and 63, inclusive, separated by commas. User class numbers are described in chapter 2.

device class is the class name of the MPE/iX device on which the data set resides. The device must be a member of the volume set on which the database resides.

item name is 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 can be search items.

(Refer to *master set name* 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.

master set name is the name of a previously defined master data set. When a master set name follows an item name, 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 halfwords.

sort item name is 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 chapter 2.)

maximum capacity is the maximum number of entries allowed in a data set (data capacity). It must be less than or equal to 2^{31} –1 (2,147,483,647) and is limited by the size of the entry as well as the maximum size of a data set, jumbo or non-jumbo. This number can 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. Selecting a very large maximum capacity minimizes the chances that the set will run out of space.

For non-jumbo data sets, if the capacity expansion parameter, initial capacity, is specified and is less (not zero) than the maximum capacity, the data set is enabled for dynamic expansion. Note that specifying initial capacity for a jumbo data set will generate an error.

blocking factor is the number of data set records in one block. If a value is not specified, it is calculated by DBSCHEMA.

initial capacity is the initial capacity for the data set, that is, the number of entries for which space will be allocated and initialized when the data set is created.

This number must be between 1 and 2^{31} –1 inclusive but must be less than or equal to the maximum capacity. Specifying initial capacity for a jumbo data set will generate an error. This parameter should be used to closely approximate the current volume of data. If it is very low, there can be frequent expansions leading to severe disk fragmentation. If it is very high, DBPUT may take a long time to complete which could impact other database users. The initial capacity is adjusted to represent an even multiple of the blocking factor. Initial capacity is an optional parameter. If initial capacity is not specified, or if initial capacity is either zero or equal to the maximum capacity, then dynamic capacity expansion is not enabled for the data set, and maximum capacity is used for the data set file

creation.

increment is either the number of entries or the percentage of the initial capacity by which the data set will be expanded each time its initialized space is exhausted. If a percentage is used, the percent sign (%) must follow the incremental amount. This increment parameter can only be used if the initial capacity parameter is also specified. This number must be 1 to 32767 inclusive for percent, or 1 to 2^{31} –1 (2,147,483,647) inclusive for number of entries. If it is very low, there can be frequent expansions leading to severe disk fragmentation. If it is very high, DBPUT may take a long time to complete which could impact other database users.

The number of entries defined, or the entries calculated from the percent, cannot exceed the maximum entry count minus the initial allocation. The increment is adjusted to represent an even multiple of the blocking factor. The increment is an optional parameter. If the increment is not specified for the data set, or is zero, but the initial capacity is greater than zero, then the increment for each expansion is defaulted to ten percent (10%) of the initial capacity for the data set. If the initial capacity is equal to the maximum capacity, or the initial capacity is zero, then this indicates the data set cannot be expanded and increment is ignored.

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: 1008,504,112;
```

Description

The example above shows the detail data set SALES which will reside on Disc1. The maximum capacity is 1008, the initial capacity is 504, and the increment is 112.

Another example is a detail data set defined as follows:

```
CAPACITY: 500000(10),10000,25%;
```

The maximum capacity is set at 500000 with a blocking factor of 10. It has an initial capacity of 10000, but will be automatically expanded by 2500 when the initialized space is exhausted. Since the incremental amount is defined as a percent, the percent is calculated as a constant number based on the initial (or original) capacity (25% of 10000 is 2500.)

To allow dynamic expansion for a detail data set, specify the maximum capacity and the initial capacity when defining the data set. If dynamic expansion is not needed for the data set, the maximum capacity is the only required parameter.

Verify that there is enough disk space for a data set to be expanded. Performance may be impacted by the number of entries incremented when a detail data set is dynamically

expanded. The number of disk extents used for the data set file may also impact the performance of TurboIMAGE/XL.

NOTE For existing databases, use DBChange Plus or a third-party utility to specify the expansion parameters.

Master Key and Detail Search Items

The master key items and detail search items (also called critical items) that define a path between two data sets must have identical type designators and simple sub-item lengths when they are defined in the item part. The sub-item lengths must be simple because the key and search items cannot be compound items. Because the same data item name can appear in more than one data set, you can use the same data item name and definition for both the master key items and detail search items. For example, the data item ACCOUNT is used as the key item in the CUSTOMER master and as the search item in the SALES detail data sets. If you want to make a distinction between the search items, they can be defined separately. An example of this technique is found in the ORDERS database. The key 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. However, the item names, read class lists, and write class lists differ. Figure 3-5. at the end of this chapter contains the listing printed by the Schema Processor when the ORDERS database schema is processed. Refer to this figure for examples of the schema parts.

Data Set Identifiers

Similar to data items, data sets can 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 in order to maintain program file independence.

Schema Processor Operation

The Schema Processor is a program that 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 as either an MPE/iX job or session. For further information about sessions and jobs, refer to the *MPE/iX Commands Reference Manual*. In either case, you must use the following MPE/iX 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 that the Schema Processor uses for the text file and list file. 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
textfile	Schema and Schema Processor commands	DBSTEXT	$STDINX
listfile	Output listing	DBSLIST	$STDLIST

If you want to equate these files to some other actual file designator, you can use the MPE/iX 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.

NOTE Parm equals 1 or 3 is recommended for large schema files.

Table 3-6. shows sample combinations of MPE/iX RUN and FILE commands that can be used to initiate DBSCHEMA execution.

Table 3-6. Examples of RUN and FILE Commands

`: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 disk text file named ORDERSSC. Outputs listing to $STDLIST.*
`:FILE DBSLIST;DEV=LP` `:RUN DBSCHEMA.PUB.SYS;PARM=2`	*Prompts for lines of schema and outputs the listing to a line printer.*
`:FILE DBSTEXT=ORDERSSC` `:FILE DBSLIST=ORDERLST` `:RUN DBSCHEMA.PUB.SYS;PARM=3`	*Processes schema from user text file named ORDERSSC; outputs the listing to a disk text file named ORDERLST.*

Only the first 72 characters of each text file record are processed.

If the schema is error-free, a root file is created, given the same name as the one specified for the database in the schema, initialized, and saved as a catalogued disk file. To process the schema without creating a root file, use the NOROOT option of the `$CONTROL` command. For more information, refer to "Schema Processor Commands" later in this chapter.

Creating the Text File

A convenient method for creating the input file is to use a text editor, for example, EDIT/3000, to enter the commands and schema in a disk file.

Figure 3-2. illustrates this process in a sample session that also executes the Schema Processor. The example shown in Figure 3-2. uses EDIT/3000.

The steps to follow are listed below and correspond to the numbers in Figure 3-2.

1. Initiate an MPE/iX 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 disk file named ORDERSSC. Then terminate the Editor.
5. Use the MPE/iX `FILE` command to equate the formal file designator DBSLIST to the line printer and DBSTEXT to the disk file ORDERSSC.
6. Initiate execution of DBSCHEMA and indicate that the text file and list file 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.

The Schema Processor can also be executed in batch mode. See "Schema Processor Commands" later in this chapter.

Figure 3-2. Sample Schema Creation Session

```
1    :HELLO USER.ACCOUNT
     HP3000  RELEASE:  B.30.00   ...   TUE, SEP 10, 1991, 1:49 PM
     MPE/iX  HP31900 B.08.14  Copyright (C) Hewlett-Packard 1987.  ...
2    :EDITOR
     HP32201A.00.00   EDIT/3000   TUE, SEP 10, 1991,  2:07 PM
     (C) HEWLETT-PACKARD CO.  1985
3    /ADD
     1  $PAGE "SCHEMA OF DATA BASE ORDERS"
     2  $CONTROL ERRORS=5, BLOCKMAX=256
     3  BEGIN DATABASE ORDERS;
     .
     .
     .
     59 END.
     60 //
     ...
4    KEEP ORDERSSC
     /EXIT
5    :FILE DBSLIST;DEV=LP
     :FILE DBSTEXT=ORDERSSC
6    :RUN DBSCHEMA.PUB.SYS;PARM=3
     HP30391C.03.02
     NUMBER OF ERROR MESSAGES: 0
     ROOT FILE ORDERS CREATED
     END OF PROGRAM
     :BYE
```

The Database Creator

The person who creates the root file is identified as the **database creator** and can subsequently create and initialize the database. To do so, the database creator must log on with the same account, user name, and group used to create the root file and execute the TurboIMAGE/XL utility program DBUTIL. This program is described in chapter 8.

Schema Processor Commands

TurboIMAGE/XL provides several commands that you can use anywhere in the schema to specify options available while processing the schema. The commands are: $PAGE, $TITLE, and $CONTROL. The dollar sign ($) 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 can be freely inserted between items in the parameter list.

Command records cannot contain comments.

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 disk file and executed from a terminal.

Figure 3-3. Schema Processor Batch Job Stream

```
!JOB USER.ACCOUNT      ← Job command
!RUN DBSCHEMA.PUB.SYS  ← Run command
$PAGE                  ← Schema Processor commands (optional)
$TITLE
$CONTROL
BEGIN DATABASE B;      ← Schema
  .
  .
  .
!EOJ                   ← EOJ command
```

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 list file to eject to the top of the next page, print character-strings that you can optionally specify, and skip two more lines before continuing the listing.

Syntax

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

Parameters

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

Example

```
$PAGE  "ORDERS DATABASE SCHEMA", "VERSION 3"

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

$PAGE
```

Description

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

Example

```
$TITLE"""PRELIM""ORDERS DATABASE"

$TITLE "ORDERS DATABASE SCHEMA  JULY, 1991"
```

Description

The $TITLE command can 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=nnnn][,ROOT
                                             ,NOROOT]
    [,BLOCKMAX=nnnn] [,TABLE
                      ,NOTABLE] [,JUMBO
                                 ,NOJUMBO]
    [, ODDPALLOWED]
```

Parameters

LIST	causes each source record of the schema to be printed on the list file.
NOLIST	specifies that only source records with errors be printed on the list file. An error message is printed after these records.
ERRORS=*nnn*	sets the maximum number of errors to *nnn*. If more than *nnn* errors are detected, the Schema Processor terminates. *nnn* can have a value between 0 and 999, inclusive. The default value is 100.
LINES=*nnnnn*	sets the number of lines per page on the list file to *nnnnn* which can be between 4 and 32767, inclusive. The default value is 60 if the list file 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. This is the default.
NOROOT	prevents the Schema Processor from creating a root file.
BLOCKMAX=*nnnn*	sets the maximum physical block length (in halfwords) for any data set in the database. *nnnn* can have a value between 128 and 2560, inclusive. The default value is 512. This is an important parameter and is discussed in detail in the section "Selecting the Block Size."
TABLE	causes the Schema Processor to write a table of summary information about the data sets to the list file device if no errors are detected. This is the default.
NOTABLE	suppresses the TABLE option.
JUMBO	allows a data set, defined following this option whose capacity is greater than 4 GB, to automatically become a jumbo data set.
NOJUMBO	disallows data sets, defined following this option, to be jumbo data sets.
ODDPALLOWED	DBSCHEMA will return an error if the sub-item length of the P data type is an odd number. Use this option to bypass the checking. However, the product of the sub-item length and sub-item count still needs to be evenly divisible by 4.

Description

The default parameters are *LIST, ROOT, TABLE,* and *NOJUMBO*. 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 can be placed in any order but must be separated by commas.

To specify a jumbo data set, the JUMBO option must be included in the schema before defining any jumbo data sets. Then any data set whose capacity is greater than 4 GB automatically becomes a Jumbo data set. If the JUMBO option is not specified, an error is generated for the data sets exceeding the 4 GB limit. Use the NOJUMBO option to turn off the jumbo feature.

DBSTORE does not store jumbo data sets; instead use TurboSTORE/iX 7x24 True-Online Backup using ONLINE=START or ONLINE=END option.

NOTE Use only third-party utilities and diagnostic tools that are enhanced to handle Jumbo data sets.

Selecting the Block Size

The data set records are transferred from the disk to memory in 4096-byte pages in TurboIMAGE/XL blocks. The block format is described in chapter 10. When you specify a maximum block size with the $CONTROL command, you should consider:

- Efficient disk space utilization.
- Localizing the block's bit map to the actual data entries within a 4096-byte page.

The Schema Processor determines the number of data records that fit in a block. Note that DBSCHEMA chooses a block size (less than or equal to the maximum block size) that makes the best use of disk space, and which can be substantially less than the maximum block size specified by $CONTROL BLOCKMAX (or the default of 512 halfwords). If the record size is greater than 512 halfwords, BLOCKMAX must be set greater than or equal to the record size. A certain amount of tuning may be necessary to determine the best block size. In general, the default block size of 512 halfwords yields reasonable performance on TurboIMAGE/XL and should be changed only when needed.

Schema Processor Output

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

```
PAGE 1      HEWLETT-PACKARD 30391C.05.00  TurboIMAGE/3000: DBSCHEMA
        MON, JAN 10, 1994, 3:32 PM  (C) HEWLETT-PACKARD CO. 1987
```

If your standard output device ($STDLIST) is different from the list file, an abbreviated product identification is also printed on $STDLIST. Subsequent pages of the list file are headed by a page number, the database 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 list file. However, if the text file and list file 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 right angle bracket (>).

Summary Information

After the entire schema has been scanned, several types of summary information can be printed on the list file.

- If some of the items defined in the item part are not referenced in the set part, and if no errors are encountered, the message UNREFERENCED ITEMS:*list of items* is printed to the list file. 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 control blocks 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. Figure 3-4. describes the information contained in the summary. The NOTABLE parameter of the $CONTROL command suppresses printing of this table.

Figure 3-4. Data Set Summary Table

```
 DATA SET     TYPE    FLD  PT  ENTR   MED  MAXIMUM   BLK  BLK   DISC
   NAME               CNT  CT  LGTH   REC  CAPACITY  FAC  LGTH  SPACE


EMPLOYEE         Mi  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   20048     28   506   1436
             INITIAL CAPACITY: 10024   INCREMENT ENTRIES: 2016


                        TOTAL DISC SECTORS INCLUDING ROOT:  1532
```

Table 3-7. Data Set Summary Table Information

Headings	Description
DATA SET NAME	The name of the data set.
TYPE	A for automatic, M for manual, or D for detail. A small letter "i" adjacent to the "M" or "A" indicates the data set has a B-Tree index on the key item.
FLD CNT	The number of data items (fields) in each entry of the data set.
PT CT	Path count. For a master data set, this is the number of paths specified for the data set key item. For a detail data set, it is the number of search items defined for the data set.
ENTR LNGTH	The length in halfwords of the data portion of the data entry (not including any of the TurboIMAGE/XL pointers or structure information associated with a data entry).
MED REC	The total length in halfwords of a media record of the data set. This length includes the entry length plus any of the TurboIMAGE/XL pointers associated with the data entry. Media records are discussed in chapter 10.
MAXIMUM CAPACITY	The maximum number of entries allowed in the data set. For master data sets specified for dynamic expansion and for detail data sets, this number can differ from the number of entries specified in the schema itself, because the capacity is adjusted to represent an even multiple of the blocking factor (see below).
BLK FAC	The number of media records that are blocked together for transfer to and from the disk.
BLK LGTH	The total length in halfwords of the physical block as defined in BLK FAC. This includes the media records and a bit map. Bit maps are discussed in chapter 10.
DISC SPACE	The amount of disk space occupied by the MPE/iX file containing the data set.
INITIAL CAPACITY	The number of entries for which space will be allocated or initialized when the data set is created.
INCREMENT ENTRIES	The number of entries by which the data set will be expanded each time its space is exhausted.
TOTAL DISC SECTORS INCLUDING ROOT: *nnnn*	The total number of disk sectors that will be occupied by the database when created using the DBUTIL program.

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

```
NUMBER OF ERROR MESSAGES:  0
ITEM NAME COUNT:  23     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 list file.

- 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 halfwords of the body of the root file. BUFFER LENGTH is the length in halfwords of each of the data buffers (unused by TurboIMAGE/XL but provided for compatibility with TurboIMAGE on MPE V; refer to the discussion on buffer specifications in appendix H). TRAILER LENGTH is the length in halfwords of an area in the control block used by TurboIMAGE/XL 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 list file:

  ```
  ROOT FILE database name CREATED
  ```

  where *database name* is the name given in the BEGIN DATABASE statement in the schema.

- The initial capacity and increment entries are optional parameters. If dynamic expansion is required, include these parameters when defining the detail data set (See chapter 3.) For existing databases, use DBChange Plus or other third-party tools you may be using to specify the expansion parameters.

Schema Errors

When the Schema Processor detects an error, it prints a message to the list file. 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, followed by 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 can 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 list file and to $STDLIST if it is not the same as the list file:

```
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 list file output printed when the schema of the sample ORDERS database is processed. The database has 5 passwords and contains 23 data item definitions and 6 data set definitions. The Schema Processor summary information is printed following the schema.

Figure 3-5. ORDERS Database Schema ORDERS Database Schema

```
PAGE 1    HEWLETT-PACKARD 30391C.05.00 TurboIMAGE/3000: DBSCHEMA
       MON, JAN 10, 1994, 3:32 PM  (C) HEWLETT-PACKARD CO. 1987

    $CONTROL LIST,LINES=46
    $PAGE "SCHEMA FOR DATABASE ORDERS"

BEGIN DATABASE ORDERS;

PASSWORDS:

 11 CREDIT;      << CUSTOMER CREDIT OFFICE >>
 12 BUYER;       << BUYER - RESPONSIBLE FOR PARTS INVENTORY >>
 13 SHIP-REC;    << WAREHOUSE - SHIPPING AND RECEIVING >>
 14 CLERK;       << SALES CLERK >>
 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-ADDRESS, X26 (12,13,14/11); << NUMBER AND STREET ADDRESS>>
 SUPPLIER,       X16 (12,13/);      << SUPPLYING COMPANY NAME>>
 TAX,            J2 (14/);          << SALES TAX (PENNIES)>>
 TOTAL,          J2 (11,14/);       << TOTAL AMOUNT OF SALE(PENNIES)>>
 UNIT-COST,      P8 (/12);          << UNIT COST OF PRODUCT>>
 ZIP,            X6 (12,13,14/11);  << ZIP CODE>>

SETS:
 NAME:       DATE-MASTER,AUTOMATIC,DISC1;      <<DATE MASTER>>
 ENTRY:      DATE(3);
 CAPACITY:   365;
```

```
 NAME:       CUSTOMER,MANUAL /INDEXED (14/11,18),DISC1;  <<CUSTOMER MASTER>>
 ENTRY:      ACCOUNT(1),
             LAST-NAME,
             FIRST-NAME,
             INITIAL,
             STREET-ADDRESS,
             CITY,
             STATE,
             ZIP,
             CREDIT-RATING;
 CAPACITY:   201;

PAGE 2  SCHEMA FOR DATABASE ORDERS

 NAME:       PRODUCT,MANUAL(13,14/12,18),DISC1;<<PRODUCT MASTER>>
 ENTRY:      STOCK#(2),
             DESCRIPTION;
 CAPACITY:   300;


 NAME:       SUP-MASTER,MANUAL(13/12,18),DISC1; <<SUPPLIER MASTER>>
 ENTRY:      SUPPLIER(1),
             STREET-ADDRESS,
             CITY,
             STATE,
             ZIP;
 CAPACITY:   201;


 NAME:       INVENTORY,DETAIL(12,14/13,18),DISC2; <<INVENTORY DETAIL>>
 ENTRY:      STOCK#(PRODUCT),
             ONHANDQTY,

             SUPPLIER(!SUP-MASTER),        <<PRIMARY PATH>>
             UNIT-COST,
             LASTSHIPDATE(DATE-MASTER),
             BINNUM;
 CAPACITY:   1800,450,10%;


 NAME:       SALES,DETAIL(11/14,18),DISC2; <<SALES DETAIL>>
 ENTRY:      ACCOUNT(CUSTOMER(PURCH-DATE)),
             STOCK#(PRODUCT),
             QUANTITY,
             PRICE,
             TAX,
             TOTAL,
             PURCH-DATE(DATE-MASTER),
             DELIV-DATE(DATE-MASTER);
 CAPACITY:   1008,504,112;

END.
```

```
 DATA SET      TYPE   FLD  PT  ENTR  MED  MAXIMUM   BLK  BLK  DISC
   NAME              CNT  CT  LGTH  REC  CAPACITY  FAC  LGTH SPACE

DATE-MASTER       A   1    3   3     26   365       19   496  96
CUSTOMER          Mi  9    1   41    52   201       7    365  96
PRODUCT           M   2    2   14    31   300       16   497  80
SUP-MASTER        M   5    1   31    42   201       12   505  80
INVENTORY         D   6    3   20    32   1800      15   481  128
               INITIAL CAPACITY: 450     INCREMENT ENTRIES: 45
SALES             D   8    4   19    35   1008      14   491  160
               INITIAL CAPACITY: 504     INCREMENT ENTRIES: 112

                         TOTAL DISC SECTORS INCLUDING ROOT:  672

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.
```

4 Using the Database

After you design the database, create the root file, and build the data sets, you can write application programs to enter and use the data. Programs written in BBASIC, C, COBOL II, FORTRAN 77, or Pascal gain access to the database through calls to TurboIMAGE/XL procedures. RPG programs contain specifications used by the Report Program Generator to make calls to the TurboIMAGE/XL procedures for you. This chapter contains a text discussion of the procedures used to open the database and to enter, read, update, and delete data; it also contains information on locking, transaction logging, checking procedure status, and interpreting errors. Use this chapter with chapter 5 which gives details about each procedure call, its parameters, and status information.

NOTE Before application programs can be executed, the database must be created using the TurboIMAGE/XL utility program DBUTIL described in chapter 8.

Opening the Database

Before you can gain access to the data, the process you are running must open the database 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/iX Intrinsics Reference Manual*. In opening a database, DBOPEN establishes an access path between the database and your program by doing the following:

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

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

DBOPEN also determines if the operating system supports the native language as defined in the root file. The following error message is returned if the language attribute of the database is not supported by the current system configuration:

```
Language is not supported
```

Refer to Chapter 6 for more information on Host Language Access and appendix A for more information on error messages.

Database Control Blocks

TurboIMAGE/XL executes using data stored in different types of control blocks stored in privileged mapped files or virtual data objects: the Database System Control Block (DBS), the Database Globals Control Block (DBG), the Database Buffer Area Control Block (DBB), the Database User Local Control Block (DBU), the Remote Database Control Block (DBR), Database User Local Index Control Block (DBUX), the Lock Table (TURBOLKT), Global Dynamic Multi-database Transaction Table (TURBOGTX), DBQUIESCE and DBOPEN Table (QOPEN), and DBQUIESCE Lock Table (QLOCK). These are described below:

- The Database System Control Block (DBS) is created by DBOPEN if it does not already exist. The DBS is used as a system-wide table to locate the current Database Globals Control Block (DBG) for any opened database. Each system has only one DBS, created as a permanent file called TURBODBS in the PUB group and the SYS account on that system.
- The Database Globals Control Block (DBG) is created for a particular database when the first user's process calls the DBOPEN procedure to open the database. The DBG contains global information required by TurboIMAGE/XL intrinsics during run-time, including a pointer to the Database Buffer Area Control Block (DBB). Each open database has exactly one DBG regardless of the number of concurrent access paths to the database. All TurboIMAGE/XL procedures on a particular database (except DBERROR and DBEXPLAIN) reference the DBG. In addition, the DBG contains the lock table which holds user-level locking information. The DBG is purged when the last user's process closes the database (DBCLOSE).
- The Database Buffer Area Control Block (DBB) is created for a particular database when the first user's process calls the DBOPEN procedure to open the database. The DBB contains a set of buffer headers which point to data in memory from any of the data sets, and contains a pointer to the DBG. 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 disk. The DBB is purged when the last user closes the database (DBCLOSE).
- One Database User Local Control Block (DBU) is created each time a user's process successfully calls DBOPEN. Each DBU contains information about the user's individual access to the database and contains pointers to the DBS, DBG, and DBB. The privileged mapped file containing the DBU is associated with this DBOPEN. The DBU is purged when the corresponding DBCLOSE closes the database. A process can open a maximum of 127 databases (or one database 63 times), depending upon the system resources; therefore, a maximum of 127 DBUs can be created. It is recommended that a process close a database once it is no longer needed for that process.
- One Remote Database Control Block (DBR) is created on the local system each time a user's process successfully opens a remote database. The DBR contains database set and item information as well as the work areas necessary to set up communications to the remote computer.
- The Database User Local Index Control Block (DBUX) is created the first time the user's process calls DBOPEN. One DBUX exists for each user's process. Its purpose is to keep track of the addresses of all the DBUs and/or DBRs for that process. Because a maximum of 127 entries are allowed in the DBUX, each process is allowed 127 DBOPENs

(63 per database) depending on the availability of system resources. The DBUX remains allocated until the user's process is terminated.

- The TurboLock Table is a permanent file, TURBOLKT.PUB.SYS, that is created by DBOPEN (if it does not exist beforehand). Thereafter, it is opened when the first user opens any database on the system. It is purged when the system is rebooted. Each system has only one TURBOLKT file. It is used to avoid deadlocks for all IMAGE/SQL users. Additionally, it is also used to detect potential deadlock for TurboIMAGE/XL users if, and only if, deadlock detection is activated by DBCONTROL mode 7.
- The Global Dynamic Multi-database Transaction Table is a permanent file, TURBOGTX.PUB.SYS, that is created by DBXBEGIN (if it does not exist beforehand). Thereafter, it is opened for all users who employ dynamic multi-database transaction(s) (DMDBX). Each system has only one TURBOGTX file, and it remains on the system even after the system is rebooted. It is used for tracking DMDBX.
- The DBQUIESCE and DBOPEN Table, QOPEN, is an unnamed global object and is first created by DBOPEN when the first writer of any database enabled for user logging opens a database. It is subsequently accessed only by writers of databases enabled for user logging. Each system has only one QOPEN table, and it is purged when all processes accessing it are terminated. It contains information pertaining to DBOPEN and user logging process. This information is used to write to log records for DBRECOV and to coordinate with DBQUIESCE called by TurboSTORE/iX 7x24 True-Online Backup.
- The DBQUIESCE LOCK Table, QLOCK, is an unnamed permanent global structure that is created by DBOPEN (if it does not exist beforehand). There is one per system, and it is accessed by all writers to all databases. It is purged only at system reboot time. It is used for containing database information required to quiesce database(s) for TurboSTORE/iX 7x24 True-Online Backup.

All TurboIMAGE/XL intrinsics process on the DBU except accesses for global and buffer area information found in the two global blocks (DBG and DBB).

Passwords

When you open the database 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 database creator and supply a semicolon as a password, you are assigned user class 64, which grants you unlimited database access privileges. Passwords and user classes are discussed in chapter 2.

Database Access Modes

There are eight different access modes for opening the database with the DBOPEN procedure. Each mode determines the type of operation that you can perform on the database, as well as the types of operations other users can perform concurrently. To simplify the definition of the various DBOPEN modes, the following terminology is used:

- **Read access** modes (5, 6, 7, and 8) allow the user to locate and read data entries.
- **Update access** mode (2) allows read access and permits the user to replace values in all data items except master data set key items and detail data set search and sort items. The critical item update (CIUPDATE) option, which can permit the values of

detail data set search and sort items to be updated and which is discussed later in this chapter, is not available in this mode.

- **Modify access** modes (1, 3, and 4) allow updates and permit the user to add and delete entries. For access modes 1, 3, and 4 only, users can update the values of detail data set search and sort items if the critical item update (CIUPDATE) option settings for the database and the current process permit them to do so. CIUPDATE is discussed later in this chapter.

The TurboIMAGE/XL library procedures (also called intrinsics) that can be used with each type of DBOPEN mode are as follows:

Table 4-1. Library Procedures DBOPEN Modes

DBOPEN Mode	Library Procedures Available
Read	DBFIND and DBGET
Update	DBFIND, DBGET, and DBUPDATE
Modify	DBFIND, DBGET, DBUPDATE, DBPUT, and DBDELETE

Table 4-2. summarizes the type of database access granted in each access mode, provided the MPE/iX security provisions and your password permit it. Access modes 3 and 7 provide exclusive access to the database; all other modes allow shared access.

Table 4-2. Database Access Mode Summary

Access Mode	Type of Access Mode Granted	Concurrent Access Modes Allowed		Special Requirements
1	Modify	1, 5	Modify (with locking)	Locking must be used for update or modify.
2	Update	2, 6	Update	
3	Modify		None	Exclusive Access
4	Modify	6	Read	
5	Read	1, 5	Modify (with locking)	TurboIMAGE/XL does not require locking, but it should be used to coordinate access with users who are modifying the database.
6	Read	2, 4, 6, 8	Modify	
7	Read		None	Exclusive Access
8	Read	6, 8	Read	

Concurrent Database Access Modes

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

Here is a summary of the acceptable environments:

- Multiple access mode 1 and access mode 5 users
- Multiple access mode 6 and access mode 2 users
- Multiple access mode 6 users and one access mode 4 user
- Multiple access mode 6 and access mode 8 users
- One access mode 3 user
- One access mode 7 user

Subsets of these environments are also allowed. For example, all users can be access mode 5, 6, or 8 users; or there could be one access mode 1 user; and so on.

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

Database Operations

This section explains in detail what occurs when a database 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 can be used. Refer to the discussion of the locking facility later in this chapter for more information.

- **Access Mode 1.** The database is opened for shared modify access. Opening in mode 1 succeeds only if all other current users of the database are using access modes 1 or 5.

  All TurboIMAGE/XL procedures are available in this mode. The critical item update (CIUPDATE) option, which can permit you to update the values of detail data set search and sort items, is available in this mode. 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 database 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/XL library procedures, DBLOCK and DBUNLOCK. The locking requirements can be met by locking the affected entries, the sets containing the entries, or the whole database.

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

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

- **Access Mode 2.** The database is opened for shared update access. The DBOPEN call succeeds only if all current users of the database are using access modes 2 and 6. All TurboIMAGE/XL procedures are available to the access mode 2 user except DBPUT and DBDELETE which are disallowed in this mode. The critical item update (CIUPDATE)

option, which can permit you to update the values of detail data set search and sort items, is *not* available in this mode. Therefore, the access mode 2 user is able to read all data entries and update some 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 access mode 2 users are simultaneously updating data entries. In many applications, it may be possible to arrange for each user's process to update unique data entries or data items so that the database 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 database will reflect only the latest values. Locking can be used, if desired, to coordinate update sequences to an entry or to coordinate with access mode 6 readers.

- **Access Mode 3.** The database is opened for exclusive modify access. If any other users are accessing the database, it cannot be opened in this mode. All TurboIMAGE/XL procedures are available to the access mode 3 user. The critical item update (`CIUPDATE`) option, which can permit you to update the values of detail data set search and sort items, is available in this mode. No other concurrent process is permitted to gain any type of access to the database.
- **Access Mode 4.** The database is opened for semi-exclusive modify access. Only one access mode 4 user can access the database, and all other current users must be in access mode 6 (read only). The access mode 4 user is permitted to call any TurboIMAGE/XL procedure and has complete control over database content. The critical item update (`CIUPDATE`) option, which can permit you to update the values of detail data set search and sort items, is available in this mode. Other read-only users are permitted concurrent access to the database in mode 4, but not in mode 3. Locking can be used to coordinate with access mode 6 readers.
- **Access Mode 5.** The database is opened for shared read access. All other concurrent users must be in access mode 1 or 5. Access mode 5 operates the same as access mode 1, except that the `DBUPDATE`, `DBPUT`, and `DBDELETE` procedures, which alter the database, are disabled for the access mode 5 user. Locking can be used, if desired, to ensure that data is not being modified while you are reading it.

  Access mode 5 is appropriate for inquiry-type applications if they can tolerate the possibility of database modifications taking place simultaneously with access mode 1 users.
- **Access Mode 6.** The database is opened for shared read access. Concurrent users must be in access mode 2, 4, 6, or 8. Access mode 6 can also be used while the database is being stored with the TurboIMAGE/XL utility program `DBSTORE`. Some of these modes are incompatible with each other as shown in the preceding discussion of concurrent access modes. All TurboIMAGE/XL procedures that alter the database are disabled. Locking can be used to synchronize with users who are concurrently updating.

  Access mode 6 is appropriate for inquiry-type applications if they can tolerate the possibility of database modifications taking place simultaneously with access mode 2 and 4 users.

- **Access Mode 7.** The database is opened for exclusive read access. No other users can access the database concurrently. Access mode 7 operates the same as access mode 3, except that the DBUPDATE, DBPUT, and DBDELETE procedures, which alter the database, are disabled for the access mode 7 user.
- **Access Mode 8.** The database is opened for shared read access. Concurrent users must either be in access mode 6 or 8, or using the TurboIMAGE/XL utility, DBSTORE. TurboIMAGE/XL procedures that alter the database are not permitted. Because access mode 8 allows only concurrent readers, a user program with this access mode can be assured that the database values it reads are unchanging.

Selecting a Database Access Mode

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

- Use the minimum capability required to accomplish the task. For example, select a read-only access mode (5, 6, 7, or 8) if the program does not alter the database in any way. Read access modes allow concurrent database STORE operations and do not set the MPE/iX "file modified" flag.
- Allow concurrent users to have as much capability as is required for successful completion of the task. If the task is merely browsing through the database, producing a quick report, or accessing an unchanging portion of the database, choose an access mode that allows concurrent users to make database modifications to other parts of the database. Allowing concurrent read-only access (modes 2, 4, and 8) can be appropriate in many situations. For programs that must be assured of no concurrent structural changes, but can tolerate simultaneous updates to entries, mode 2 is suitable. Locking can be used to control simultaneous updates to a data entry. If it is necessary to make additions or deletions to a database from concurrent multiple processes, modes 1 and 5 must be used. Fully exclusive operation (modes 3 and 7) are available if needed.

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

- Programs that perform operations, which include adding and deleting entries or which need to update detail data set search and sort items via the critical item update (CIUPDATE) option, should open with mode 1, 3, or 4. Consider the following when choosing among access modes 1, 3, and 4:

  Access Mode 1 Can be used if other processes need to add and delete entries simultaneously. In this case, the affected parts of the database must be locked while performing updates, additions, or deletions.

  Access Mode 3 Can be used if the program must have exclusive access to the database.

  Access Mode 4 Can be used if exclusive ability to change the database is required but access mode 6 processes need to be able to read the database while changes are being made.

- 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 database in access mode 2. Consider the following when choosing access mode 2:

Access Mode 2 Can be used if processes are allowed to update the database concurrently. Locking should be used to coordinate updates. The critical item update (CIUPDATE) option, which can permit you to update the values of detail data set search and sort items, is *not* available in this mode.

- Programs that only locate and read or report on information in the database should open with one of the read-only access modes. In this case, the access mode selected depends upon either the type of process running concurrently or the need for an unchanging database while the program is running. Consider the following when choosing among access modes 5, 6, 7, and 8:

  Access Mode 5 Can be used if concurrent processes will operate in access mode 1 or 5. Parts or all of the database should be locked to prevent concurrent changes during one or more read operations. Because concurrent access mode 1 processes are allowed, programs performing chained reads should lock the chain (see the discussion of DBGET in chapter 5).

  Access Mode 6 Can be used if it is not important what other processes are doing to the database. In this case, access mode 2 processes can update entries; one access mode 4 user can update, add, or delete entries; or access mode 6 or 8 users can read entries while the program is using the database.

  Access Mode 7 Can be used if the program must have exclusive read access to the database.

  Access Mode 8 Can be used if other processes are allowed to read but not modify the database. In this case, access mode 6 and 8 users can read entries while the program is using the database.

Locking within a Database Process

Refer to the locking discussion later in this chapter for considerations when locking and unlocking transactions within a database process.

User Transaction Logging

Users opening the database in access modes 1 through 4 use the MPE/iX user logging facility if the database administrator has enabled the database for logging (a procedure described in chapter 7). In this case, calls to the TurboIMAGE/XL intrinsics listed in Table 4-3. are automatically logged to a log file. Note that nothing is logged for programs opening the database with read-only access modes (5 through 8), regardless if the database was enabled for logging. The logging facility is described more fully later in this chapter and in chapter 7.

Table 4-3. Logged Intrinsics

DBBEGIN	DBCLOSE	DBDELETE	DBEND
DBMEMO	DBOPEN	DBPUT	DBUPDATE
DBXBEGIN	DBXEND	DBXUNDO	

The DBBEGIN and DBEND intrinsics are used to designate or block logical, static transactions for logging and recovery purposes. The DBXBEGIN and DBXEND intrinsics are used to mark logical, dynamic transactions spanning one database, or multiple databases up to fifteen, for dynamic roll-back recovery. Refer to Table 4-5. later in this chapter for a definition of TurboIMAGE/XL transaction types.

In addition to the above intrinsics, log records for DBQUIESCE procedures called by True-Online Backup to quiesce and unquiesce the database can also be logged to a log file.

Dynamic roll-back uses the MPE/iX Transaction Management (XM) facility to roll back transactions online while other database activity is occurring. User logging is not required for this type of recovery, but is recommended to guard against a hard disk failure. See chapter 7 for a discussion of logging and recovery methods.

Entering Data in the Database

Data is added to the database, one entry at a time, using the DBPUT procedure. You can 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

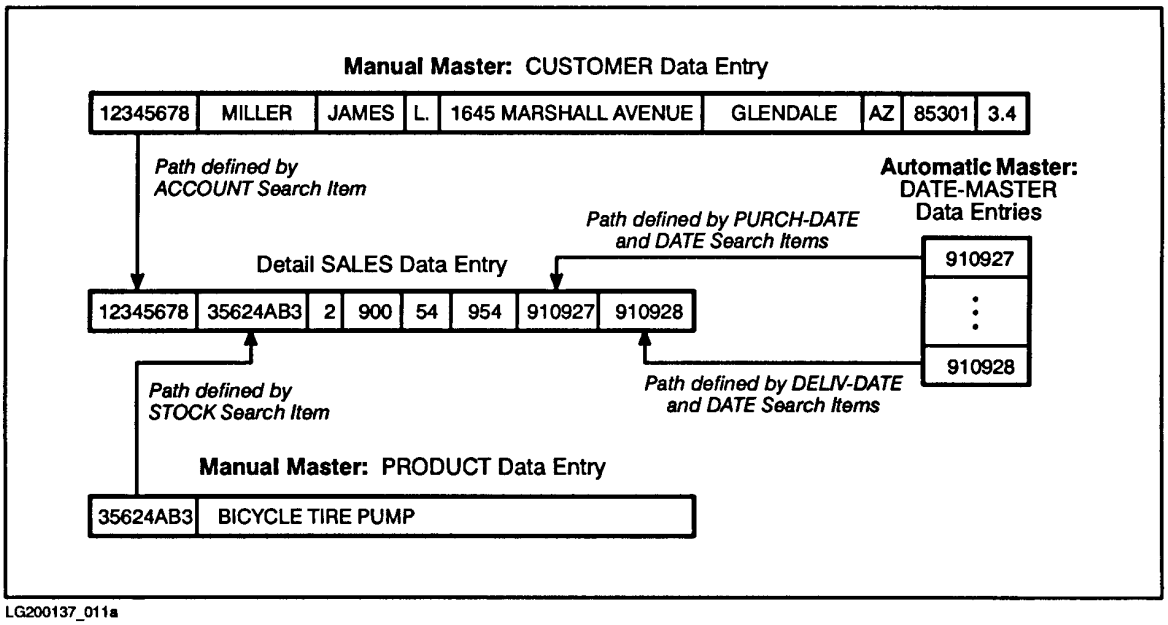
Before you can add an entry to a detail data set indexed by a manual master data set, the manual master must contain an entry with a key item value equal to the search item value you intend to put in the detail. If more than one manual master is used to index the detail, entries that have a key item value identical to the detail search item value for the same path must exist in each master. To illustrate, consider the ORDERS database 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 because 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/XL automatically adds entries to the DATE-MASTER with a DATE item value of 910927 and 910928 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.

Figure 4-1. Sample Data Entries from ORDERS Database



Coordinating Additions to a Database

A single DBPUT call involves chain maintenance and other activity that can span multiple data sets and multiple blocks within data sets even if an application is only adding entries to a single data set. TurboIMAGE/XL coordinates calls to DBPUT to ensure that one database user's DBPUT activity does not interfere with calls made by other users accessing the same database. Each DBPUT request finishes processing before TurboIMAGE/XL moves on to the next one. DBPUT calls are serialized even if they access unrelated data sets.

TurboIMAGE/XL provides a data prefetching option to enhance DBPUT (and DBDELETE) processing. You can enable this option with the DBUTIL utility and are recommended to do so only when *all* of the following conditions are true:

- Multiple processes are accessing the database.
- The processes are adding data with the DBPUT intrinsic (or deleting data with the DBDELETE intrinsic).
- Adequate CPU and memory resources are available on your system.

For additional information, refer to the discussion of DBPUT in chapter 5 and the description of the DBUTIL >>ENABLE command in chapter 8.

Access Mode and User Class Number

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

The database 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 can be a data entry, data set, or database lock. For master data sets, this can be a data set or a database lock. Note that the locking mechanism accepts a request to lock a data entry that does not yet exist; therefore, you can lock a data entry before you add it.

Key and Search Items

TurboIMAGE/XL performs checks on the values of detail data set search items before adding an entry to a data set. When adding records to a manual master, TurboIMAGE/XL verifies that the master data set key item value is unique for the set (that is, no entry currently contains a key item with the same value). If the data set is a detail, TurboIMAGE/XL verifies that the value of each search item forming a path with a manual master has a matching key value in that master. It also checks to be sure that room is available 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 database, you specify which data set and which entry in that data set you want. If the user class number with which you opened the database grants you read access, you can 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 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 or more disk files depending on whether or not the set is jumbo, 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/XL maintains internal information indicating which records of a data set contain entries and which do not.

Current Path

TurboIMAGE/XL maintains a **current path** for each detail data set and for each database accessor (that is, each DBOPEN). 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/XL 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 chapter.

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

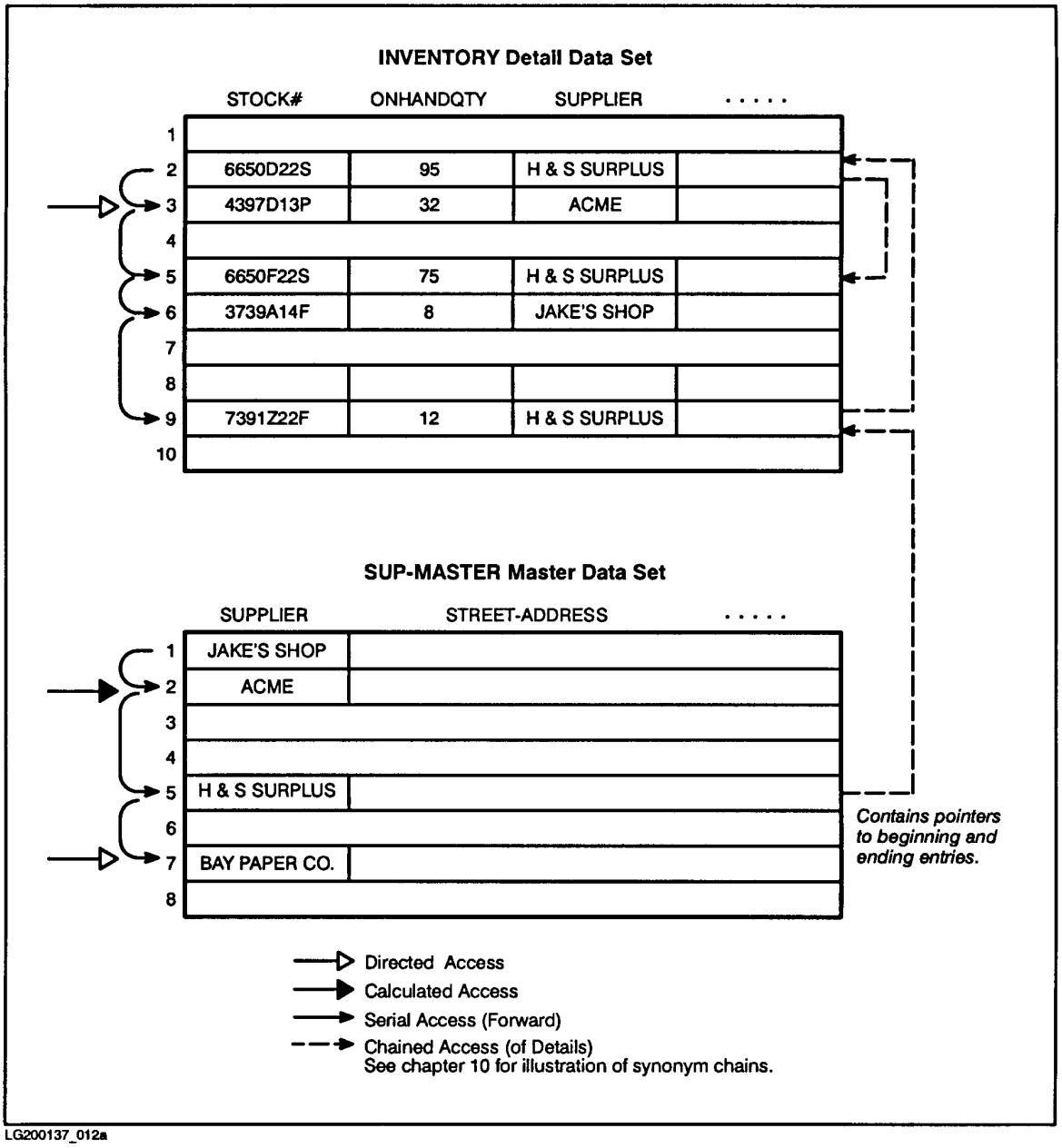
Reading Methods

The methods for requesting a data entry are categorized as follows:

- Directed access
- Serial access
- Calculated access
- Chained access
- Sorted sequential access

All of these methods are available through the TurboIMAGE/XL 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 database.

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



Directed Access

The directed access method of selecting the data entry to be read requires that you specify the record number of that entry. Record numbers are returned in status halfwords 3-4 by a successful call to the TurboIMAGE/XL intrinsic DBPUT (refer to the discussion of DBPUT in chapter 5). Other numbers that can be used for directed reads are the forward and backward pointers returned in status halfwords 7-8 and 9-10. To use these numbers for directed reads, you need to save them because subsequent TurboIMAGE/XL procedure calls can overwrite the status area.

During directed access the calling program specifies a record number or address. If any entry exists at this address, TurboIMAGE/XL 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, beginning-of-file, or no entry.

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 hollow 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.

NOTE When using directed access with master data sets, you should be aware of migrating secondaries. These are described in chapter 10.

Locking. If concurrent users are allowed to add to or delete from a 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.

Serial Access

In this mode of retrieval, TurboIMAGE/XL 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 can 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/XL returns an end-of-file if requested access is forward, or a beginning-of-file if it is backwards.

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

The entries connected by curved arrows 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 in a serial manner, copy it to a file, and sort it with routines external to TurboIMAGE/XL 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/iX file. Thus, you have the advantages of TurboIMAGE/XL database organization and the efficiency of serial access.

NOTE When using serial access with master data sets, you should be aware of migrating secondaries. These are described in chapter 10.

Locking. If concurrent users are allowed to modify the data set (access mode 1), you may wish to lock the data set or database before you begin the serial access sequence. Locking prevents 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 key item value. For example, the SUP-MASTER data entry for the supplier Acme shown in Figure 4-2. can be retrieved with this method because SUPPLIER is a key item in the SUP-MASTER data set. TurboIMAGE/XL locates the entry in the data set whose key item value matches the requested value. The exact technique used to perform calculated access is described in chapter 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 database.

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/XL determines which master set forms a path with the specified search item and locates the entry in that master data set whose key 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/XL uses a calculated read to locate the SUP-MASTER entry with a key 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 have been established for a detail data set, the calling

program can use the chained access method of retrieving data. You can 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 chapter 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 database. When a B-Tree index is created on the key item of the master set, chained access following a DBFIND can also be done on the master data set. More detailed information on B-trees is given in chapter 11.

Locking. If concurrent users are allowed to modify data entries in the chain you are currently accessing, you can 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 canceling the order by deleting data entries one by one, user A could retrieve an incomplete order. To prevent this from happening, a lock can 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 chapter.

Sorted Sequential Access

The sorted sequential access is used for B-Tree indices and can be used both for master and detail data sets. It is used for retrieving records in a sorted sequential order of the key item or search item value. For masters, each key item value is different and the records retrieved will be in a sorted sequential order of the key item value. The order can be ascending when DBGET mode 5 is used, and it is descending when DBGET mode 6 is used following a B-Tree DBFIND. For detail data sets, the B-Tree search can result into traversing multiple chains of the detail data set. Each chain has the same search item value. The subsequent chain will have a different search item value in a sorted sequential order. Therefore, even for detail data sets, the records retrieved will be in a sorted sequential order of the search item value. Note that within each such chain of the detail data set, the order of the records is chronological.

For example, when a B-Tree index is created on the key item SUPPLIER of SUP-MASTER, DBFIND can be used with SUP-MASTER in the *dset* parameter and the key item SUPPLIER in the *item* parameter. If the argument is equivalent to "Greater than BAY PAPER CO.", the order of records retrieved will be record number 5, the entry pertaining to the supplier H & S SURPLUS and record number 1 pertaining to JAKE'S SHOP in Figure 4-2. IF the DBFIND is for the detail data set, INVENTORY, and the same argument, the order of records retrieved will be 9, 2, 5, and 6 in Figure 4-2. The record numbers 9, 2, and 5 are pertinent to the supplier H & S SURPLUS (one chain) and 6 is pertinent to JAKE'S SHOP (second chain). The second chain has only 1 record. The chain head for H & S SURPLUS in SUP-MASTER contains record number 9 for beginning entry and 5 as ending entry for the path related to detail data set INVENTORY.

Rereading 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 reread the entry causes DBGET to return an exceptional condition indicating that the current record contains no entry. Refer to table 5-13 for more information on DBGET return status values.

Updating Data

TurboIMAGE/XL allows you to change the values of data items if the user class number with which you opened the database grants this capability to you. These items cannot be master data set key items or detail data set search or sort items, unless you take advantage of the critical item update (CIUPDATE) option. Depending on the CIUPDATE option settings for the database and the current process, you could change the values of detail data set search and sort items if the database access mode is 1, 3, or 4. This option is described later in this section.

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 can 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 database, 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 database must be opened in access mode 1, 2, 3, or 4. If it is opened in access mode 1, the data entry, data set, or database must be locked while the update is occurring. If the CIUPDATE option settings for the database and the current process permit updates of detail data set search and sort items, the database must be opened in access mode 1, 3, or 4 to take advantage of this option.

TurboIMAGE/XL 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/XL always completes the processing of one DBUPDATE request before it begins processing under another. However, data consistency problems can 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 such errors, a lock covering the data entry can be put in effect before it is read and released after it is updated.

TurboIMAGE/XL attempts to enforce this locking technique for users in database access 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.

NOTE To avoid locking around terminal reads, you may need to establish the lock, perform a mode 1 DBGET, check the value, update the entry, then remove the lock.

The password you use to open the database 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 both the read class list of the data set and in the write class list of the data item.

Updating Key, Search, and Sort Items

For compatibility and security reasons, by default you cannot use the DBUPDATE library procedure to update a master data set key item, or a detail data set search or sort item. However, in database access mode 1, 3, or 4, you can update search or sort items if permitted by the CIUPDATE option settings for the database and the current process; refer to the following discussion of CIUPDATE. If you do not or cannot take advantage of the CIUPDATE option and need to change search or sort items, or if you need to change master data set key items, you can first delete the selected entry with DBDELETE (see the section entitled "Deleting Data Entries"), and then add the entry back into the database with DBPUT. However, keep in mind that this method places the new record at the end of *each* unsorted chain, which disturbs the chronological order of each path.

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 could 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.

To facilitate updates of detail data set critical items, you should design any new databases to take advantage of the CIUPDATE option. Applications to be used with these databases can be written as described in the next section.

Critical Item Update

TurboIMAGE/XL provides an option called critical item update (CIUPDATE) which, depending on the settings for the database and the current process, can permit you to update the values of detail data set search and sort items. To take advantage of this option, you first need to set it through the TurboIMAGE/XL utility DBUTIL. Use the DBUTIL >>SET command, and set the CIUPDATE option equal to ON or ALLOWED.

NOTE The default setting is now ALLOWED. It was DISALLOWED in releases before C.07.00.

The ON setting permits any process to use critical item update on the database unless the process explicitly disables the option by calling the DBCONTROL procedure in mode 6; this call disables the option for the duration of the process or until a call to DBCONTROL in mode 5 enables the option. The ALLOWED setting requires that a process first call DBCONTROL in mode 5 to enable the CIUPDATE option for the duration of the process or until a call to

DBCONTROL in mode 6 disables the option. Using DBCONTROL modes 5 and 6 does not alter the permanent setting set with the DBUTIL >>SET command. The database must be opened in access mode 1, 3, or 4; and your user class must have write access at the set level, or both read access at the set level and write access at the item level.

Programmers should review the design of any of their applications that are used to update the values of data items to ensure that the data will be updated as expected. Applications that open the database in access mode 2 do not need to be modified, because only access modes 1, 3, and 4 can be used with CIUPDATE. *Applications that rely on TurboIMAGE/XL to restrict update of detail data set search and sort items can continue to do so as long as the database access mode is not 1, 3, or 4, or the CIUPDATE option is set as follows:*

- *DISALLOWED, or*
- *ALLOWED and the process does not call DBCONTROL in mode 5.*

Otherwise, these applications need to be modified to call the DBINFO procedure to check the CIUPDATE flags for the database and the current process, and (if need be) to disable CIUPDATE for the current DBOPEN via the DBCONTROL procedure in mode 6. Another method of restricting access to search and sort items is by granting read-only access at the data set level and limiting write access at the data item level.

Applications that allow the sort item for the current chain to be updated must allow the current entry to be moved within the chain. An entry which is moved can be encountered twice within one chained read.

To update a detail data set search or sort item, the following sequence of operations is recommended for your programs:

1. Call DBOPEN in access mode 1, 3, or 4.
2. Call DBLOCK to lock all data which must not be changed by other processes.
3. If this is a logical transaction, you need to declare the beginning of modifications by calling DBBEGIN or, if you are using dynamic roll-back, DBXBEGIN (see the discussion of logical transactions later in this chapter).
4. Read data using DBFIND and DBGET to determine what needs to be modified.
5. Call DBINFO to check the specific CIUPDATE option setting for the database and the setting for the current DBOPEN.
6. If the CIUPDATE setting is ON and has not been disabled by a DBCONTROL mode 6, go to the next step. If the setting is ALLOWED, call DBCONTROL mode 5 to enable the option. Note that if the CIUPDATE setting is ON and the option has not been disabled, you can still call a DBCONTROL mode 5 successfully.
7. If the CIUPDATE option is permitted, call DBUPDATE.
8. If the CIUPDATE option is not permitted, call DBDELETE and DBPUT.
9. Check the status code.
10. If the status code is not 0, call DBERROR and take appropriate action. For example, if this is a dynamic transaction, you must call DBXUNDO to roll back the transaction.

NOTE If this is a dynamic transaction, a call to DBXUNDO must be processed if an error occurs or if the transaction needs to be rolled back for any other reason. TurboIMAGE/XL will not go on to the next transaction in the event of a transaction abort unless an intervening call to DBXUNDO occurs.

11. If the status code is zero and this is a logical transaction, call DBEND or, alternatively, DBXEND to declare the end of the modifications. However, if this is a dynamic transaction and DBXUNDO was used to roll back the transaction, your program logic should ensure that the subsequent call to DBXEND is not processed.
12. Call DBUNLOCK to release all of the locks.

Deleting Data Entries

Data is deleted from the database using the DBDELETE procedure. You can delete entries from manual master and detail data sets. Entries are automatically deleted from automatic master data sets as explained below.

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/XL verifies that your password and associated user class number allow you to delete the current entry of the specified data set.

Sequence for Deleting Entries

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/XL automatically deletes the master entry.

If the data entry is in a manual master data set, TurboIMAGE/XL 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 because 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 key 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.

Coordinating Deletions from a Database

A single DBDELETE call involves chain maintenance and other activity that can span multiple data sets and multiple blocks within data sets even if an application is only deleting entries from a single data set. TurboIMAGE/XL coordinates calls to DBDELETE to ensure that one database user's DBDELETE activity does not interfere with calls made by other users accessing the same database. Each DBDELETE request finishes processing before TurboIMAGE/XL moves on to the next one. DBDELETE calls are serialized even if they access unrelated data sets.

TurboIMAGE/XL provides a data prefetching option to enhance DBDELETE (and DBPUT) processing. You can enable this option with the DBUTIL utility and are recommended to do so only when *all* of the following conditions are true:

- Multiple processes are accessing the database.
- The processes are deleting data with the DBDELETE intrinsic (or adding data with the DBPUT intrinsic).
- Adequate CPU and memory resources are available on your system.

For additional information, refer to the discussion of DBDELETE in chapter 5 and the description of the DBUTIL >>ENABLE command in chapter 8.

Access Modes and User Class Numbers

To update data items, the database 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 database 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, that is, 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 has write access to the data set. Write access to the data set means the database was opened in a write access mode and the user class number associated with the DBOPEN password is on the set write class list.

Using the Locking Facility

The DBLOCK procedure applies a logical lock to a database 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 database. With the DBLOCK procedure, application programs can isolate temporarily a subsection of the database in order to perform a transaction against the isolated data. Locking is not required to protect the structure of the database. TurboIMAGE/XL has internal mechanisms that do this.

If a program opens the database in access mode 1 and locks a part of the database, 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/XL does not allow changes in access mode 1 unless a lock covers the data to be changed. If one process has the database opened in access mode 1, TurboIMAGE/XL requires that all other processes that modify the database must also operate in access mode 1.

The DBLOCK procedure operates in one of six modes. Modes 1 and 2 can be used for locking the database and modes 3 and 4 for locking a data set. In modes 5 and 6, you describe the database 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 the customer has placed. The data entries for the customer's account that are in the SALES data set could be locked while the order is changed and other database activity can 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 data item 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 can be that none, one, or many entries are locked depending on how many entries qualify.

The following relational operators can be used:

- less than or equal (<=)
- greater than or equal (>=)
- equal (= ␣ or ␣ =), where ␣ indicates a space character

The value must be specified exactly as it is stored in the database. A lock will succeed even if no data item with the specified value exists in the data set; no check is made during the DBLOCK procedure 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 can be used in a lock descriptor; that is, the lock item need not be a search item.

TurboIMAGE/XL 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 can specify any number of lock descriptors with a single DBLOCK call. For example, the following lock descriptors can be specified in one DBLOCK call:

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

NOTE 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 chapter.

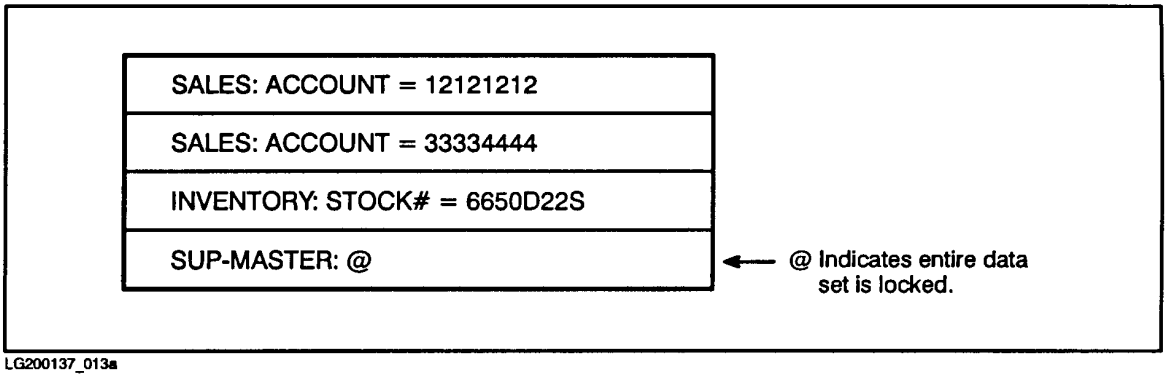
How Locking Works

The internal implementation of locking does not involve reading or writing to the database element to be locked. TurboIMAGE/XL keeps a table of everything that is locked by all processes that have the database opened. One table is associated with each database. This table serves as a global list of lock descriptors. In locking mode 5 or 6, a database 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/XL 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/XL compares the newly specified lock descriptors with those that are currently in the list. If a conflict exists, TurboIMAGE/XL 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/XL adds the new lock descriptors to the list.

Users whose programs have MR capability and issue multiple DBLOCK calls, without a DBUNLOCK call between the DBLOCK calls, may cause a deadlock if DBLOCK calls are not done carefully. Recovery from a deadlock requires a restart of the operating system. If it is desired to activate the database for automatic deadlock detection, DBCONTROL mode 7 must be done prior to the first DBLOCK call for the database. In this case, when a possibility of a potential deadlock is detected by the DBLOCK procedure, a status code of 26 is returned to the calling process. The calling process must call DBUNLOCK to release all locks acquired by DBLOCK(s) for the same access path.

Figure 4-3. Lock Descriptor List



Conditional and Unconditional Locking

You can request conditional or unconditional locking. If you request unconditional locking, TurboIMAGE/XL returns control to your calling program only after the specified entity has been locked. If you request conditional locking, TurboIMAGE/XL 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 if only 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/XL 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 can be a data entry, data set, or database lock.
- To add to or delete from (DBPUT or DBDELETE) a manual master data set, you must first successfully lock the data set or database. To update (DBUPDATE) a master data set, data entry level locks are sufficient.

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

TurboIMAGE/XL 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 database, 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.

Because access mode 3 and 7 users have exclusive control of the database 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 database, whole data sets, or data entries. TurboIMAGE/XL allows mixed levels of locking. For example, one user could 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 database locks also. If data set or data entry locks are in effect at the time a database lock is requested, the database lock must wait until they are released and all subsequent locking requests must wait behind the pending database lock. In either case, if the request is for a conditional lock, an exceptional condition is generated. (Refer to the "Locking Mode Options" table in chapter 5.)

Deciding on a Locking Strategy

It is important, especially for on-line interactive applications, to establish a locking strategy at application design time. In general, locking is related to the **transaction**, the basic unit of work performed against a database. TurboIMAGE/XL transactions are either single or logical, and logical transactions can be static, multiple database, or dynamic. Refer to "User Logging and Logical Transactions" later in this chapter for more details and to chapter 5 for additional information.

Typically a transaction consists of several calls to TurboIMAGE/XL intrinsics to locate and modify data. For example, a transaction to add a new order with three line items could 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 database 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 (with DBPUT, DBDELETE, or DBUPDATE) and/or all data entries which must not be changed by other processes during the transaction.*

Choosing a Locking Level

Because TurboIMAGE/XL needs more information to lock data entries than to lock the whole database, program complexity tends to increase as locks are employed at lower and lower levels. Locking the whole database 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 concurrency; however, there are situations in which the extra programming effort for data entry locking is not worthwhile. Concurrency is least optimum at the higher level of the lock. Concurrency and programming effort should be considered; some other considerations that could affect your choice of locking level are discussed below.

Locking at the Same Level

All programs concurrently accessing a database 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 database 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 can 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, you should probably lock at the entry level. For shorter transactions, you can use locks at either the database or data set level with satisfactory 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 can read some data entries, interact with a terminal operator, and modify some or all of the entries. A lock to cover this transaction can last several minutes which is an unacceptable amount of time to stop all database or data set activity. In this situation, data entry level locking should be used.

Because 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 time-out feature of MPE/iX can be used to avoid having the locked entity inaccessible when the terminal operator is interrupted in the middle of the dialog. The time-out feature can 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/iX Intrinsics Manual*.

Strong Locking and Dynamic Transactions

Dynamic transactions, which are described later in this chapter, are only allowed with a database access mode that enforces locking, because strong locking is required for this type of transaction. TurboIMAGE/XL requires that dynamic transactions be independent of all other types of transactions. This is guaranteed when the database access mode is 3 or 4, because the mode guarantees exclusive modify access.

When a database is opened in access mode 1, the programmer must ensure that strong locks are in place. In other words, any call to DBUNLOCK must occur after the call to DBXEND,

or the dynamic transaction is aborted.

NOTE A call to DBXUNDO must be processed if an error occurs or if the transaction needs to be rolled back for any other reason, because TurboIMAGE/XL will not go on to the next transaction in the event of a transaction abort unless an intervening call to DBXUNDO occurs. Furthermore, if DBXUNDO was used to roll back a transaction, the program logic should ensure that the subsequent call to DBXEND is not processed.

Choosing an Item for Locking

An important convention to follow in designing a locking scheme is that all programs sharing the database concurrently use the same data item to lock data entries in a particular data set. At any one time, TurboIMAGE/XL allows no more than one data item per data set to be used for locking purposes. However, several values of the data item can be locked at the same time. For example, if one process has successfully locked SALES:ACCOUNT = 54321000, another process could 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 because 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 can be useful to add comments in the schema indicating which item is the locking item for each set. If a chain is used heavily for chained reads, its search item is a prime candidate for a lock item.

Examples of Locking

The examples in this section show the order in which TurboIMAGE/XL intrinsics can be called when locking is used. The ORDERS database is used in the examples. (Refer to the ORDERS database schema in chapter 3.) For descriptions of the procedures used in these examples, refer to chapter 5.

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

Table 4-4. 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 database. This is mandatory if the database is open in access mode 1.

Add a New Customer

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

Note that TurboIMAGE/XL requires a data set or database lock to cover the addition of an entry to a master data set.

Update Inventory Information

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

Insert a New Product with a New Supplier

1. DBLOCK the PRODUCT master data set, the SUP-MASTER data set, and the data item STOCK# = 4444A33B in the INVENTORY detail data set. (This can be done in one DBLOCK mode 5 call.)
2. DBBEGIN.
3. DBPUT a new data entry in PRODUCT master data set. (For example: 4444A33B CALIPER).

4. DBPUT a new data entry in SUP-MASTER data set.
5. DBPUT a new data entry in INVENTORY data set for STOCK# = 4444A33B.
6. DBEND.
7. DBUNLOCK.

The locking in the above example was done around the entire transaction to maintain data consistency during the multistep transaction.

Interactively Modify a Customer Account Order

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 the user until the correct order is located.
4. DBUPDATE the contents of the data entry according to the user's request.
5. DBUNLOCK.

In this interactive example, 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 could 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, when a mode 6 DBLOCK is called with lock descriptor SALES: ACCOUNT = 89393899 and the lock does not succeed, a message similar to the one below can be displayed by the program:

```
RECORDS BEING MODIFIED.  WANT TO WAIT?
```

If the response is NO, then proceed with other processing. If the answer is YES, call DBLOCK again with mode 5.

Issuing Multiple Calls to DBLOCK

In order to guarantee that two processes cannot deadlock, TurboIMAGE/XL does not allow two DBLOCK calls to be made without a DBUNLOCK between the DBLOCK calls. Two exceptions to this rule are stated here:

- A redundant call can be made to lock the whole database 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 can be made if the program from which multiple DBLOCK calls are issued has the MPE/iX Multiple RIN (MR) capability.

The DBLOCK procedure is similar to the MPE/iX FLOCK procedure in that DBLOCK can put a process into a waiting state and thus can cause a deadlock to occur. For example, a deadlock can occur if process A is waiting for an MPE/iX file to be freed by process B, and process B is waiting for a database entity to be unlocked by process A. Therefore, issuing a DBLOCK in conjunction with a lock applied by an MPE/iX intrinsic, such as 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/XL guarantees that deadlocks will never occur provided that no executing program that accesses the database has MR capability. Programs that execute successfully using TurboIMAGE/XL 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 database are relinquished when the process calls DBUNLOCK; they are automatically relinquished when the process closes the database, 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 mode 5 or 6 should lock the chain in the detail data set. This prevents the execution of any DBPUTs or DBDELETEs to the detail data set from modifying the current chain, thereby preventing a status 18 (broken chain) error.

TurboIMAGE/XL Logging Services

TurboIMAGE/XL provides logging through the following two MPE/iX services:

- Transaction Management (XM) logging
- User logging

XM logging is used to ensure the physical integrity of the database. It is transparent to the user because it is automatically enabled *unless AUTODEFER is enabled*. Dynamic roll-back recovery (explained in chapter 7) works in conjunction with XM and ensures the logical integrity of the database. Dynamic roll-back uses XM to roll back dynamic transactions online while other database activity is occurring. Dynamic transactions are defined later in this chapter. User logging, which is described below, is not required for dynamic roll-back recovery, but is recommended to guard against a hard disk crash.

User logging is used to ensure the logical integrity of the database. It is initiated by the user and can be used for several purposes. For example, user logging is required to perform roll-forward recovery. It can also be used to keep a record of individual database transactions (see the description of the `>FILE` command of `DBRECOV` in chapter 8). In addition, it is sometimes used for audit purposes to keep a record of all database transactions. Because some user logging considerations are important to applications, they are discussed below. For a more complete discussion of TurboIMAGE/XL logging services, refer to chapter 7.

What User Logging Does

The user logging and recovery facility enables all database modifications to be logged automatically to a tape or disk log file. In the event of a system failure the log file 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 log file is actually a record of all modifications to the database. The intrinsic `DBMEMO`, capable of logging user text, facilitates interpretation of the log files for future reference.

The database administrator is responsible for enabling or disabling the logging and recovery processes and generating backup database copies, thus making logging a global function controlled at the database level rather than at the individual user level. A process is said to be logging if all of the following are true:

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

How User Logging Works

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

DBXBEGIN, DBXEND, and DBXUNDO. In addition, DBQUIESCE procedures called by True-Online Backup to quiesce and unquiesce the database are also logged to the logging file.

TurboIMAGE/XL calls the MPE/iX logging intrinsics OPENLOG, WRITELOG, and CLOSELOG in order to log information to the log file. When a database is opened, DBOPEN calls the OPENLOG intrinsic using the log identifier and password stored in the database root file. If this call succeeds, DBOPEN calls WRITELOG to log a DBOPEN log record containing information about the database and the new user. The WRITELOG intrinsic is also used to log information when the TurboIMAGE/XL intrinsics DBPUT, DBDELETE, and DBUPDATE are called. WRITELOG is called after all error checks are made, but before actually modifying the working database. Consequently, a log record is not written until the TurboIMAGE/XL procedure has committed itself to succeed. WRITELOG is also used by the TurboIMAGE/XL intrinsics DBBEGIN, DBEND, DBMEMO, DBXBEGIN, DBXEND, DBXUNDO, and DBQUIESCE procedures.

DBCLOSE (mode 1) calls WRITELOG to log a DBCLOSE log record, and then calls CLOSELOG to terminate access to the log file. If a transaction initiated with DBBEGIN fails to call DBEND, or a DBXBEGIN does not have a matching DBXEND, DBCLOSE causes an abnormal DBEND or DBXEND log record to terminate access to the log file. DBCLOSE also causes an abnormal DBEND or DBXEND log record to be written if the program aborts with an unfinished transaction.

User Logging and Logical Transactions

TurboIMAGE/XL transactions are of two basic types--single and logical. A **single transaction** is a single call to an intrinsic. It is not delimited by begin and end intrinsics. A **logical transaction** can be considered as the basic work unit performed against a database. A logical transaction could consist of a single modification, but more typically consists of several calls to TurboIMAGE/XL intrinsics which lock, read, modify, and unlock information. Logical transactions transfer the database from one consistent state to another, but in the midst of a multiple-step transaction, the database could be temporarily inconsistent with itself. (For an example, see "Logical Transactions" in chapter 7.)

TurboIMAGE/XL logical transactions are defined in Table 4-5.

Table 4-5. Types of Logical Transactions

Transaction	Definition
Static	A logical transaction that begins with a DBBEGIN call and ends with a DBEND call. A static transaction spans only one database. This type of transaction can be recovered with DBRECOV.
Multiple database	A logical transaction that spans more than one database. A multiple database transaction begins with a DBBEGIN call and ends with a DBEND call. This type of transaction is recovered with DBRECOV.
Dynamic	A logical transaction that begins with a DBXBEGIN call and ends with a DBXEND call. Unlike non-dynamic transactions (that is, static and multiple database transactions), dynamic transactions can be rolled back with a call to the DBXUNDO procedure and are automatically rolled back in the event of a system failure or program abort.

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

For reasons explained more fully under "Locking Requirements for Logical Transactions" in chapter 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 be changed by other processes during the transaction. This includes data to be read and data to be modified.
2. If you wish, read data using DBFIND and DBGET to determine the needed modifications.
3. Call DBBEGIN or DBXBEGIN to declare the beginning of modifications.
4. Make modifications using DBPUT, DBDELETE, or DBUPDATE.
5. If this is a dynamic transaction, call DBXUNDO in the event an error is encountered or the modifications should be backed out for any other reason.
6. Call DBEND or DBXEND to declare the end of the modifications. However, if this is a dynamic transaction and DBXUNDO was used to roll back a transaction, your program logic should ensure that the subsequent call to DBXEND is not processed.
7. Call DBUNLOCK to release all of the locks.

NOTE The call to DBUNLOCK *must* occur after the call to DBXEND in the case of a dynamic transaction when the database is opened in access mode 1, because TurboIMAGE/XL requires strong locking for dynamic transactions.

Transaction Numbers

TurboIMAGE/XL maintains a transaction number for each user accessing the database. Transaction numbers enable the DBRECOV recovery program to associate one access path's log records with a particular transaction. This number is initialized by DBOPEN and incremented each time DBBEGIN or DBXBEGIN 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, or DBXBEGIN or DBXEND. Transaction numbers are included in all DBBEGIN, DBXBEGIN, DBPUT, DBUPDATE, DBDELETE, and DBMEMO log records. The transaction number is always incremented as described, regardless of whether the user's process is actually logging. A user's process can determine its transaction count (and whether the database and user is logging) by calling DBINFO using mode 401.

User Logging and Process Suspension

The MPE/iX logging intrinsics will suspend a calling process if the logging buffers become full. Consequently, a user's process which calls TurboIMAGE/XL can become suspended, for example, if a tape log file reaches the end of a reel and logging buffers become full before a new tape can be mounted.

Obtaining Database Structure Information

The DBINFO library procedure allows you to acquire information programmatically about the database. It provides information about database settings, data items, data sets, data paths, B-Tree indices, and third-party indices. The information returned is restricted by the user class number and access mode established when the database is opened.

Any data items, data sets, or paths of the database 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 can 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 database 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.
- 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.
- Determine the number of items in the database 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.
- 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.
- Get a data set description including the data set name, type, length in halfwords and blocking factor for data entries in the set, number of entries in the set, and the capacity.
- 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.
- 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.
- Obtain capacity information. Determine if a data set is jumbo, and if so, information on chunks.
- Determine if master data set has a B-Tree index.

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.

- Determine the key 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/XL 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 because a change to the database 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/3000 database inquiry facility. DBINFO can also be used to obtain information regarding the logging facility, dynamic transactions, third-party indexing, and the critical item update (CIUPDATE) option settings for a database and the current process. In relation to Native Language Support (NLS), DBINFO can be used to get the MPE/iX numeric code that defines the native language supported by the database. (Refer to "Database Description Language" and "Schema Structure" in chapter 3.)

Checking Subsystem Flag

A subsystem flag can be set with the DBUTIL >>SET command. This flag indicates whether subsystems, including user programs, can access the TurboIMAGE/XL database 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. QUERY/3000 is the primary subsystem which uses the subsystem flag.

Closing the Database or a Data Set

After you have completed all the tasks with the database, use the DBCLOSE library procedure to terminate access to the database. When DBCLOSE is used for this purpose, all data sets and the root file are closed, and the DBU control block is released to the MPE/iX system. If there are no other concurrent users of the database, the DBB and DBG control blocks are also released. All locks that you could still have on the database 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 to its initial state the dynamic status information kept by TurboIMAGE/XL. 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/iX system without terminating access to the database. 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.

Checking the Status of a Procedure

Each time a procedure is called, TurboIMAGE/XL returns status information in a buffer specified by the calling program and, if the program is in Compatibility Mode, sets the CM condition code maintained by MPE/iX. The CM condition code, or the TurboIMAGE/XL return status (described later), should be checked immediately after TurboIMAGE/XL returns from the procedure to the calling program before another procedure call is made.

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

CM 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 **return status** whose value corresponds to the CM condition code as follows:

CM Condition Code	Return Status Value
CCE	0
CCG	>0
CCL	<0

The calling program must check either the CM condition code or the TurboIMAGE/XL return status to determine the success or failure of the procedure. The return status is also used to indicate various exceptional conditions and errors. See appendix A for a summary.

The other elements of status information vary with the outcome of the call and from one procedure to another. The content of these elements is described in detail with each procedure definition later in chapter 5 and in appendix A, which describes error conditions.

Interpreting Errors

TurboIMAGE/XL 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 database and the name of the procedure that returned the status information. DBERROR returns the English language error message contained in a buffer specified by the calling program.

Abnormal Termination

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

5 TurboIMAGE/XL Library Procedures

This chapter contains the reference specifications for the TurboIMAGE/XL library procedures (also known as intrinsics), arranged alphabetically. Table 5-1. gives a summary of the procedures with a brief description of their function.

Using TurboIMAGE/XL Intrinsics

On the following pages, the calling parameters for each procedure are defined in alphabetical order for easy look-up rather than the order in which they appear in the call statement. Every parameter must be included when a call is made because a parameter's meaning is determined by its position.

NOTE All parameters must be on halfword boundaries. Database names, data set names, and data item names that are passed to the TurboIMAGE/XL intrinsics must be in uppercase.

Table 5-1. TurboIMAGE/XL Procedures

Procedure	Function
DBBEGIN	When logging, designates the beginning of a transaction and optionally writes user information to the log file.
DBCLOSE	Terminates access to a database or a data set, or resets the pointers of a data set to their original state.
DBCONTROL	Allows a process operating in exclusive mode to enable or disable the deferred output (AUTODEFER) option. Also allows a process to enable or disable the critical item update (CIUPDATE) option, activate deadlock detection, and set wildcard character and BTREEMODE1 option for database, or for the current DBOPEN without affecting other processes operating on the same database.
DBDELETE	Deletes an existing entry from a data set.
DBEND	When logging, designates the end of a transaction and optionally writes user information to the log file.
DBERROR	Supplies an ASCII language message that interprets the status information set by any callable TurboIMAGE/XL procedure. The message is returned to the calling program in a buffer.
DBEXPLAIN	Examines status information returned by a TurboIMAGE/XL procedure and prints a multiline message on the $STDLIST device.
DBFIND	Locates the first and last entries of a data chain in preparation for access to entries in the chain for non-B-Tree searches. For B-Tree searches, master and detail data sets can be included.
DBGET	Reads the data item values of a specified entry.
DBINFO	Provides information about the database being accessed, such as the name and description of a data item or data set. It also provides information on logging, including logging of dynamic and multiple database transactions, and on third-party indexing, critical item update, and other options.
DBLOCK	Locks one or more data entries, a data set, or an entire database (or a combination of these) temporarily to allow the process calling the procedure to have exclusive access to the locked entities.

Table 5-1. TurboIMAGE/XL Procedures

Procedure	Function
DBMEMO	When logging, writes user information to the log file.
DBOPEN	Initiates access to a database. Specifies the user access mode and user class number for the duration of the process.
DBPUT	Adds a new entry to a manual master or detail data set.
DBUNLOCK	Releases those locks obtained with previous call(s) to DBLOCK.
DBUPDATE	Modifies the values of data items. Cannot be used to update master data set key items. Can be used to update detail data set search and sort items in database access mode 1, 3, or 4 if permitted by the critical item update (CIUPDATE) option settings for the database and the current process.
DBXBEGIN	Designates the beginning of a dynamic transaction. Refer to "Transactions" later in this chapter for a description of dynamic transactions.
DBXEND	Designates the end of a dynamic transaction.
DBXUNDO	Rolls back the active dynamic transaction.

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

Table 5-2. Calling a TurboIMAGE/XL Procedure

COBOL II	CALL "*name*" USING *parameter,parameter,...,parameter*.
FORTRAN 77	CALL *name* (*parameter,parameter,...,parameter*)
Pascal	*name* (*parameter,parameter,...,parameter*);
BBASIC	*linenumber* CALL *name* (*parameter,parameter,...,parameter*)
C	*name* (*parameter,parameter,...,parameter*);

All procedures can be called from programs in any of the host languages.

Intrinsic Numbers

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

Database Protection

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

Unused Parameters

When calling some procedures for a specific purpose, one of the parameters can be ignored; however, it still must be listed in the call statement. An application program may find it useful to set up a variable named `Not_Used_Parm` or `DUMMY` to be listed as the unused parameter as a reminder that the value of the parameter does not affect the procedure call. Refer to the examples in chapter 6.

The Status Array

The status array is a communication area. If the procedure executes successfully, the contents of the array reflect this as described under each intrinsic discussion in this chapter. If the procedure fails, standard error information is returned in the array as described in this chapter and appendix A.

Transactions

TurboIMAGE/XL transactions are defined below:

Table 5-3. Types of Transactions

Transaction	Definition
Single	A single call to an intrinsic. A single transaction is not delimited by `DBBEGIN` and `DBEND`, or `DBXBEGIN` and `DBXEND`.
Logical	A sequence of one or more procedure calls that begins with a `DBBEGIN` or `DBXBEGIN` call and ends with a `DBEND` or `DBXEND` call. A logical transaction can contain several intrinsic calls, but is logically considered one transaction.
Static	A logical transaction that begins with a `DBBEGIN` call and ends with a `DBEND` call. A static transaction spans only one database, and can be recovered with `DBRECOV`.
Dynamic	A logical transaction that begins with a `DBXBEGIN` call and ends with a `DBXEND` call. A dynamic transaction can be rolled back dynamically with `DBXUNDO`. A dynamic transaction spans only one database.
Multiple database	A logical transaction that spans more than one database. A multiple database transaction begins with a `DBBEGIN` call and ends with a `DBEND` call. A multiple database transaction can be recovered with `DBRECOV`. A dynamic transaction can also span more than one database by beginning with a `DBXBEGIN` call and ending with a `DBEND` call.

Refer to chapters 4 and 7 for more information on transactions.

DBBEGIN

INTRINSIC NUMBER 412

Designates the beginning of a sequence of TurboIMAGE/XL procedure calls regarded as a static or multiple database transaction (based on the mode) for the purposes of logging and recovery. The *text* parameter can be used to log user information to the log file. DBBEGIN is used in conjunction with DBEND to begin and end a static or multiple database transaction.

Syntax

```
DBBEGIN,{ base
        baseidlist} ,text,mode,status,textlen
```

Parameters

base is the name of the array used as the base parameter when opening the database. The first element of the array must contain the base ID returned by DBOPEN. (Refer to DBOPEN for more information about the base ID.) Use *base* when calling DBBEGIN mode 1 (static transaction).

baseidlist is the name of the integer array containing the base IDs of the databases which are involved in a multiple database transaction. Use *baseidlist* when calling DBBEGIN mode 3 or 4 (multiple database transaction). The layout of this array is shown here (each element is a halfword or two bytes):

Element	Contents
1-2	The application program must set these two halfwords to binary 0s before calling DBBEGIN. After returning to the calling program, these two halfwords contain the transaction ID. This ID can be used in order to end the transaction by calling DBEND.
3	The number of base IDs involved in the multiple database transaction. This must be a number between 1 and 15, inclusive.
4-*n*	Base IDs of the databases involved in the transaction. Base ID is the first halfword of the *base* parameter used to call TurboIMAGE/XL intrinsics.

text is the name of an array up to 256 halfwords long which contains user ASCII or binary data to be written to the log file 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 is an integer indicating the type of transaction desired as follows:

Mode 1: Indicates a static transaction.

Mode 3: Indicates a multiple database transaction with one log record per database. If user logging is enabled for the databases, mode 3 generates *multiple* entries in the log file in order to mark multiple database transactions. For example, assume that base IDs 11, 12, and 13 are involved in a multiple database transaction. DBBEGIN mode 3 (with base IDs 11, 12, and 13 specified in the *baseidlist* parameter) generates the following log record sequence:

```
DBBEGIN (11, 1/3)
DBBEGIN (12, 2/3)
DBBEGIN (13, 3/3)
          .
          .
          .
    database updates
          .
          .
          .
DBEND (11, 1/3)
DBEND (12, 2/3)
DBEND (13, 3/3)
```

where the notations 1/3, 2/3, 3/3 in the log records indicate "first of three," "second of three," and "third of three." Refer to chapter 7 for more details about user logging.

Mode 4: Indicates a multiple database transaction. If user logging is enabled for the databases, mode 4 generates *one* entry in the log file in order to mark multiple database transactions. For example, assume that base IDs 11, 12, and 13 are involved in a multiple database transaction. DBBEGIN mode 4 (with base IDs 11, 12, and 13 specified in the *baseidlist* parameter) generates the following log record sequence:

```
MDBXBEGIN (11, 12, 13)
          .
          .
          .
    database updates
          .
          .
          .
MDBXEND (11, 12, 13)
```

Refer to chapter 7 for more details about user logging.

status is the name of an array of 10 halfwords in which TurboIMAGE/XL returns status information. If the procedure executes successfully, the status array contents are:

Element	Contents
1	If the procedure succeeds, the return status is 0. Table 5-3. describes the contents of element 1 when the procedure does not succeed.

	2-4	Unchanged from previous procedure call using this array.
	5-10	Procedure call information. Refer to "Library Procedure Error Messages" in appendix A for a description of this information.

textlen is an integer equal to the number of halfwords to be logged from the *text* parameter, or is a negative integer equal to the number of bytes to be logged. Length can be zero.

Discussion

DBBEGIN is called to designate the beginning of a sequence of TurboIMAGE/XL procedure calls which are jointly considered a single logical transaction. The transaction is either a static or multiple database transaction, based on the mode. 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 log file which includes such information as the time, date, and user text buffer. DBBEGIN log records are used by the database recovery program DBRECOV to identify the beginning of all logical transactions.

DBBEGIN returns an error condition if it is called twice without an intervening call to DBEND or if it is called while a dynamic transaction is still active, whether the process is actually logging or not.

Table 5-4. DBBEGIN Return Status Values

Calling Errors:	-11	Bad database reference.
	-31	Bad mode.
	-139	Invalid number of base IDs.
	-140	Bad base ID list.
	-141	All MDBX databases must be on the same system.
	-142	All MDBX databases must log to the same log file.
	-143	Logging must be enabled or disabled for all MDBX databases.
	-144	MUSTRECOVER must be enabled or disabled for all MDBX databases.
	-145	Roll-back must be enabled or disabled for all MDBX databases.
	-151	Text length greater than 512 bytes.
	-152	Transaction is in progress.
	-221	Cannot begin transaction when a dynamic transaction is active.
	-222	Only DBXUNDO allowed when a dynamic transaction encounters an error.

Table 5-4. DBBEGIN Return Status Values

Communications Errors:	-102	DSWRITE failure.
	-106	Remote 3000 data inconsistent.
	-107	NS 3000 or DS 3000 system error.
Logging System Failures:	-111	WRITELOG failure.
Exceptional Conditions:	-193	DBU control block is full.
	-332	Error in QLOCK table operation.
	62	DBU full.
	63	DBG disabled; potential damage; only DBCLOSE allowed.

Consult appendix A for more information about these conditions.

DBCLOSE

INTRINSIC NUMBER 403

Terminates access to a database or a data set, or rewinds a data set. DBCLOSE is used in conjunction with DBOPEN to establish and terminate access to a database.

Syntax

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

Parameters

base is the name of an array used as the *base* parameter when opening the database. The first element 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 can be up to 16 characters long. If shorter, it must be terminated by a semicolon or a blank.

mode is an integer equal to 1, 2, or 3 indicating the type of termination desired as follows:

Mode 1: Access to the database is terminated. Any locks held by this user for this base ID are released. If DBCLOSE mode 1 is called while a dynamic transaction is still active, an error is returned, the transaction is aborted, and the database is closed automatically. You do not need to call DBXEND or DBXUNDO.

Mode 2: The data set referenced by the *dset* array is closed, but locks held in the data set are not released. If DBCLOSE mode 2 is called while a dynamic transaction is still active, an error is returned. You must check the error and decide to use DBXEND, DBXUNDO, or continue with the transaction even if DBCLOSE mode 2 failed. DBXUNDO will abort the entire dynamic transaction. DBXEND will terminate the dynamic transaction; the modifications completed thus far within the transaction will remain in the database.

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

If your database is enabled for third-party indexing (TPI), refer to your vendor documentation for additional DBCLOSE mode information. The section on DBUTIL in chapter 8 of this book has a brief description of the

TPI option.

status is the name of an array of 10 halfwords in which TurboIMAGE/XL returns status information about the procedure. If the procedure executes successfully, the status array contents are:

Element	Contents
1	If the procedure succeeds, the return status is 0. Table 5-5. describes the contents of element 1 when the procedure does not succeed.
2-4	Unchanged from previous procedure call using this array.
5-10	Procedure call information. Refer to "Library Procedure Error Messages" in appendix A for a description of this information.

Discussion

You must call DBCLOSE mode 1 to terminate access to the database when you have completed all the tasks you want to perform. If a process has issued multiple calls to DBOPEN for the same database, 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. The current list is not reset. Table 5-5. summarizes the functions performed in each mode.

Table 5-5. DBCLOSE Modes 2 and 3 Functions

Function	Mode 2	Mode 3
Reinitialize dynamic status information for the data set (chain count, forward and backward pointers, current record number and last return status).	YES	YES
Close the data set.	YES	NO
Release locks held within the data set.	NO	NO
Current list reset.	NO	NO

Because mode 3 does not close and reopen a data set, it is more efficient than mode 2 if the data set is to be accessed again before the database is closed.

Only mode 3 is allowed within a dynamic transaction; mode 2 will return an error, and mode 1 will abort the transaction.

If the process is logging, a mode 1 DBCLOSE causes a DBCLOSE log record to be written to the log file. 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 database. If the process aborts before completing an active transaction, a special DBEND log record is written prior to the DBCLOSE.

DBCLOSE returns an error condition if the process has not completed an active transaction; in other words, the process has called DBBEGIN without a matching call to DBEND. Transactions that 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.

Table 5-6. DBCLOSE Return Status Values

File System, Memory Management, and Transaction Management Failures:	-2	FCLOSE failure.
	-3	FREADDIR failed.
	-5	FWRITEDIR failure.
	-6	FWRITELABEL failure.
Calling Errors:	-11	Bad database reference.
	-21	Bad data set reference.
	-31	Bad mode.
	-222	Only DBXUNDO allowed when a dynamic transaction encounters an error.
	-232	Illegal DBCLOSE mode 2 used during an active dynamic transaction.
	-235	Dynamic transaction aborted due to DBCLOSE mode 1; database closed.
	-420	Feature not implemented.
Communications Errors:	-101	DSCLOSE failure.
	-102	DSWRITE failure.
	-103	Remote 3000 stack space insufficient.
	-106	Remote 3000 data inconsistent.
	-107	NS 3000 or DS 3000 system error.
Logging System Failures:	-111	WRITELOG failure.
	-112	CLOSELOG failure.
	-152	Transaction is in process.
Exceptional Conditions:	-194	Invalid DBB.
	-332	Error in QLOCK table operation.
	-333	Error in QOPEN table operation.
	63	DBG disabled; potential damage; only DBCLOSE allowed.

Consult appendix A for more information about these conditions.

DBCONTROL

INTRINSIC NUMBER 411

Allows a process accessing the database to have control over some DBUTIL options. In exclusive mode (DBOPEN mode 3), DBCONTROL can be used to enable or disable the deferred output (AUTODEFER) option and perform several B-Tree related tasks such as adding, dropping, or rebuilding a B-Tree index. It can also be used to set BTREEMODE1 option to ON or OFF and specify the wildcard character for the current DBOPEN or for the database (permanent).

For database access modes 1, 3, and 4, DBCONTROL can be used to temporarily enable or disable the critical item update (CIUPDATE) option, depending on the setting for the database, without impacting other processes operating on the same database. Using DBCONTROL does not alter the permanent AUTODEFER or CIUPDATE settings for the database set with DBUTIL. However, the permanent setting for BTREEMODE1 and wildcard character can be altered based on the DBCONTROL mode.

Syntax

```
DBCONTROL,base,qualifier,mode,status
```

Parameters

base is the name of the array used as the base parameter when opening the database. The first element of the array must contain the base ID returned by DBOPEN. (Refer to DBOPEN for more information about the base ID.)

qualifier used only for certain DBCONTROL modes.

mode must be an integer equal to 1, 2, 5, 6, 7, 9, 10, 13, 14, 15, or 16 indicating the following:

Mode 1: Turn on the deferred output option. If AUTODEFER has not been enabled for the database (using DBUTIL >>ENABLE), mode 1 enables the deferred output option for the duration of only the current DBOPEN. When the database is closed, AUTODEFER will no longer be enabled. Mode 1 is not allowed while a dynamic transaction is active.

Mode 2: Turn off the deferred output option. If AUTODEFER has been enabled for the database (using DBUTIL >>ENABLE), mode 2 disables the deferred output option for the duration of only the current DBOPEN. When the database is closed, AUTODEFER will again be enabled.

Mode 5: Enable the critical item update option. If CIUPDATE is ALLOWED for the database (default or by using DBUTIL >>SET), mode 5 enables the option for the current DBOPEN until either a DBCONTROL mode 6 call disables the option

or the database is closed. You can call mode 5 successfully if the CIUPDATE setting for the database equals ON, but the call has no impact on the option setting for the current process unless an intervening call to DBCONTROL mode 6 disabled the option. If the CIUPDATE flag is DISALLOWED, a call to mode 5 returns an error. The CIUPDATE option is available only in database access modes 1, 3, and 4.

Mode 6: Disable the critical item update option. If CIUPDATE has been set to ON for the database (using DBUTIL>>SET), mode 6 disables the option for the current DBOPEN until either a DBCONTROL mode 5 call enables the option or the database is closed. If the CIUPDATE option setting for the database equals ALLOWED and the process has called DBCONTROL in mode 5 to enable the option, then mode 6 disables the option for that same process. The CIUPDATE option is available only in database access modes 1, 3, and 4.

Mode 7: Allow the database to be included in the **dynamic multiple database transaction** (**DMDBX**). DBCONTROL mode 7 needs to be done once, for every database, before including it in DBXBEGIN mode 3 call for DMDBX. Mode 7 is used programmatically to allow it for DMDBX and remains activated until the database is closed or the application terminates.

Mode 7 also activates the database for deadlock detection. In case of a deadlock, DBLOCK will return an error, 26, instead of causing a process hang. Note that mode 7 will activate both deadlock detection and inclusion in DMDBX. If you only want deadlock detection and not DMDBX, your DBXBEGIN call can use mode 1 instead of 3.

Mode 9: Enable the HWMPUT option of DBPUT for the current DBOPEN. This causes DBPUT to try placing entries at the high-water mark first instead of at the delete chain head first.

Mode 10: Disable the HWMPUT option of DBPUT for the current DBOPEN. This causes DBPUT to try placing entries at the delete chain head first. This is the default action if DBCONTROL is not called.

Mode 13: Allow a privileged mode caller to perform B-Tree related tasks such as adding, dropping, rebuilding, releasing, or securing a B-Tree index file. Exclusive database access is required to add, drop, or rebuild a B-Tree index. Qualifier has a structured record containing dataset information and directives. Refer to chapter 11, "B-Tree Indices" for the qualifier layout.

Mode 14: Allow a privileged mode caller to set BTREEMODE1 option to ON or OFF and set wildcard character for the database (permanent). Refer to chapter 11, "B-Tree Indices" for the qualifier layout.

Mode 15: Enable B-Tree wildcard search for DBFIND mode 1, X and U types, for the master data set's key item and its corresponding search items in details, if the key item has a B-Tree index. In other words, turn on the BTREEMODE1 flag for the current database open. The first byte of the qualifier is examined. If it is null or blank, then the current wildcard character is not changed. If it is in the ASCII range (33...126), then the wildcard character is changed to that value for the current DBOPEN. If the qualifier is any other value, DBCONTROL returns an error. Affects only the current database open. It needs to be done once per database and remains activated until the database is closed or the application terminates. This mode allows you to perform B-Tree searches on ASCII types without making application changes.

Mode 16: Disable B-Tree wildcard search for DBFIND mode 1 regardless of the existence of a B-Tree index. That is, turn off the BTREEMODE1 flag for the current database open. The qualifier is ignored, and BTREEMODE 1 is turned off. Affects the current database open only. It needs to be done once per database and remains activated until the database is closed or the application terminates.

If your database is enabled for third-party indexing (TPI), refer to your vendor documentation for additional DBCONTROL modes. The section on DBUTIL in chapter 8 of this book has a brief description of the TPI option.

status is the name of an array of 10 halfwords in which TurboIMAGE/XL returns status information. If the procedure executes successfully, the status array contents are as follows:

Element	Contents
1	If the procedure succeeds, the return status is 0. Table 5-6. describes the contents of element 1 when the procedure does not succeed.
2-4	Unchanged from previous procedure call using this array.
5-10	Procedure call information. Refer to "Library Procedure Error Messages" in appendix A for a description of this information.

Discussion

DBCONTROL can be called by a program for the following purposes:

- To enable or disable the AUTODEFER option for the current mode 3 DBOPEN.
- To enable or disable the critical item update option for the current process accessing the database in mode 1, 3, or 4 depending on the CIUPDATE setting for the database.
- To enable or disable the HWMPUT for the current DBOPEN.
- To include a database in dynamic multiple database transaction.
- To activate deadlock detection for the database.
- To perform B-Tree index related tasks.

In TurboIMAGE/XL default mode, MPE/iX Transaction Management (XM) is used to log database modifications (DBPUTs, DBDELETEs, DBUPDATEs) to the XM log file. With deferred output, MPE/iX Transaction Management is not used. Instead, the MPE/iX file system default mode is used. This mode keeps data pages in memory for as long as possible, either until file close time or until no more memory is available.

Thus, with deferred output, database modifications caused by calls to DBPUT, DBUPDATE, or DBDELETE cannot be written to the disk (or can only be partially written). Although TurboIMAGE/XL generally operates more efficiently in this mode, a system failure while the database is operating in this mode has a very high probability of causing internal structural damage to the database.

A program that opens the database exclusively can call DBCONTROL mode 1 to enter the deferred mode of operation, except when a dynamic transaction is active. In this case, all database modifications will be kept in memory for as long as possible, either until the database is closed or until no more memory is available.

A program that opens the database exclusively can call DBCONTROL mode 2 to turn off the deferred mode of operation. In this case, all database modifications will be written to the MPE/iX Transaction Management log file until the database is closed.

Programs that are designed to modify the values of detail data set search and sort items can call DBCONTROL mode 5 if the CIUPDATE setting for a database equals ALLOWED. The CIUPDATE option is available only in database access modes 1, 3, and 4. The mode 5 call enables CIUPDATE for only this process until either a DBCONTROL mode 6 call disables the option for the process or the database is closed. Other processes operating on the same database are not impacted by these calls.

A program, which must ensure that the values of detail data set search and sort items remain unchanged for the duration or a portion of the process, can call DBCONTROL mode 6 to disable CIUPDATE if this option has been set to ON for the database, or if the option is ALLOWED (default or set by DBUTIL) and an earlier call to DBCONTROL mode 5 enabled the option. CIUPDATE is available only in database access modes 1, 3, and 4. When DBCONTROL mode 6 is used to disable CIUPDATE, the option is disabled for that process alone until a call to DBCONTROL mode 5 enables the option for the process or the database is closed. Other processes operating on the same database are not affected by these calls.

NOTE If HWMPUT is enabled, DBPUT will not inform you when it has reached the end of file and has started using the delete chain head. In general, it is not a good practice to toggle HWMPUT.

If you plan to use roll-forward recovery, do not toggle HWMPUT after storing the database.

By default, DBPUT first checks the delete chain head, then if it is empty, DBPUT places the new entry at the high-water mark. If the high-water mark option (HWMPUT) is enabled, DBPUT will place the entry at the high-water mark first; after the high-water mark reaches the file limit, DBPUT will use the delete chain head. Use DBCONTROL mode 9 to enable or mode 10 to disable this feature.

DBCONTROL allows DBUTIL and privileged callers to perform several B-Tree index file related functions such as adding, deleting, or rebuilding of a B-Tree index file for a specified master dataset. There are four new DBCONTROL modes pertaining to B-Tree indices: modes 13, 14, 15, and 16. Refer to chapter 11, "B-Tree Indices," for more information.

Consult appendix A for more information about the conditions for the return status values shown in Table 5-7.

Table 5-7. DBCONTROL Return Status Values

File System, Memory Management, and Transaction Management Failures:	-4	FREADLABEL failure.
	-168	Cannot attach *n* to MPE XL XM: file system error *nn*.
	-175	Cannot attach *n* to MPE XL XM: XM error *nn*.
	-176	Cannot detach *n* from MPE XL XM: XM error *nn*.
	-178	Cannot detach *n* from MPE XL XM: file system error *nn*.
	-179	Cannot begin MPE XL transaction for attach: XM error *nn*.
	-189	Cannot begin MPE XL transaction for detach: XM error *nn*.

Table 5-7. DBCONTROL Return Status Values

Calling Errors:	-11	Bad database reference.
	-14	Illegal intrinsic in current access mode.
	-31	Bad mode.
	-80	Output deferred not allowed when ILR enabled.
	-81	Output deferred not allowed with roll-back enabled.
	-82	CIUPDATE is set to DISALLOWED; cannot use critical item update.
	-222	Only DBXUNDO allowed when a dynamic transaction encounters an error.
	-224	DBCONTROL mode 1 not allowed inside a dynamic transaction.
	-421	BTE: unknown qualifier value for DBCONTROL mode 13.
Calling Errors (cont):	-422	BTE: data set# not in valid range.
	-423	BTE: B-Tree already exists.
	-424	BTE: Failed to create B-Tree.
	-425	BTE: DB not opened exclusively.
	-426	BTE: B-Tree doesn't exist.
	-427	BTE: FCLOSE, purge failed.
	-428	BTE: Rebuildindex failed.
	-432	BTE: Bad wildcard character.
	-434	BTE: Data set is detail and not master.
	-436	BTE: Failed to extract data from root file.
	-440	BTE: XM Attach of index file failed
	-441	BTE: XM Detach of index file failed.
	-442	BTE: RELEASE of index file failed.
	-443	BTE: SECURE of index file failed.
	-451	BTE: Root version less than "C"4.
	-452	BTE: Key length greater than 252 bytes (maximum index key size).
	92	Need PM capability.
Communications Errors:	-102	DSWRITE failure.
Exceptional Conditions:	63	DBG disabled; potential damage; only DBCLOSE allowed.

DBDELETE

INTRINSIC NUMBER 408

Deletes the current entry from a manual master or detail data set. The database must be opened 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 database. The first element 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 can be up to 16 characters long. If shorter, it must be terminated by a semicolon or a blank.

mode must be an integer equal to 1.

If your database is enabled for third-party indexing (TPI), refer to your vendor documentation for additional DBDELETE mode information. The section on DBUTIL in chapter 8 of this book has a brief description of the TPI option.

status is the name of an array of 10 halfwords in which TurboIMAGE/XL returns status information about the procedure. If the procedure executes successfully, the status array contents are:

Element	Contents
1	If the procedure succeeds, the return status is 0. Table 5-8. describes the contents of element 1 when the procedure does not succeed.
2	Zero.
3-4	Unchanged current record number.
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.

Discussion

When deleting entries from *detail* data sets, and if the database is open in access mode 1, you must establish a lock covering the data entry to be deleted, the data set, or the database.

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 cannot be any detail entries on the paths defined by the master which have the same search item value as the key item in the master entry to be deleted.
- If the database is open in access mode 1, a lock must be in effect on the data set or the whole database.

DBDELETE to an indexed master triggers a similar operation to the indexed master's B-Tree file and is considered atomic with the DBDELETE intrinsic.

Because of the way TurboIMAGE/XL 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/XL moves the first synonym in the chain to the deleted primary's location. A subsequent DBGET mode 3 will read the next sequential entry, leaving an entry (the new primary) in the previous location.

A solution to this problem is to check elements 5 and 6 of the status parameter following each DBDELETE call. If the synonym count in these elements 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 in a serial manner. (Refer to chapter 4 for a discussion of serial access and to chapter 10 for a discussion of synonym chains.)

TurboIMAGE/XL 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.

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.

The execution of a call to DBDELETE could require extensive resources depending on the amount of chain maintenance required. For example, when an entry is deleted from a detail data set, the links connecting that entry to all other related entries with the same key values and to all other related master entries are eliminated. This operation could involve many blocks of data. TurboIMAGE/XL prevents data block access conflicts with all other users and ensures data integrity by applying a temporary lock against other processes until the call to DBDELETE completes. The timing of this temporary lock can be controlled with the PREFETCH option of DBUTIL. Refer to "Coordinating Deletions to a

Database" in chapter 4 for what to consider when enabling or disabling this option.

If the process is logging, a call to DBDELETE causes a log record to be written with such information as the time, date, user identification number, and a copy of the record to be deleted. In a dynamic transaction, DBDELETE causes a log record to be written after the physical transaction has been successfully completed. If DBDELETE cannot complete within a dynamic transaction, an error is returned. This error condition must be checked, and you must decide to use DBXUNDO, DBXEND, or continue with the remainder of the dynamic transaction. DBXUNDO will abort the entire transaction. DBXEND will terminate the dynamic transaction; the modifications completed thus far within the transaction will remain in the database.

Table 5-8. DBDELETE Return Status Values

File System, Memory Management, and Transaction Management Failures:	-1	FOPEN failure.
	-3	FREADDIR failure.
	-4	FREADLABEL failure.
	-5	FWRITEDIR failure.
	-6	FWRITELABEL failure.
	-167	Cannot begin MPE XL XM transaction: XM error *nn*.
	-168	Cannot attach *n* to MPE XL XM: file system error *nn*.
	-169	Invalid mode for XM attach options.
	-175	Cannot attach *n* to MPE XL XM: XM error *nn*.
	-176	Cannot detach *n* from MPE XL XM: XM error *nn*.
	-178	Cannot detach *n* from MPE XL XM: file system error *nn*.
	-199	Cannot end MPE XL XM transaction: XM error *nn*.
	-209	Invalid mode for XM detach options.
Calling Errors:	-11	Bad database reference.
	-12	No lock covers the data entry to be deleted. (Occurs only if database open in access mode 1.)
	-14	Illegal intrinsic in current access mode.
	-21	Bad data set reference.
	-23	Data set not writable.
	-24	DBDelete not allowed on Auto Master.
	-31	Bad mode.
	-222	Only DBXUNDO allowed when a dynamic transaction encounters an error.

Table 5-8. DBDELETE Return Status Values

Communications Errors:	-102	DSWRITE failure.
	-106	Remote 3000 data inconsistent.
	-107	NS 3000 or DS 3000 system error.
Logging System Failures:	-111	WRITELOG failure.
Exceptional Conditions:	-193	DBU control block is full.
	-196	DBB control block is full.
	-264	Error while writing to TPI files.
	-3*nn*	Internal error.
	-314	Error while obtaining patch information for set.
	-322	Error while validating qualifier parameter.
	-332	Error in QLOCK table operations.
	17	No entry.
	44	Can't delete master entry with non-empty detail chains.
	63	DBG disabled; potential damage; only DBCLOSE allowed.

Consult appendix A for more information about these conditions.

DBEND

INTRINSIC NUMBER 413

Designates the end of a sequence of TurboIMAGE/XL procedure calls regarded as a static or multiple database transaction (based on the mode) for the purposes of logging and recovery. The *text* parameter can be used to log user information to the log file. DBEND is used in conjunction with DBBEGIN to begin and end a static or multiple database transaction.

Syntax

```
DBEND,{ base
        baseidlist
        transid      } ,text,mode,status,textlen
```

Parameters

base is the name of the array used as the base parameter when opening the database. The first element of the array must contain the base ID returned by DBOPEN. (Refer to DBOPEN for more information about the base ID.) Use with DBEND mode 1 or 2.

baseidlist is the name of an integer array containing the list of base IDs which are involved in a multiple database transaction. Use with DBEND mode 3 or 4, and set the first two halfwords to binary zeroes. The layout of this array is shown here (each element is a halfword, or two bytes):

Element	Contents
1-2	Must be set to binary zeroes.
3	The number of base IDs involved in the multiple database transaction. Must be a number between 1 and 15, inclusive.
4-*n*	Base IDs of the databases involved in the transaction. Base ID is the first halfword of the *base* parameter used to call TurboIMAGE intrinsics.

transid is the name of the integer array containing the two-halfword transaction ID. The transaction ID was returned by DBBEGIN mode 3 or 4. Use with DBEND mode 3 or 4, and do not set the first two halfwords to binary zeroes.

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

mode must be an integer equal to 1, 2, 3, or 4.

Mode 1: End of static transaction.

Mode 2: Write contents of the logging buffer in memory to disk, and end the static transaction.

Mode 3: End of multiple database transaction. If user logging is enabled for the databases, mode 3 generates *multiple* entries in the log file in order to mark multiple database transactions. For example, assume that base IDs 11, 12, and 13 are involved in a multiple database transaction. DBEND mode 3 (with base IDs 11, 12, and 13 specified in the *baseidlist* parameter) generates the following log record sequence:

```
DBBEGIN (11, 1/3)
DBBEGIN (12, 2/3)
DBBEGIN (13, 3/3)
          .
          .
          .
    database updates
          .
          .
          .
DBEND (11, 1/3)
DBEND (12, 2/3)
DBEND (13, 3/3)
```

where the notations 1/3, 2/3, 3/3 in the log records indicate "first of three," "second of three," and "third of three." Refer to chapter 7 for more information about user logging.

Mode 4: Write contents of the logging buffer in memory to disk, and end the multiple database transaction. If user logging is enabled for the databases, mode 4 generates *one* entry in the log file in order to mark multiple database transactions. For example, assume that base IDs 11, 12, and 13 are involved in a multiple database transaction. DBEND mode 4 (with base IDs 11, 12, and 13 specified in the *baseidlist* parameter) generates the following log record sequence:

```
MDBXEND (11, 12, 13)
          .
          .
          .
    database updates
          .
          .
          .
MDBXEND (11, 12, 13)
```

Refer to chapter 7 for more information about user logging.

status is the name of an array of 10 halfwords in which TurboIMAGE/XL returns status information about the procedure. If the procedure executes successfully, the status array contents are as follows:

Element	Contents
1	If the procedure succeeds, the return status is 0. Table 5-8. describes the contents of element 1 when the procedure does not succeed.
2-4	Unchanged from previous procedure call using this array.
5-10	Procedure call information. Refer to "Library Procedure Error Messages" in appendix A for a description of this information.

textlen is an integer equal to the number of halfwords to be logged from the *text* parameter, or is a negative integer equal to the number of bytes to be logged. Length can be zero.

Discussion

DBEND is called to designate the end of a sequence of TurboIMAGE/XL procedure calls which are collectively considered a static or multiple database transaction. The beginning of such a sequence is designated by a previous call to DBBEGIN.

NOTE DBEND is not valid if a transaction was begun with DBXBEGIN. DBEND checks for any active dynamic transactions before executing.

If the process is logging, DBEND causes a log record to be written to the log file which includes such information as the time, date, and user text buffer. DBEND log records are used by the database recovery program DBRECOV to identify the end of static and multiple database transactions. However, if a program aborts, a static or multiple database transaction that has not been completed by a call to DBEND will be recovered by default. For additional information, refer to the discussion of the ABORTS and NOABORTS options under the description of the DBRECOV >CONTROL command in chapter 8.

If you call DBEND with mode 2 or 4 and logging is enabled, DBEND forces the log buffer to be written from memory to disk before returning to the calling process. This flush of the log buffer occurs after the intrinsic has logged the end of the logical transaction. Use this option only for critical transactions; too many mode 2 or mode 4 DBEND calls can degrade performance by causing a disk access each time a static or multiple database transaction ends.

NOTE When you call DBEND with mode 2 or 4 to force writing a static or multiple data base transaction to disk, logging must have been enabled prior to executing the transaction.

DBEND returns an error condition if it is called without a prior matching call to DBBEGIN,

whether the process is actually logging or not.

Table 5-9. DBEND Return Status Values

Calling Errors:	-11	Bad database reference.
	-31	Bad mode.
	-146	Invalid transaction ID.
	-147	Mode doesn't match DBBEGIN mode.
	-148	Base ID list doesn't match DBBEGIN base ID list.
	-151	Text length greater than 512 bytes.
	-153	No transaction in progress to end.
	-216	Cannot end a dynamic transaction with a DBEND.
	-222	Only DBXUNDO allowed when a dynamic transaction encounters an error.
Communications Errors:	-102	DSWRITE failure.
	-106	Remote 3000 data inconsistent.
	-107	NS 3000 or DS 3000 system error.
Logging System Failures:	-111	WRITELOG failure.
	-113	FLUSHLOG returned error number to DBEND.
Exceptional Conditions:	-193	DBU control block is full.
	63	DBG disabled; potential damage; only DBCLOSE allowed.

Consult appendix A for more information about these conditions.

DBERROR

INTRINSIC NUMBER 419

Moves a 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/XL procedure.

Syntax

```
DBERROR,status,buffer,length
```

Parameters

status	is the name of the array used as the *status* parameter in the TurboIMAGE/XL procedure call about which information is requested.
buffer	is the name of an array in the calling program's data area, at least 36 halfwords long, in which the message is returned.
length	is a short, 16-bit integer variable which is set by DBERROR to the positive byte length of the message placed in the *buffer* array. The length will never exceed 72 characters.

NOTE The call to DBERROR must be made immediately after receiving an error status before any other intrinsics are executed to ensure the display of valid messages.

Discussion

Like DBEXPLAIN, DBERROR messages are appropriate and helpful when 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/iX intrinsic DBOPEN can fail due to concurrent database access. In this case, printing the following DBERROR message:

```
DATABASE OPEN EXCLUSIVELY
```

can 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 5-10. lists all messages that can be returned by DBERROR with their corresponding return status values. Variable information is represented by a lowercase word or phrase. Several messages can correspond to one return status and the interpretation of the value depends on the context in which it is returned; the message returned depends on additional information returned by the TurboIMAGE/XL intrinsic.

Table 5-10. DBERROR Messages

Return Status	DBERROR Message
0	SUCCESSFUL EXECUTION - NO ERROR
-1	NO SUCH DATABASE DATABASE OPEN IN AN INCOMPATIBLE MODE BAD ACCOUNT REFERENCE or BAD GROUP REFERENCE BAD ROOT FILE REFERENCE INSUFFICIENT DISC SPACE VIRTUAL MEMORY NOT SUFFICIENT TO OPEN ROOT FILE DATABASE ALREADY OPEN FOR MORE THAN READ DATABASE IN USE DATABASE OPEN EXCLUSIVELY MPE SECURITY VIOLATION MPE FILE ERROR *decimal integer* RETURNED BY FOPEN ON { *ROOT FILE* *DATA SET* # *decimal integer* }
-2	EXCEEDS GROUP DISC SPACE EXCEEDS ACCOUNT GROUP DISC SPACE DUPLICATE FILE NAME MPE FILE ERROR *decimal integer* RETURNED BY FCLOSE ON { *ROOT FILE* *DATA SET* # *decimal integer* }
-3	MPE FILE ERROR *decimal integer* RETURNED BY FREADDIR ON { *ROOT FILE* *DATA SET* # *decimal integer* }
-4	MPE FILE ERROR *decimal integer* RETURNED BY FREADLABEL ON { *ROOT FILE* *DATA SET* # *decimal integer* }
-5	MPE FILE ERROR *decimal integer* RETURNED BY FWRITEDIR
-6	MPE FILE ERROR *decimal integer* RETURNED BY FWRITELABEL
-7	PREVIOUS MPE FILE ERROR *decimal integer* FOUND IN DESIRED BUFFER
-8	MPE FILE ERROR *decimal integer* RETURNED BY FUNLOCK
-9	CANNOT CREATE *control block name*: MPE ERROR *nn*
-10	MPE FILE ERROR *decimal integer* RETURNED BY FFILEINFO

Table 5-10. DBERROR Messages

Return Status	DBERROR Message
-11	BAD DATABASE NAME OR PRECEDING BLANKS MISSING BAD DATABASE REFERENCE (FIRST 2 CHARACTERS)
-12	DATABASE MUST BE IN LOGON GROUP AND ACCOUNT *intrinsic name* CALLED WITHOUT COVERING LOCK IN EFFECT
-13	NOT ALLOWED; MUST BE CREATOR OF ROOT FILE OR DATABASE
-14	CALLS TO *intrinsic name* NOT ALLOWED IN ACCESS MODE *decimal integer*
-15	DSLINE OR REMOTE HELLO FAILURE: SETUP FOR RDBA FAILED
-21	BAD PASSWORD - GRANTS ACCESS TO NOTHING DATA ITEM NONEXISTENT OR INACCESSIBLE SPECIFIED INTRINSIC CANNOT ACCESS THE DATA SET DATA SET NONEXISTENT OR INACCESSIBLE BAD MAINTENANCE WORD (CONTAINS COMMA OR DOES NOT MATCH) ILLEGAL NUMBER OF BUFFERS REQUESTED
-22	MAINTENANCE WORD REQUIRED
-23	USER (CLASS) LACKS WRITE ACCESS TO DATA SET
-24	OPERATION NOT ALLOWED ON AUTOMATIC MASTER DATA SET
-30	MPE V ILR ENABLED; ONLY DBOPEN (MODE 1-8) AND DBUTIL DISABLE ILR ALLOWED
-31	DBGET MODE *decimal integer* ILLEGAL FOR DETAIL DATA SET DBGET MODE *decimal integer* BAD--SPECIFIED DATA SET LACKS CHAINS BAD (UNRECOGNIZED) *intrinsic name* MODE: *decimal integer*
-32	UNOBTAINABLE ACCESS MODE: AOPTIONS REQUESTED:*%octal integer*, GRANTED:*%octal integer*
-33	MODE 7 DIAGNOSTICS NOT ALLOWED
-34	DATABASE MUST BE RECOVERED BEFORE ACCESS IS ALLOWED.
-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 FOR ROOT FILE
-61	ERROR WHILE OBTAINING INFORMATION ABOUT FILE EQUATION.
-80	OUTPUT DEFERRED NOT ALLOWED WITH ILR ENABLED
-81	OUTPUT DEFERRED NOT ALLOWED WITH ROLLBACK ENABLED

Table 5-10. DBERROR Messages

Return Status	DBERROR Message
-82	CIUPDATE IS SET TO DISALLOWED; CANNOT USE CRITICAL ITEM UPDATE
-88	DATABASE BAD: THIRD PARTY INDEXING WAS IN PROCESS (INDEX AGAIN)
-89	DATABASE BAD RESTRUCTURING WAS IN PROCESS (RESTORE DATABASE)
-90	ROOT FILE BAD: UNRECOGNIZED STATE: *%octal integer*
-91	ROOT FILE (DATABASE) NOT COMPATIBLE WITH CURRENT TURBOIMAGE INTRINSICS
-92	DATABASE REQUIRES CREATION (VIRGIN ROOT FILE)
-93	DATABASE ALREADY EXISTS.
-94	DATABASE BAD: OUTPUT DEFERRED, MAY NOT BE ACCESSED IN MODE *decimal integer*
-95	DATABASE BAD - CREATION WAS IN PROCESS (CREATE AGAIN)
-96	DATABASE BAD - ERASE WAS IN PROCESS (ERASE AGAIN)
-97	DATABASE BAD - ILR ENABLE IN PROGRESS (ENABLE AGAIN)
-98	DATABASE BAD - ILR DISABLE IN PROGRESS (DISABLE AGAIN)
-99	UNSUPPORTED FEATURE.
-100	MPE ERROR *decimal integer* RETURNED BY DSOPEN
-101	MPE ERROR *decimal integer* RETURNED BY DSCLOSE
-102	MPE ERROR *decimal integer* RETURNED BY DSWRITE
-103	REMOTE 3000 STACK SPACE INSUFFICIENT
-104	REMOTE 3000 DOES NOT HAVE TURBOIMAGE/XL.
-105	REMOTE 3000 CANNOT CREATE TURBOIMAGE CONTROL BLOCK
-106	REMOTE 3000 DATA INCONSISTENT
-107	NS/3000 OR DS/3000 SYSTEM ERROR
-108	HPUNLOADCMPROCEDURE CALL FAILED (The bracketed numbers in the following messages –110 τo – 112 refer to the value in halfword 2 of the status array. For other error numbers, refer to WRITELOG in the MPE/iX Intrinsics Manual.)
-109	ERROR RETURNED BY LOGINFO INTRINSIC.

Table 5-10. DBERROR Messages

Return Status	DBERROR Message
-110	OPENLOG RETURNED ERROR NUMBER *nn* TO DBOPEN LOGGING ENABLED AND NO LOG PROCESS RUNNING [3] DATABASE CONTAINS INVALID LOGID 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] DATABASE CONTAINS INVALID LOG IDENTIFIER [16]
-111	WRITELOG RETURNED ERROR NUMBER *nn* TO *intrinsic name* LOG PROCESS TERMINATED [3] LOG FILE CAN'T OBTAIN NECESSARY DISC SPACE [12] END OF FILE ON LOGFILE [15]
-112	CLOSELOG RETURNED ERROR NUMBER *nn* TO *intrinsic name* LOG PROCESS TERMINATED [3] LOG FILE CAN'T OBTAIN NECESSARY DISC SPACE [12] END OF FILE ON LOGFILE [15]
-113	FLUSHLOG RETURNED ERROR NUMBER *nn* TO DBEND
-114	ROLLBACK ENABLED WITHOUT ENABLING LOGGING
-120	OUT OF STACK SPACE TO PROCV.
-121	ILLEGAL LOCK DESCRIPTOR COUNT
-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 4094 BYTES
-137	USER ABOUT TO WAIT FOR SELF.

Table 5-10. DBERROR Messages

Return Status	DBERROR Message
-139	INVALID NUMBER OF BASE IDs.
-140	BAD BASE ID LIST.
-141	ALL MDBX DATABASES MUST BE ON THE SAME SYSTEM.
-142	ALL MDBX DATABASES LOG TO THE SAME LOG FILE.
-143	LOGGING MUST BE ENABLED OR DISABLED FOR ALL MDBX DATABASES.
-144	MUSTRECOVER MUST BE ENABLED OR DISABLED FOR ALL MDBX DATABASES.
-145	ROLL-BACK MUST BE ENABLED OR DISABLED FOR ALL MDBX DATABASES.
-146	INVALID TRANSACTION ID.
-147	MODE DOESN'T MATCH DBBEGIN MODE.
-148	BASE ID LIST DOESN'T MATCH DBBEGIN BASE ID LIST.
-151	TEXT LENGTH GREATER THAN 512 BYTES
-152	DBCLOSE CALLED WHILE A TRANSACTION IS IN PROGRESS DBBEGIN CALLED WHILE A TRANSACTION IS IN PROGRESS DBXBEGIN 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 AN ILR LOG FILE CONFLICT: FILE SYSTEM ERROR *nn*
-166	CANNOT PURGE ILR LOG FILE: FILE SYSTEM ERROR *nn*
-167	CANNOT BEGIN MPE XL XM TRANSACTION: XM ERROR *nn*
-168	CANNOT ATTACH DATA SET *# decimal integer* ``to MPE XL XM: FILE SYSTEM ERROR *nn*
-169	INVALID MODE FOR XM ATTACH OPTIONS
-170	CANNOT OPEN ILR LOG FILE: FILE SYSTEM ERROR *nn*
-172	CANNOT READ ILR LOG FILE: FILE SYSTEM ERROR *nn*
-173	UNABLE TO OBTAIN WRITE ACCESS TO THE DATABASE (NEEDED FOR ILR RECOVERY)
-174	THE DATABASE MUST BE OPENED IN MODE 1 - 4 SINCE ILR RECOVERY IS NECESSARY
-175	CANNOT ATTACH DATA SET # *decimal integer* TO MPE XL XM: XM ERROR *nn*
-176	CANNOT DETACH DATA SET # *decimal integer* FROM MPE XL XM: XM ERROR *nn*

Table 5-10. DBERROR Messages

Return Status	DBERROR Message
-177	USER LOG FILE IS NOT ON THE SAME VOLUME SET AS DATABASE
-178	CANNOT DETACH DATA SET # *decimal integer* FROM MPE XL XM: FILE SYSTEM ERROR *nn*
-179	CANNOT BEGIN MPE XL XM TRANSACTION FOR ATTACH: XM ERROR *nn*
-180	ILR LOG INVALID - ILR INCOMPATIBLE IN MPE XL
-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
-187	ILR ALREADY ENABLED FOR THIS DATABASE
-188	ILR ALREADY DISABLED FOR THIS DATABASE
-189	CANNOT BEGIN MPE XL TRANSACTION FOR DETACH: XM ERROR *nn*
-190	BAD DATABASE REFERENCE OR INVALID SYSTEM DATABASE CONTROL BLOCK
-191	SYSTEM DATABASE CONTROL BLOCK FULL
-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
-198	TOTAL DBOPEN COUNT PER USER EXCEEDS LIMIT
-199	CANNOT END MPE XL XM TRANSACTION: XM ERROR *nn*
-200	DATABASE LANGUAGE NOT SYSTEM SUPPORTED
-201	NATIVE LANGUAGE SUPPORT NOT INSTALLED
-202	MPE NATIVE LANGUAGE SUPPORT ERROR *nn* RETURNED BY NLINFO
-204	USER STACK IS TOO SMALL FOR RECOVERY IN DBOPEN.
-205	WRONG VERSION OF DS SUBSYSTEM.

Table 5-10. DBERROR Messages

Return Status	DBERROR Message
-206	REMOTE TURBOIMAGE DATABASE EXCEEDS IMAGE/3000 LIMITATION
-208	MPE ERROR *nn* RETURNED BY FLABELINFO FOR MPE XL XM
-209	INVALID MODE FOR MPE XL XM DETACH
-210	MPE ERROR *nn* WHILE GETTING LOG FILE NAME
-211	INVALID OR NO USER LABEL.
-212	DATABASE CORRUPTION DETECTED
-213	DBXEND ENCOUNTERED XM ERROR *nn* WHEN ENDING DYNAMIC TRANSACTION
-214	CANNOT CALL DBXUNDO WHEN A TRANSACTION IS STARTED BY DBBEGIN.
-215	XM ERROR *nn* ENCOUNTERED WHEN ROLLING OUT DYNAMIC TRANSACTION
-216	CANNOT END A DYNAMIC TRANSACTION WITH A DBEND
-217	DBOPEN MODE *n* INCOMPATIBLE WITH DYNAMIC ROLLBACK
-218	OUTPUT DEFERRED IS INCOMPATIBLE WITH DYNAMIC ROLLBACK
-219	REMOTE DATABASE ACCESS IS INCOMPATIBLE WITH DYNAMIC ROLLBACK
-220	DATABASE AND USER LOG NOT ATTACHED TO THE SAME XM LOG SET
-221	CANNOT BEGIN A TRANSACTION WHEN A DYNAMIC TRANSACTION IS ACTIVE
-222	ONLY DBXUNDO ALLOWED WHEN A DYNAMIC TRANSACTION ENCOUNTERS AN ERROR
-223	CANNOT DBXEND OR DBXUNDO A TRANSACTION WHICH WAS NOT ACTIVE
-224	DBCONTROL MODE 1 NOT ALLOWED INSIDE A DYNAMIC TRANSACTION
-225	RECORD TABLE FULL FOR DYNAMIC ROLLBACK
-226	ERROR OCCURRED WHEN CREATING THE 00 FILE
-227	ERROR OCCURRED IN 00 FILE RECOVERY
-228	DBXBEGIN ENCOUNTERED XM ERROR *nn* WHEN STARTING A DYNAMIC TRANSACTION
-229	CANNOT DELETE MANUAL MASTER WITH EMPTY CHAINS.
-230	A DBUNLOCK INSIDE A DYNAMIC TRANSACTION IS NOT ALLOWED
-231	DURING DYNAMIC ROLLBACK RECOVERY, INTERNAL PROCEDURE FAILED; ERROR *nn*
-232	ILLEGAL DBCLOSE MODE 2 USED DURING AN ACTIVE DYNAMIC TRANSACTION

Table 5-10. DBERROR Messages

Return Status	DBERROR Message
-233	KEY DATA FOUND IN THE DATABSE DOES NOT MATCH THAT IN THE MEMO RECORD
-234	CANNOT PURGE THE 00 FILE
-235	DYNAMIC TRANSACTION ABORTED DUE TO DBCLOSE MODE 1; DATABASE CLOSED
-236	INTERNAL ERROR OCCURRED WHEN OPENING THE AUX FILE; ERROR *nn*
-237	CANNOT DBXEND OR DBXUNDO A DBBEGIN TRANSACTION
-238	MDBX, MODES OF DBXBEGIN/DBXEND DON'T MATCH.
-240	ERROR IN DYNAMIC ROLLBACK.
-241	BAD TAG FOR TURBOLKT TABLE.
-242	ERROR IN TURBOGTX FILE OPERATION.
-243	INVALID DYNAMIC ROLLBACK TRANSACTION ID.
-244	BASE COUNT OVER 15 FOR DMDBX.
-245	OUT OF SPACE FOR TURBOGTX FILE.
-246	ERROR IN TURBOGTX OPERATION RELATED TO ATC TRANSACTION.
-250	CBINIT FAILED ON *nn*
-251	DBS WAS OBTAINED BUT NOT RELEASED.
-253	DATABASE ENABLED FOR INDEXING, BUT THIRD-PARTY INDEXING IS NOT CONFIGURED
-254	ROLLBACK OF THIRD-PARTY INDEX FAILED; INDEXING DISABLED FOR DATA SET
-255	THIRD-PARTY INDEXING DISABLE FAILED; INDEXING DISABLED FOR DATABASE
-256	THIRD-PARTY INDEX FOR PATH *decimal integer: nn* IS FULL
-257	THIRD-PARTY INDEX FOR PATH *decimal integer: nn* IS DAMAGED
-258	INVALID ARGUMENT FOR INDEX
-259	INVALID MODE FOR INDEX
-260	NO PREVIOUS LIST OF QUALIFIED DATA ENTRIES
-261	DYNAMIC PROCEDURE LOAD ERROR FOR INTRINSIC ROLLBACK.
-262	OLDER/INCOMPATIBLE VERSION OF IMAGE/SQL.
-263	INVALID PCODE RETURNED BY TPI.

Table 5-10. DBERROR Messages

Return Status	DBERROR Message
-264	WRITE ERROR FOR TPI FILES.
-265	ERROR IN THIRD-PARTY SHADOWING PACKAGE.
-266	ERROR WHILE DISABLING THIRD-PARTY SHADOWING.
-267	DAMAGED FILE ERROR RETURNED BY THIRD-PARTY SHADOWING.
-268	INVALID PCODE RETURNED BY TPS.
-269	WRITE ERROR FOR TPS FILES.
-3*nn*	INTERNAL TURBOIMAGE ERROR RETURNED (*#n*)
-305	INVALID DATA SET NUMBER.
-306	INVALID DATA SET TYPE.
-307	INVALID RECORD NUMBER FOUND.
-308	ERROR RELATED TO BEGINNING OF FILE.
-309	BUFFER IO NOT YET COMPLETE.
-310	ERROR RELATED TO END OF FILE.
-312	INTERNAL ERROR ENCOUNTERED WHILE READING DATABASE BLOCK.
-314	ERROR WHILE OBTAINING PATH INFORMATION FOR SET.
-322	INTERNAL TURBOIMAGE ERROR RETURNED *nn*
-323	UNEXPECTED EMPTY RECORD FOUND.
-331	DSET CAPACITY INFORMATION NOT CURRENT.
-332	ERROR IN QLOCK OPERATION.
-333	ERROR IN QOPEN OPERATION.
-420	FEATURE NOT IMPLEMENTED.
-421	BTE:UNKNOWN QUALIFIER VALUE FOR DBCONTROL MODE 13.
-422	BTE: DATA SET # NOT IN VALID RANGE.
-423	BTE: B-TREE ALREADY EXISTS.
-424	MESSAGE\BTE: FAILED TO CREATE B-TREE.
-425	MESSAGE\BTE: DB NOT OPENED EXCLUSIVELY.
-426	BTE: B-TREE DOESN'T EXIST.
-427	BTE: FCLOSE, PURGE FAILED.
-428	BTE: REBUILDINDEX FAILED.

Table 5-10. DBERROR Messages

Return Status	DBERROR Message
-429	BTE: DBFIND ARGUMENT VERSION IS BAD.
-430	BTE: DBFIND (mode 4/24) ARGUMENT TYPE IS BAD.
-431	BTE: DBFIND (mode 4/24) ARGUMENT #1 LENGTH IS BAD.
-432	BTE: WILDCARD NOT ASCII.
-433	BTE: DBFIND (MODE 4/24) ARGUMENT #2 LENGTH IS BAD.
-434	DATASET DETAIL INSTEAD OF MASTER.
-436	BTE: FAILED TO EXTRACT DATA FROM ROOT FILE.
-437	BTE: FAILED TO CONVERT @c TO [] DBFIND.
-438	BTE: BAD ITEM # IN INIT BTREE.
-439	BTE: CONVERSION OF KEY FROM EXTERNAL TO INTERNAL FORMAT FAILED.
-440	BTE: XM ATTACH OF INDEX FILE FAILED.
-441	BTE: XM DETACH OF INDEX FILE FAILED.
-442	BTE: RELEASE OF INDEX FILE FAILED.
-443	BTE: SECURE OF INDEX FILE FAILED.
-444	BTE: DBFIND ON NON-KEY FIELD OF MASTER.
-446	BTE: ARGUMENT 2 SPECIFIED FOR RELOP OF (</<=/=/>=/>).
-447	BTE: FAILED TO BUILD RECORD HOLDING ROOT DATA INFORMATION.
-448	BTE: FAILED TO SETUP INFORMATION FOR USERLABEL 0 OF DATASET.
-449	BTE: FAILED TO POSITION INDEX AT START OF KEY RANGE.
-451	BTE: ROOT VERSION LESS THAN "C4".
-452	BTE: KEY LENGTH GREATER THAN 252 BYTES (MAXIMUM INDEX KEY SIZE).
-458	DBOPEN FAILED. OUT OF DISK SPACE
-1000	SWITCH TO NM FAILED ON *intrinsic name*, INFO *nn* SUBSYS *nn*
-1001	SWITCH TO CM FAILED ON CX'PCBXIMAGE
-1002	HPLOADCMPROCEDURE FAILED ON CX'PCBXIMAGE
-1003	SWITCH TO NM FAILED ON *intrinsic name*, INFO *nn* SUBSYS *nn*
-1004	HPLOADNMPROC FAILED ON CM *intrinsic name*
-3999 to -3000	ERROR RETURNED BY THIRD-PARTY INDEXING PRODUCTS.

Table 5-10. DBERROR Messages

Return Status	DBERROR Message
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 (The following messages are returned only if capacity expansion is specified for the data set. *Dataset#* and FSERR # are replaced by the actual number.) DBPUT CANNOT EXPAND *dataset#*: DATA SET AT MAXIMUM CAPACITY DBPUT *dataset#* INCOMPLETE EXPANSION: FILE SYSTEM ERROR # DBPUT CANNOT EXPAND *dataset#*: OUT OF DISC SPACE (FSERR #)
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)
18	BROKEN CHAIN - FORWARD AND BACKWARD POINTERS NOT CONSISTENT
20	DATABASE CURRENTLY LOCKED SETS OR ENTRIES LOCKED WITHIN DATABASE
22	DATA SET ALREADY LOCKED
23	CANNOT LOCK SET DUE TO LOCKED ENTRIES WITHIN IT *(Conditional Locks Only)*
24	ENTRIES CURRENTLY LOCKED USING DIFFERENT ITEM *(Conditional Locks Only)*
25	CONFLICTING DATA ENTRY LOCK ALREADY IN EFFECT
26	IMMINENT DEADLOCK.

Table 5-10. DBERROR Messages

Return Status	DBERROR Message
41	DBUPDATE ATTEMPTED TO MODIFY VALUE OF CRITICAL ITEM--KEY, SEARCH OR SORT (The following messages are returned only if the critical item update (CIUPDATE) option is permitted for the database and the current process. The bracketed numbers after the messages refer to the value in halfword 3 of the status array. The *nn* in the value represents the path number returned.) DBUPDATE: NO CHAIN HEAD (MASTER ENTRY) FOR PATH *decimal integer: nn* [1*nn*] DBUPDATE: FULL CHAIN FOR PATH *decimal integer: nn* (CONTAINS 2,147,483,647 ENTRIES) [2*nn*] DBUPDATE: FULL AUTOMATIC MASTER FOR PATH *decimal integer: nn* [3*nn*] DBUPDATE: FULL AUTOMATIC MASTER SYNONYM CHAIN FOR PATH *decimal integer: nn* [4*nn*]
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
49	ILLEGAL BUFFER ADDRESS
50	USER'S BUFFER IS TOO SMALL FOR REQUESTED DATA (only returned if buffer is too small and the data transfer would write over in the user's stack)
51	STACK OVERFLOW FOR BASIC - IMAGE INTERFACE.
52	INVALID PARAMETER FOR BASIC - IMAGE INTERFACE.
53	INVALID PARAMETER TYPE FOR BASIC - IMAGE INTERFACE.
60	DATABASE ACCESS DISABLED
61	PROCESS HAS THE DATABASE OPEN 63 TIMES; NO MORE ALLOWED
62	DBG CONTROL BLOCK FULL
63	DBG DISABLED; POTENTIAL DAMAGE; ONLY DBCLOSE ALLOWED
64	NO ROOM FOR DBG ENTRY IN PCBX (MPE PORTION OF STACK)
65	CAN'T GRANT BUFFER REQUEST.
66	DBG POINTED TO BY ROOT FILE DOES NOT MATCH
67	DBU DISABLED; POTENTIAL DAMAGE; ONLY DBCLOSE ALLOWED
68	DBB DISABLED; POTENTITIAL DAMAGE; ONLY DBCLOSE ALLOWED
69	BAD DATABASE (if database does not close normally and AUTODEFER is active)
71	LOGGING NOT ENABLED FOR USER.
72	TURBOLKT TABLE FULL.
73	ERROR IN TURBLKT TABLE OPERATION.

Table 5-10. DBERROR Messages

Return Status	DBERROR Message
1*nn*	NO CHAIN HEAD (MASTER ENTRY) FOR PATH *decimal integer: nn*
2*nn*	FULL CHAIN FOR PATH *decimal integer: nn* (CONTAINS 2,147,483,647 ENTRIES)
3*nn*	FULL AUTOMATIC MASTER FOR PATH *decimal integer: nn*
4*nn*	FULL AUTOMATIC MASTER SYNONYM CHAIN FOR PATH *decimal integer: nn*
944	WARNING: ASSUMING NO MESSAGE CATALOG
3000—3999	ERRORS RETURNED BY THIRD-PARTY INDEXING PRODUCTS
Others	UNRECOGNIZED RETURN STATUS: *decimal integer*

DBEXPLAIN

INTRINSIC NUMBER 418

Prints a multiline message on the $STDLIST device which describes a TurboIMAGE/XL 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/XL procedure call about which information is requested.

NOTE The call to DBEXPLAIN must be made immediately after receiving an error status before any other intrinsics are executed to ensure the display of valid messages.

Discussion

Table 5-11. 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/XL 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/XL 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 10 status elements to return a return status, lengths, and record numbers.

If the status array contains an unrecognized error code, the second line 4 choice is printed. If the return status is greater than or equal to zero, the word ERROR in line 2 is replaced by RESULT because non-negative return statuses indicate success or exceptional conditions, such as end-of-chain. Return status 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/XL library procedure call and contains valid offset information.

Table 5-11. DBEXPLAIN Message Format

Line	Format
1	(a blank line)
2	TurboIMAGE{ ERROR RESULT } [AT *offset*] *RETURN STATUS=retstat*
3	{ *intrinsicname*,MODE *x*,ON [*setname* OF] *basename* [;PASSWORD=*password*] TurboIMAGE CALL INFORMATION NOT AVAILABLE }
4	{ *message* UNRECOGNIZED RETURN STATUS: *retstat* }
5	[HEX DUMP OF STATUS ARRAY FOLLOWS OCTAL DUMP OF STATUS ARRAY FOLLOWS]
6	[*hex display* *octal display*]
7	(a blank line)

Parameter	Explanation
offset	The code offset of the TurboIMAGE/XL procedure call in a CM procedure. It is the virtual memory address of the TurboIMAGE/XL procedure call in a NM procedure.
retstat	The return status (from the first element of *status*) printed as a decimal integer and corresponding to the return statuses described in appendix A.
intrinsicname	The name of the TurboIMAGE/XL library procedure (intrinsic) which was called and which sets the contents of the *status* array.
x	The value of the *mode* parameter as a decimal integer.
setname	The value of the second parameter, usually a data set name or number, as passed to the procedure which set the *status* 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 DBCLOSE, *setname* is omitted.
password	The word printed at the end of line 3 only if the error relates to the *password* parameter of DBOPEN.
basename	The database specified in the procedure which was called and set the *status* array contents.
message	A description of the result based on the condition word and other *status* array information. The message is generated by the DBERROR procedure which is also described in this chapter. See Table 5-9 for all possible messages returned in line 4.
hex display	A listing of each halfword of *status* printed as a string of 4 hex digits. Adjacent *status* elements are separated by a blank and the entire line is 49 characters long. The hex display is generated for NM applications only.
octal display	A listing of each halfword of *status* printed as a string of 6 octal digits. Adjacent *status* elements are separated by a blank and the entire line is 49 characters long. The octal display is generated for CM applications only.

Figure 5-1. contains four examples of messages generated by DBEXPLAIN for a Native Mode

application.

Figure 5-1. Sample DBEXPLAIN Messages

```
TURBOIMAGE RESULT AT $0001d76c: RETURN STATUS=0 DBOPEN=intrinsic name
DBOPEN,MODE1, ON ORDERS                         ORDERS=database name
SUCCESSFUL EXECUTION - NO ERROR                 NO ERROR=message

TURBOIMAGE ERROR AT $0001d76c: RETURN STATUS=-12
DBPUT,MODE1, ON DATE-MASTER OF ORDERS           DATE-MASTER=data set name
DBPUT CALLED WITHOUT COVERING LOCK IN EFFECT

TURBOIMAGE RESULT AT $0001d76c: RETURN STATUS=16
DBPUT,MODE1, ON #1 OF ORDERS                    #1=data set number
THE DATA SET IS FULL

TURBOIMAGE RESULT: RETURN STATUS=4792
TURBOIMAGE CALL INFORMATION NOT AVAILABLE
UNRECOGNIZED RETURN STATUS: 4792
HEX DUMP OF STATUS ARRAY FOLLOWS:
12b8 0040 0c63 ff82 4d33 02a7 32e8 0000 0000 0000
...............hex display...................
```

Because the application is in Native Mode, the display is in hex. For Compatibility Mode applications, the display is in octal.

DBFIND

INTRINSIC NUMBER 404

Can be used both when B-Tree index exists on a key item of the master and when B-Tree index does not exist. For a detailed discussion on B-Trees, refer to chapter 11, "B-Tree Indices."

In the absence of a B-Tree index, or when the B-Tree index exists but the BTREEMODE1 flag is off, locates related master set key item 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 are located on the basis of a specified search item and its value.

When a B-Tree index exists for the master (explicit), DBFIND can be used for B-Tree searches both for the master data set as well as its corresponding detail data sets (implicit). The *dset* parameter determines if the DBFIND is for the master or one of its corresponding detail data sets. A B-Tree DBFIND using mode 1 on binary items (not X or U) will be treated as non-B-Tree search regardless of the presence of a B-Tree index as well as the BTREEMODE1 option. To do B-Tree index searches on binary items, use modes 4 or 24 in conjunction with a structured argument.

Syntax

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

Parameters

base is the name of the array used as the *base* parameter when opening the database. The first element 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 master or the detail data set to be accessed, or is an integer referencing the data set by number. The data set name can be up to 16 characters long. If shorter, it must be terminated by a semicolon or a blank.

mode must be an integer equal to 1, 4, 10, 21, or 24. If your database is enabled for third-party indexing, refer to your vendor's documentation for additional DBFIND modes.

Mode	Method
1	Used for both B-Tree or non-B-Tree DBFIND. Perform a B-Tree DBFIND if ALL of the following conditions prevail: • Item type is X or U (ASCII). • DBFIND is for master and a B-Tree index exists on the key item or DBFIND is for detail and B-Tree index exists

on the key item of the master to which the search item has a path.

- BTREEMODE1 flag is set ON by DBUTIL or DBCONTROL mode 15.
- Argument contains a wildcard character.

The chain count is accurate in halfword 5-6, and status halfwords 7-8 and 9-10 record the last entry in the last chain, and the first entry in the first chain of the super-chain for detail data sets. For master data sets, the status halfwords 5-6 (chain-count for detail) reflect the total number of entries qualified in the master data set. All other fields of the status array return zeroes.

4 — For B-Tree searches on numeric as well as ASCII types. For detail data sets, returns accurate chain (super-chain) counts and record numbers of last entry in last chain and first entry in first chain. For masters, returns the total number of qualified entries in status halfword 5-6. Other fields return zeroes. Requires a structured argument format described under *argument*.

10 — Allows you to simulate the current DBFIND mode 1, as in versions prior to B-Trees, as if there were no B-Tree index, even when the item has B-Tree index, BTREEMODE1 is on, and the argument contains a wildcard character. It returns accurate chain count. This is the same as TPI mode 10. Requires a simple argument.

21 — Same as mode 1, except it is a faster version and does not return accurate chain count or record numbers of first entry and last entries. Requires a simple argument.

For details, the chain count and status halfwords 7-8 and 9-10 will have 2^{31}–1. For master data sets, the status halfwords 5-6 (chain-count for detail) reflect the total number of entries qualified in the master data set. All other fields of the status array return zeroes. A DBFIND mode 21 on a non-ASCII key returns an error.

24 — Same as mode 4, except it is a faster version and does not return accurate chain counts and record numbers of last entry and first entry in super-chain. For detail data set, the halfwords 5-6, 7-8, and 9-10 will have 2^{31}–1. For master data set, the halfword 5-6 will have 2^{31}–1 and 7-8 and 9-10 will have zeroes. The argument is in structured format described under *argument*.

The third-party indexing (TPI) modes of DBFIND which are NOT supported for B-Trees are: 11, 12, and various TPI DBFIND modes 1nn, 2nn, 3nn, 4nn, and 5nn. If your database is enabled for TPI, refer to your vendor

documentation for additional DBFIND modes. The section on DBUTIL in chapter 8 of this book has a brief description of the TPI option.

status is the name of an array of 10 halfwords in which TurboIMAGE/XL returns status information about the procedure. If the procedure executes successfully, the status array contents are:

Element	Contents
1	If the procedure succeeds, the return status is 0. Table 5-11. describes the contents of element 1 when the procedure does not succeed.
2	Zero.
3-4	Word current record number set to zero.
5-6	Word count of number of entries in chain or qualifier. For modes 21 and 24, 2^{31}–1.
7-8	Word record number of last entry in chain. For modes 21 and 24, 2^{31}–1 for details. Always zeroes for masters.
9-10	Word record number of first entry in chain. For modes 21 and 24, 2^{31}–1 for details. Always zeroes for masters.

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 specified search item defines the path to which the chain belongs. For a B-Tree DBFIND on a master, it is the key item name or a number. The name can be up to 16 characters long. If shorter, it must be terminated by a semicolon or a blank.

argument contains a value for the key item to be used in calculated access to locate the desired chain head in the master data set for non-B-Tree index searches. This is referred to as a **simple argument**. For B-Tree index searches, there are two formats. One is for mode 1 wildcard searches on X and U types and allows a trailing wildcard character. The argument is scanned for the first occurrence of the wildcard character, for example, an @ character. Subsequent text in the item value is ignored.

The other is for modes 4 and 24, which requires a **structured format** as follows:

Bytes	Meaning
1-2	Type of generic search. An ASCII character pair is in this field:
	=␣* search for key values equal to argument1 (similar to a DBFIND mode 1)
	<␣* search for key values less than argument1
	<= search for key values less than or equal to argument1
	>␣* search for key values greater than argument1
	>= search for key values greater than or equal to argument1
	[ ] search for key values greater than or equal to argument1 AND less than or equal to argument2
	@c wildcard search when the wildcard character is *c*. Scan argument for the first wildcard character *c*. If a wildcard character is found in position *n*, 1-based, search for keys that match first *n*–1 characters of argument. If *c* is non-blank and non-null, then it is the wildcard character that will be used. Some examples are: @* and @@. If *c* is a blank or null, then the one set by DBCONTROL mode 15, if any, is used. Otherwise, the current default wildcard character (stored in the root file) is used. The wildcard character is changeable via the DBUTIL SET command or DBCONTROL mode 15.
	PK Partial Key search. Search for key values that match *n* characters in argument1 (*n* is length of argument1 provided in bytes 4 and 5 described later). Argument1 need not contain a wildcard. If it does within the *n* characters, it will be considered as one character part of the search value. For example, if n is 4, argument1 is ABC@, and the wildcard to be used is @, DBFIND will return records containing ABC@ as the first four characters in the key value.

* ␣ indicates a space character.

3-4	version number. Must be numeric zero, or an error is returned.
5-6	The size (in bytes) of argument1 (not including these two bytes) for search types =, ␣<, ␣<=, >, ␣>=, @*c*, PK.
7-8	The size (in bytes) of argument2 (not including these two bytes) for the search type [ ]. Must be numeric zero for other search types, or an error is returned.
9... 9+n-1	Argument1. The *n* bytes of argument data (for example, for an X10 field, *n* = 10).
9+n... 9+n+m-1	Argument2. For search-type [ ] only. The *m* bytes of the second argument's data (for example, for an X10 field, *m* = 10, *n* must match *m*).

If a wildcard character is present in the argument(s), the wildcard will be considered as part of the value for these B-Tree search types: =␣, <␣, <=, >␣, >=, [ ], or PK on ASCII types.

If your database is enabled for third-party indexing (TPI), refer to your vendor documentation for information on DBFIND arguments. The section on DBUTIL in chapter 8 of this book has a brief description of the TPI option.

Discussion

When the DBFIND is not for a B-Tree index, 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 chapter 10 for further information about chain heads and internally maintained data set information.

Although a master set entry with the specified key item value exists, the data set chain may be empty.

For B-Tree DBFINDs, there could be multiple qualified chain heads. The chained access could traverse multiple chains, referred to as **super-chain**. When using modes 1 or 4 for B-Tree DBFINDs, the qualified chain heads are retrieved, and the chain count becomes a cumulative chain count of the super-chain. The record number of last entry is obtained from the last entry of the last chain and the record number of the first entry is obtained from the first entry of the first chain. For modes 21 and 24, the qualified chain-heads are retrieved only during DBGETs, and hence, inaccurate chain counts and record numbers of last entry and first entry are returned for DBFIND.

For B-Tree DBFINDs:

- Can be used both for master data set and detail data sets.
- Allows wildcard search, relational operators, as well as range search.
- DBFIND mode 1 will result in a B-Tree wildcard search, if all of the following are true:

— Item type is X or U (ASCII).

— DBFIND is for master and a B-Tree index exists on the key item or DBFIND is for detail and B-Tree index exists on the key item of the master to which the search item has a path.

— BTREEMDODE1 flag is set ON by DBUTIL or DBCONTROL mode 15.

— Argument contains a wildcard character.

- Status array reflects information based on set, mode, and search type. B-Tree searches for modes 1 and 4 for details give super-chain (multiple detail chain) counts and record numbers of the first entry in first detail chain and last entry in last detail chain. For masters, modes 1 and 4 give a count of entries qualified in the master and zeroes for the first and last qualified entries in the master.

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, as well as the B-Tree index file (if applicable). This causes extra overhead not incurred by subsequent calls to the same data set by DBFIND or DBGET.

Consult appendix A for more information about the conditions for the DBFIND return status values shown in Table 5-12.

Table 5-12. DBFIND Return Status Values

File System, Memory Management, and Transaction Management Failures:	-1	FOPEN failure.
	-3	FREADDIR failure.
	-4	FREADLABEL failure.
	-168	Cannot attach *n* to MPE XL XM: file system error *nn*.
	-169	Invalid mode for XM attach options.
	-175	Cannot attach *n* to MPE XL XM: XM error *nn*.
	-176	Cannot detach *n* from MPE XL XM: XM error *nn*.
	-178	Cannot detach *n* from MPE XL XM: file system error *nn*.
	-209	Invalid mode for XM detach options.

Table 5-12. DBFIND Return Status Values

Calling Errors:	-11	Bad database reference.
	-21	Bad data set reference.
	-31	Bad mode.
	-51	Bad list length.
	-52	Bad item.
	-222	Only DBXUNDO allowed when a dynamic transaction encounters an error.
	-258	Invalid argument for index.
	-259	Invalid mode for index.
	-260	No previous list of qualified data entries.
	-426	BTE: B-Tree doesn't exist.
	-429	BTE: DBFIND argument version is bad.
	-430	BTE: DBFIND (mode 4/24) argument type is bad.
	-431	BTE: DBFIND (mode 4/24) argument #1 length is bad.
	-432	BTE: Bad wildcard character.
	-433	BTE: DBFIND (mode 4/24) argument #2 length is bad.
	-434	BTE: Data set detail instead of master.
	-436	BTE: Failed to extract data from root file.
	-437	BTE: Failed to convert @c to dbfind.
	-438	BTE: Invalid item number.
	-439	BTE: Conversion error.
	-444	BTE: DBFIND on non-key field of master.
	-446	BTE: Argument 2 specified for relop of (</<=/=/>=/>).
	-449	BTE: Failed to position index at start of key range.
Communications Errors:	-102	DSWRITE failure.
	-106	Remote 3000 data inconsistent.
	-107	NS 3000 or DS 3000 system error.
Exceptional Conditions:	-193	DBU control block is full.
	-3*nn*	Internal error.
	-314	Error in patch information for set.
	17	No master entry.
	63	DBG disabled; potential damage; only DBCLOSE allowed.

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 database. The first element 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 can be up to 16 characters long. If shorter, it must be terminated by a semicolon or a 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 number (*argument* parameter is ignored).
2	**Serial Read.** Read the first entry whose record number is greater than the internally maintained current number (*argument* parameter is ignored).
3	**Backward Serial Read.** Read the first entry whose record number is less than the internally maintained current number (*argument* parameter is ignored).
4	**Directed Read.** Read the entry, if it exists, at the record number specified in the *argument* parameter (*argument* is treated as a 32-bit record number).
5	**Chained Read or Next Qualified Entry Read.** Read the next entry in the current chain, or read the next qualified entry for a B-Tree DBFIND. This is the entry referenced by the internally maintained forward pointer (*argument* parameter is ignored). Super-chains are traversed for detail data sets. If your database is enabled for third-party indexing (TPI), refer to your vendor documentation for additional information.

6	**Backward Chained Read.** Read the previous entry in the current chain, or the previous qualified entry for a B-Tree DBFIND. This is the entry referenced by the internally maintained backward pointer (*argument* parameter is ignored). Super-chains are traversed for detail data sets. If your database is enabled for third-party indexing (TPI), refer to your vendor documentation for additional information.
7	**Calculated Read.** (Master data sets only.) Read the entry with a key item value that matches the value specified in *argument*. The entry is in the master data set specified by *dset*.
8	**Primary Calculated Read.** (Master data sets only.) Read the entry occupying the primary address of a synonym chain using the key item value specified in *argument* to locate the entry. The key item value returned is always that of the primary entry and might not match the value specified in *argument*. (Refer to chapter 10 for synonym chain description.)

If your database is enabled for third-party indexing (TPI), refer to your vendor documentation for additional DBGET modes. The section on DBUTIL in chapter 8 of this book has a brief description of the TPI option.

status is the name of a 10-halfword array in which TurboIMAGE/XL returns status information about the procedure. If the procedure executes successfully, the status array contents are:

Element	Contents
1	If the procedure succeeds, the return status is 0. Table 5-12. describes the contents of element 1 when the procedure does not succeed.
2	Length of the logical entry read into the buffer array in halfwords.
3-4	Word record number of the data entry read.
5-6	Word zero, unless the entry read is a primary entry in which case it is the number of entries in the synonym chain.
7-8	Word record number of the preceding entry in the chain of the current path for the detail data sets. Zeroes for master data sets.
9-10	Word record number of the next entry in the chain of the current path for the detail data sets. Zeroes for master data sets.

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.

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.

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

The *list* not only specifies the data items to be retrieved immediately but is saved internally by TurboIMAGE/XL 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 can 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/XL will assume a null list. If a DBCLOSE mode 2 is used after DBGET, using the asterisk construct, TurboIMAGE/XL uses the previously defined item 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 elements 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 word record number of the entry to be read.

If *mode* is 7 or 8, *argument* contains a key 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, and not a B-Tree index, they are synonym chain pointers (refer to chapter 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. DBGET mode 5 or 6 will reread the current record and will try to continue the chain read if it encounters a broken chain.

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

Table 5-13. DBGET Return Status Values

File System, Memory Management, and Transaction Management Failures:	-1	FOPEN failure.
	-3	FREADDIR failure.
	-4	FREADLABEL failure.
	-168	Cannot attach *n* to MPE XL XM: file system error *nn*.
	-169	Invalid mode for XM attach options.
	-175	Cannot attach *n* to MPE XL XM: XM error *nn*.
	-176	Cannot detach *n* from MPE XL XM: XM error *nn*.
	-178	Cannot detach *n* from MPE XL XM: file system error *nn*.
	-209	Invalid mode for XM detach options.
Calling Errors:	-11	Bad database reference.
	-21	Bad data set reference.
	-31	Bad mode.
	-51	Bad list length.
	-52	Bad list or bad item.
	-222	Only DBXUNDO allowed when a dynamic transaction encounters an error.
Communications Errors:	-102	DSWRITE failure.
	-106	Remote 3000 data inconsistent.
	-107	NS 3000 or DS 3000 system error.

Table 5-13. DBGET Return Status Values

Exceptional Conditions:			DBGET Modes
	10	Beginning of file.	(3)
	11	End of file.	(2)
	12	Directed beginning of file.	(4)
	13	Directed end of file.	(4)
	14	Beginning of chain/qualifier entries.	(6)
	15	End of chain/qualifier entries.	(5)
	17	No entry.	(1,2,7,8)
	18	Broken chain.	(5,6)
	49	Illegal buffer address.	
	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	DBG disabled; potential damage; only DBCLOSE allowed.	
	-193	DBU control block is full.	
	-3*nn*	Internal error.	
	-332	Error in QLOCK table operation.	
	-333	Error in QOPEN table operation.	

Consult appendix A for more information about these conditions.

DBINFO

INTRINSIC NUMBER 402

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

Syntax

DBINFO,*base*,*qualifier*,*mode*,*status*,*buffer*

Parameters

base is the array name used as the *base* parameter when opening the database; must contain the base ID returned by DBOPEN. (Refer to DBOPEN for additional base ID information.)

qualifier 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 *mode* parameter (refer to "Discussion" for *mode/qualifier* relationship). This parameter form is identical to the *dset* and *item* parameters for DBFIND, and the *dset* and *list* parameters for DBPUT.

mode is an integer indicating the type of information desired. Refer to "Discussion" for *mode* integer information (data item modes 1*nn*; data set modes 2*nn*; path modes 3*nn*; logging, dynamic roll-back recovery, and multiple database transaction modes 4*nn*; subsystem and critical item update modes 5*nn*; third-party indexing modes 8*nn*; and language modes 9*nn*.)

status is the name of an array of 10 halfwords in which TurboIMAGE/XL returns status information about the procedure. If the procedure executes successfully, the status array contents are:

Element	Contents
1	If the procedure succeeds, the return status is 0. Table 5-14. describes the contents of element 1 when the procedure does not succeed.
2	Length of information in buffer array (in halfwords).
3-4	Unchanged from previous procedure call using this array.
5-10	Information about the procedure call and its results. Refer to "Library Procedure Error Messages" in appendix A for a description of this information.

buffer is the name of an array in which the requested information is returned. The contents of the *buffer* array vary according to the mode parameter

used. They are also described in "Discussion" on the following pages.

Table 5-14. DBINFO Return Status Values

File System, Memory Management, and Transaction Management Failures:	-1	FOPEN failure.
	-4	FREADLABEL failure.
	-168	Cannot attach *n* to MPE XL XM: file system error *nn*.
	-169	Invalid mode for XM attach options.
	-175	Cannot attach *n* to MPE XL XM: XM error *nn*.
	-176	Cannot detach *n* from MPE XL XM: XM error *nn*.
	-178	Cannot detach *n* from MPE XL XM: file system error *nn*.
	-209	Invalid mode for XM detach options.
Calling Errors:	-11	Bad database reference.
	-21	Bad data set reference.
	-31	Bad mode.
	-222	Only DBXUNDO allowed when a dynamic transaction encounters an error.
Communications Errors:	-102	DSWRITE failure.
	-106	Remote 3000 data inconsistent.
	-107	NS 3000 or DS 3000 system error.
	-206	Remote TurboIMAGE/XL database exceeds IMAGE/3000 limits.
Exceptional Conditions:	49	Illegal buffer address.
	50	Buffer too small for requested data.
	63	DBG disabled; potential damage; only DBCLOSE allowed.

Consult appendix A for more information about these conditions.

Discussion

This section provides mode integer information for the following modes (note that the *mode/qualifier* relationship is provided):

1*nn* – Data item modes

2*nn* – Data set modes

3*nn* – Path modes

4*nn* – Logging, dynamic roll-back recovery, and multiple database transaction modes

5*nn* – Subsystem and critical item update modes

8*nn* – Third-party indexing modes

9*nn* – Language modes

Mode 101: Item Number

Mode 101 defines the type of access available for a specific item.

Qualifier identifies the data item name or number for which the information is requested.

Buffer returns the following (each element is a halfword or two bytes):

Element	Contents
1	+/– Data item number

If the data item number is positive, the user class has only read access to the data item. If the number is negative, the data item can be updated or the entry containing it can be added or deleted in at least one data set.

Mode 102: Item Name

Mode 102 describes a specific data item.

Qualifier identifies the data item name or number for which the information is requested.

Buffer returns the following (each element is a halfword or two bytes):

Element	Contents
1–8	Data item name
9	One of the following data types followed by a blank: I, J, K, R, U, X, Z, P
10	Sub-item length
11	Sub-item count
12	0
13	0

The data item name is left-justified and will be padded with blanks if the name is shorter than 16 characters.

Mode 103: Items in Database

Mode 103 identifies data items available in the database and displays the type of access allowed. This mode does **not** identify unreferenced data items, that is, those items that are defined in the item section of the schema but are not referenced by at least one data set.

Qualifier is ignored.

Buffer returns the following (each element is a halfword or two bytes):

Element	Contents
1	Item count *x*
2	+/– Data item number 1
. . .	. . .
n + 1	+/– Data item number n

If the data item number is positive, the user class has only read access to the data item. If the number is negative, the user class has both read and write access to the given data set. The data items are listed in data item number order.

Mode 104: Items in Data Set

Mode 104 identifies data items available in a specific data set and the type of access allowed.

Qualifier identifies the data set name or number for which the information is requested.

Buffer returns the following (each element is a halfword or two bytes):

Element	Contents
1	Item count *x*
2	+/– Data item number 1
. . .	. . .
n + 1	+/– Data item number n

If the data item number is positive, the user class has only read access to the data item. If the number is negative, the user class has both read and write access in the given data set. The data items are listed in order of occurrence in data entry.

Mode 113: BTREEMODE1 and Wildcard Character

Mode 113 gives the settings of BTREEMODE1 and wildcard character in the root file as well as current DBOPEN (DBU).

Qualifier is ignored.

Buffer must be at least 32 bytes and returns the following (each element is a halfword or 2 bytes):

Element	Contents
1	0 if BTREEMODE1 is off in the root file 1 if BTREEMODE1 is on in the root file
2	The first byte (8 bits) is always 0. The second byte (8 bits) represents *c*, where *c* is the current wildcard character. For example, if the current wildcard character is @, the element's hex value will be $0040, or decimal 64. This is from the root file.
3	Highest B-Tree argument version supported (0 for the first B-Tree release version).
4	Number of sets with B-Trees indices attached.
5	0 if BTREEMODE1 is off for current DBOPEN 1 if BTREEMODE1 is on for current DBOPEN
6	The first byte (8 bits) is always 0. The second byte (8 bits) represents *c*, where *c* is the current wildcard character. This is for the current DBOPEN.
7...16	(reserved)

Mode 201: Set Number

Mode 201 defines the type of access available for a specific data set.

Qualifier identifies the data set name or number for which the information is requested.

Buffer returns the following (each element is a halfword or two bytes):

Element	Contents
1	+/– Data set number

If the data set number is positive, the user class has only read access. If the number is negative, the user class has both read and write access.

Mode 202: Set Name

Mode 202 describes a specific data set.

Qualifier identifies the data set name or number for which the information is requested.

Buffer returns the following (each element is a halfword or two bytes):

Element	Contents
1–8	Data set name
9	One of the following set types followed by a blank: M, A, D
10	Entry length
11	Blocking factor
12	0
13	0
14–15	Number of entries in set
16–17	Capacity of set. If data set is dynamically expandable, current capacity for detail, and initial capacity (also primary capacity) for master. Note that for master, it is always primary capacity regardless of expansion.

The data set name is left-justified and will be padded with blanks if the name is shorter than 16 characters.

Mode 203: Sets in Database

Mode 203 identifies all data sets available in a database and the type of access allowed. If you are using third-party indexing, this mode does *not* show third-party index files.

Qualifier is ignored.

Buffer returns the following (each element is a halfword or two bytes):

Element	Contents
1	Set count *x*
2	+/– Data set number 1
. . .	. . .
n + 1	+/– Data set number n

If the data set number is positive, the user class has only read access to the data set and possibly is in the write class list of specific data item(s). If the number is negative, the user class has both read and write access. The data sets are listed in data set number order.

Mode 204: Sets with Item

Mode 204 identifies all data sets available which contain a specified data item and indicates the type of access allowed.

Qualifier identifies the data item name or number for which the information is requested.

Buffer returns the following (each element is a halfword or two bytes):

Element	Contents
1	Set count *x*
2	+/– Data set number
. . .	. . .
n + 1	+/– Data set number

If the data set number is positive, the user class has only read access to the data set and possibly is in the write class list of specific data item(s). If the number is negative, the user class has both read and write access. The data sets are listed in data set number order.

Mode 205: Set Capacity

Mode 205 is an extension of mode 202 with dynamic capacity expansion information.

Qualifier identifies the data set name or number for which the information is requested.

Buffer returns the following (each element is a halfword or two bytes):

Element	Contents
1-8	Data set name
9	One of the following set types followed by a blank: M, A, D
10	Entry length
11	Blocking factor
12	0
13	0
14-15	Number of entries in set
16-17	Current capacity of set, including expansions
18-19	High-water mark
20-21	Maximum capacity
22-23	Initial capacity (primary capacity for master)

Element	Contents
24-25	Incremental number of entries
26	Incremental percent
27	Dynamic capacity expansion flag (0 = off, 1 = on)

The data set name is left-justified and will be padded with blanks if the name is shorter than 16 characters.

Mode 205 can be used for any master or detail data set with or without dynamic capacity expansion capability.

Mode 205 allows users to obtain information on dynamic data set capacity expansion programmatically. It is an extension of mode 202 to include dynamic capacity expansion information such as maximum capacity, initial capacity, incremental number of entries, incremental percent, and the dynamic capacity expansion flag (0 for off and 1 for on) for the data set.

Mode 206: Number of Data Set Chunks

Mode 206 gives the number of chunks in a data set in short format.

Qualifier identifies the data set name or number for which the information is requested.

Buffer returns the following (each element is a halfword or two bytes):

Element	Contents
1	# of chunks in a jumbo data set

If the data set is not a jumbo data set, zero is returned for the number of chunks.

Mode 207: Size of Data Set Chunks

Mode 207 identifies the size of each chunk in terms of IMAGE records in addition to providing the number of chunks.

Qualifier identifies the data set name or number for which the information is requested.

Buffer returns the following (each element is a halfword or two bytes):

Element	Contents
1	# of chunks in a jumbo data set
2	0
3-4	Size of chunk 1 (# entries, not # of blocks!)
5-6	Size of chunk 2 (# entries, not # of blocks!)
7-8	. . .
2n + 3	Size of chunk n (# entries, not # of blocks!)

Total size: (n + 1) * 4 bytes.

If the data set is not a jumbo data set, then zero is returned for the number of chunks.

Mode 208: Primary and Actual Capacity

Mode 208 returns the primary and actual capacity.

Qualifier identifies the data set name or number for which the information is requested.

Buffer must be at least a 64-byte record and returns the following (each element is a 32-bit Word):

Element	Contents
1	Primary (hashing) capacity for masters, 0 for details
2	Current capacity, including expansions
3	Maximum capacity
4	Expansion threshold: -1...100 percentage -1 this value means expansion is not triggered by percentage, or it is a non-expandable set
5	Delete chain free head (0 for non-expanded masters)
6	high-water mark (0 for non-expanded masters)
7	Expansion threshold: -1...2 billion blocks -1 this value means expansion not triggered by traversing # blocks without success, or it is a non-expandable set
8...16	Reserved; 0 is returned

DBINFO mode 208 does not return an expandable flag, while DBINFO 205 returns an expandable flag. The flag can also be deduced by comparing the current and maximum

capacities in DBINFO 208.

DBINFO mode 208 returns information about internals which will be meaningful to only a few customers.

Mode 209: B-Tree Attachment

Mode 209 informs whether or not a B-Tree exists for a master.

Qualifier is a master data set name or number.

Buffer must be at least a 64-byte record and returns the following (each element is a 32-bit Word):

Element	Contents
1	0 if no B-Tree index exists 1 if B-Tree index exists
2	0 if attached B-Tree not damaged or index does not exist 1 if the attached B-Tree index is damaged
3...32	For internal use

Mode 301: Paths

Mode 301 identifies the paths defined for a specified data set.

Qualifier identifies the data set name or number for which the information is requested.

Buffer returns the following (each element is a halfword or two bytes):

Element	Contents
1	Path count *x*
2	Data set number of path 1
3	Search item number of path 1
4	Sort item number of path 1
. . .	. . .
3*n*–1	Data set number of path *n*
3*n*	Search item number of path *n*
3*n*+1	Sort item number of path *n*

Elements 2 to 4 are repeated for each path.

If *qualifier* specifies a master data set, the set number identifies the detail data set.

If *qualifier* specifies a detail data set, the set number identifies the master data set.

If *qualifier* specifies a master data set, the item numbers identify items in the detail data sets. If you do not have access to a search item, it is not included in the path count and the path information is not returned. If a sort item does not exist or you do not have access to it, the sort item number is zero.

Path designators are presented in the order in which they appear in the schema.

Mode 302: Key or Search Item

Mode 302 identifies the key or search item for a specified data set. For this mode the two qualifiers are shown separately.

Qualifier identifies the master data set name or number for which the information is requested.

Buffer returns the following (each element is a halfword or two bytes):

Element	Contents
1	Key item number
2	0

If *qualifier* specifies a master data set, the key item number is the number in the master set. The number is 0 if you do not have access to the key item.

OR

Qualifier identifies the detail data set name or number for which the information is requested.

Buffer returns the following (each element is a halfword or two bytes):

Element	Contents
1	Search item number
2	Master data set number

If *qualifier* specifies a detail data set, the primary search item and the related master data set number are returned. Both numbers are 0 if you do not have access to the search item.

Mode 401: Logging

Mode 401 obtains information related to logging.

Qualifier is ignored.

Buffer returns the following (each element is a halfword or two bytes):

Element	Contents
1–4	Log identifier name
5	Database log flag
6	User log flag
7	Transaction flag
8–9	User transaction number

The log identifier name is left-justified and padded with blanks if shorter than 8 characters.

If the database is enabled for logging, the database log flag is 1; otherwise it is 0.

If you are logging, the user log flag is 1; otherwise it is 0.

If you have a transaction in progress, the transaction flag is 1; otherwise it is 0.

The user transaction number is one word.

Mode 402: ILR

Mode 402 returns information about Intrinsic Level Recovery (ILR).

Qualifier is ignored.

Buffer returns the following (each element is a halfword or two bytes):

Element	Contents
1	ILR log flag
2	Calendar date (*mmddyy*)
3–4	Clock time (one word in format *hhmmsstt*)
5	0
6–14	
15–16	Reserved

If the database is enabled for ILR, the ILR log flag is 1; otherwise it is 0.

The calendar date is the date ILR was enabled.

The clock time is the time ILR was enabled.

Element 5 is always 0.

ß indicates blank. Elements 6 to 14 are always blank.

Elements 15 and 16 are reserved.

Mode 403: Dynamic Roll-Back

Mode 403 obtains information related to dynamic transaction activity on a given database.

Qualifier is ignored.

Buffer returns the following (each element is a halfword or two bytes):

Element	Contents
1–4	Log identifier name
5	Database log flag
6	User log flag
7	Logical transaction flag
8–9	User transaction number
10–11	XM log set size (in megabytes)
12	XM log set type
13	Database attached flag
14	Dynamic transaction flag
15–26	XM log set name

The log identifier name is left-justified and padded with blanks if shorter than 8 characters.

If the database is enabled for logging, the database log flag is 1; otherwise it is 0.

If you are logging, the user log flag is 1; otherwise it is 0.

If no logical transaction is in progress, the transaction flag is 0. If a static transaction is in progress, the flag is set to 1; if a multiple database transaction is in progress, the flag is set to 2.

The user transaction number is one word.

The Transaction Management (XM) log set is measured in megabytes.

If the XM log set is circular, the log set type is CR; otherwise it is LN indicating a linear log.

If the database is attached to XM, the database attached flag is 1; otherwise it is 0.

If the user is processing a dynamic transaction, the dynamic transaction flag is 1; otherwise it is 0.

If the database is associated with the default XM user log set, the XM log set name element contains blanks; otherwise it contains the name of the XM log set.

Mode 404: Logging Subsystem Information

Mode 404 returns information about multiple database transactions.

Qualifier is ignored.

Buffer returns the following (each element is a halfword or two bytes):

Element	Contents
1	Database log flag
2	User log flag
3	Roll-back log flag
4	ILR log flag
5	MUSTRECOVER flag
6	Database remote flag
7	Logical transaction flag
8–11	Log identifier name
12–13	Log index
14–15	Multiple database transaction ID
16	Number of databases involved in the multiple database transaction
17–31	Base IDs of the databases involved in the multiple database transaction

If the database is enabled for logging, the database log flag is 1; otherwise it is 0.

If you are logging, the user log flag is set to 1; otherwise it is 0. If a user accesses the database with a DBOPEN mode 5, this flag is set to 0.

If the database is enabled for roll-back logging, the roll-back log flag is set to 1; otherwise it is 0.

If the database is enabled for ILR, the ILR log flag is set to 1; otherwise it is 0.

If the database is enabled for MUSTRECOVER, the MUSTRECOVER flag is set to 1; otherwise it is 0.

If the database resides on a remote system, the database remote flag is set to 1. If the database resides on the local system, the flag is set to 0.

If no logical transaction is in progress, the transaction flag is set to 0. If a static transaction is in progress, the flag is set to 1. If a multiple database transaction is in progress, the flag is set to 2.

The log index is used to call the WRITELOG intrinsic. It is set to 0 if logging is not used.

The transaction ID represents a multiple or single database transaction.

Elements 16-31 are set when a multiple database transaction is in progress.

Mode 406: Database Information

Mode 406 returns information about fully qualified database name and open mode.

Qualifier is ignored.

Buffer returns the following (each element is a halfword or two bytes):

Element	Contents
1-14	Fully qualified database name, left justified, blank trailing.
15	Open mode for current DBOPEN.
16	Root file version of the database. In the form C*n* where C is uppercase ASCII and *n* is a number. For example, C2 or C3.
17-24	For internal use.
25-32	Reserved.

Mode 501: Subsystem Access

Mode 501 checks subsystem access to the database. Refer to the DBUTIL >>SHOW and >>SET commands described in chapter 8 for more information.

Qualifier is ignored.

Buffer returns the following (each element is a halfword or two bytes):

Element	Contents
1	Subsystem access

The following values are used for subsystem access:

0 No access

1 Read access

3 Read/write access

Mode 502: Critical Item Update

Mode 502 checks the critical item update (CIUPDATE) option settings for the database and the current DBOPEN. The CIUPDATE option is set for the database with the DBUTIL >>SET command and then, depending on the setting, can be enabled or disabled with DBCONTROL for the current DBOPEN. Refer to the discussion of DBCONTROL in this chapter and the descriptions of the DBUTIL >>SHOW and >>SET commands in chapter 8 for more information.

Qualifier is ignored.

Buffer returns the following (each element is a halfword or two bytes):

Element	Contents
1	Critical item update flag
2	Current setting for accessor

For element 1, the following values are used for the CIUPDATE option setting specified with DBUTIL for the database:

0 Critical item update is disallowed.

1 Critical item update is allowed (default).

2 Critical item update is on.

For element 2, the following values are used for the current DBOPEN setting specified with DBCONTROL:

0 Critical item update is disabled for this accessor.

1 Critical item update is enabled for this accessor.

Modes 8nn: Third-Party Indexing

Modes 8*nn* are used to return information related to third-party indexing (TPI). If your database is enabled for TPI, refer to your vendor documentation for additional DBINFO mode information. The section on DBUTIL in chapter 8 of this book has a brief description of the TPI option.

Mode 901: Language

Mode 901 obtains the Native Language attribute of the database. It returns the MPE/iX code for the language attribute.

Qualifier is ignored.

Buffer returns the following (each element is a halfword or two bytes):

Element	Contents
1	Language ID

DBLOCK

INTRINSIC NUMBER 409

Applies a logical lock to a database, 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 database. The first element 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 "@", when applying a database 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 can also change.

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

NOTE If the database is open in user access mode 1, a lock must be in effect on either the data set or the whole database when adding to or deleting from master data sets. If a data entry level lock is specified, any subsequent DBPUTs or DBDELETEs will fail with error number –12 and the following message is returned:

intrinsic name CALLED WITHOUT COVERING LOCK IN EFFECT

Note, however, that a lock on either the entire database or data set can be achieved with a data entry lock when an @ sign is used to specify either all data sets or all data items.

status is the name of an array of 10 halfwords in which TurboIMAGE/XL returns status information about the procedure. If the procedure executes successfully, the status array contents are:

Element	Contents
1	If the procedure succeeds, the return status is 0. Table 5-17. describes the contents of element 1 when the procedure does not succeed.

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 the return status is 20, this element contains 0 if the database is locked, 1 if the data set or entries are locked.
4	Reserved: Contents undefined.
5-10	Information about the procedure call and its results. Refer to "Library Procedure Error Messages" in appendix A for a complete description of this information.

NOTE Concurrent processes running in a process-handling environment must have MR capability if they call 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-halfword 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 halfwords 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 halfwords long. The length of the entire descriptor array cannot exceed 4094 bytes.

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

Table 5-15. Locking Mode Options

Lock Mode	Lock Level	Locking Type	Description
1	Base	Unconditional	DBLOCK applies an unconditional lock to the whole database, returning to the calling program only after the lock is successful (or if an error occurs). The *qualifier* parameter is ignored.
2	Base	Conditional	DBLOCK applies a conditional lock to the database and returns immediately. A return status of zero indicates success. A non-zero return status indicates the reason for failure. (Refer to Table 5-17.)
3	Set	Unconditional	DBLOCK applies an unconditional lock to a data set. The *qualifier* 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 can be 16 characters long or, if shorter, terminated by a semicolon or blank.
			The data set need not be accessible for read or write access to the user requesting the lock.
4	Set	Conditional	DBLOCK applies a conditional lock of the same type as mode 3. It always returns to the calling program immediately. A return status of zero indicates success and a non-zero return status indicates a reason for failure. (Refer to Table 5-17.)
5	Entry	Unconditional	DBLOCK applies unconditional locks to the data entries specified by lock descriptors. The *qualifier* 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	DBLOCK applies conditional locks of the same type as mode 5. If multiple lock descriptors are specified and DBLOCK encounters a lock descriptor that it cannot apply, it returns. All locks that have been applied until that point are retained.
			Because 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 element 2 indicates how many lock descriptors were actually successful. It is recommended that a DBUNLOCK be issued after any unsuccessful mode 6 DBLOCK.

NOTE Be careful when changing modes. The *qualifier* parameter can change.

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

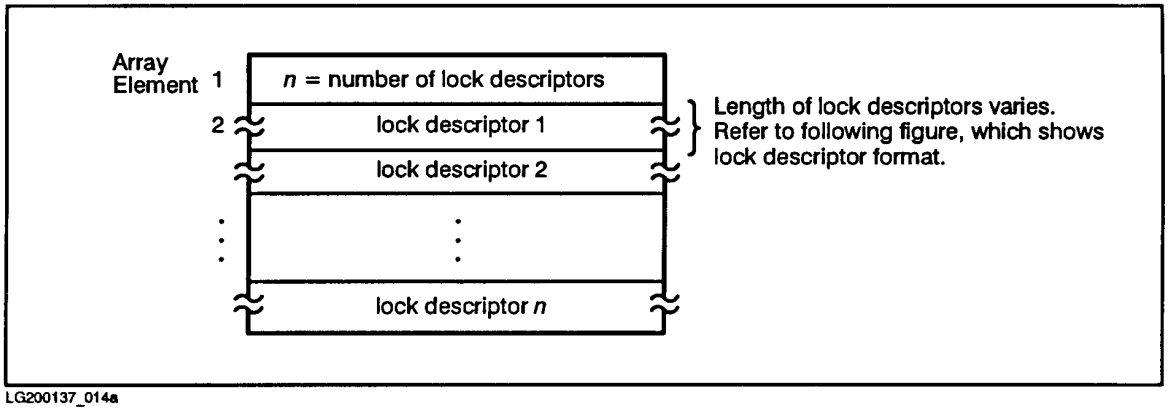


Figure 5-3. Lock Descriptor Format

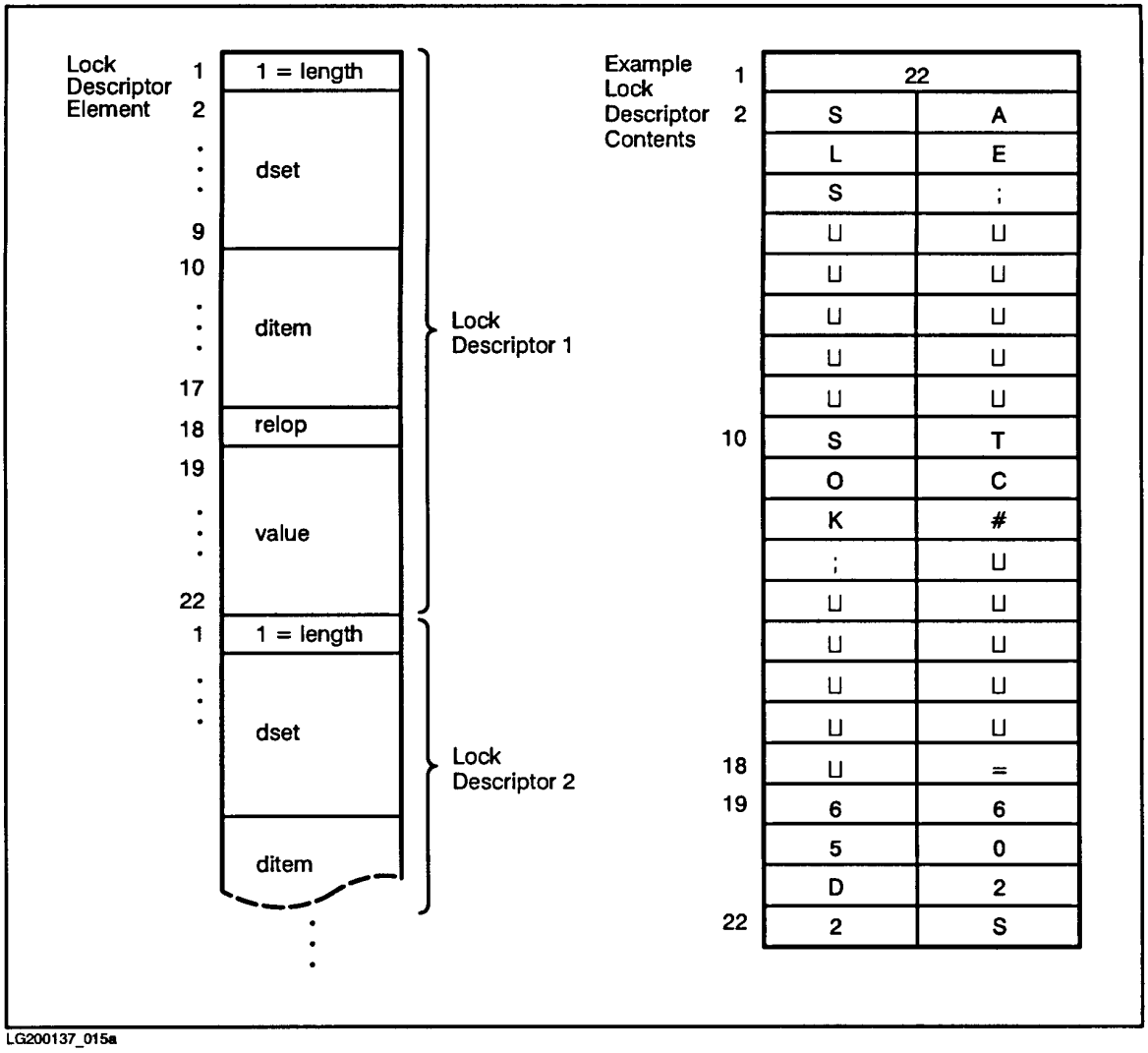


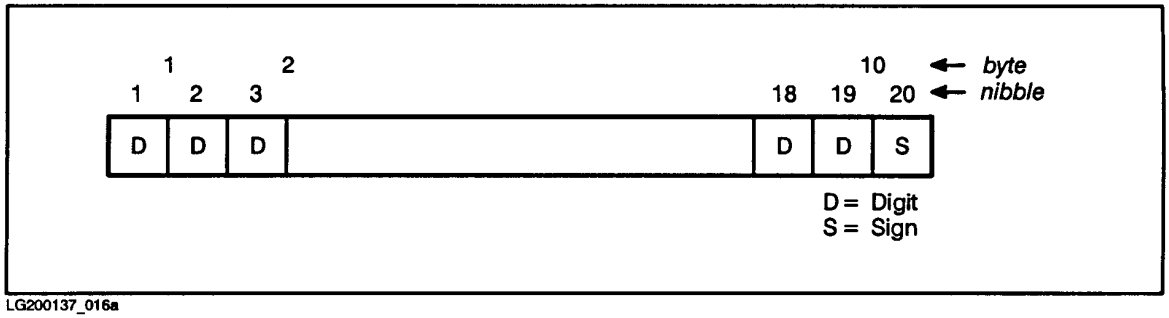
Table 5-16. Lock Descriptor Fields

Field Name	Description
length	A halfword integer specifying the physical length in halfwords of the lock descriptor, including the *length* field itself.
dset	Describes the data set in which locks are placed. It is always 8 halfwords long and can be one of the following: • A data set name, left-justified, 16 characters long or, if shorter, terminated with a blank or semicolon (for example, SALES;). • A data set number, an integer in the range of 1 to 199 stored in the first element. • An at-sign (@) stored in the first byte of the *dset* and a lock descriptor length of 2 indicating that the whole database is to be locked. All unused bytes are ignored. In this case, the *ditem*, *relop*, and *value* fields are ignored and can be omitted if desired. • A blank or semicolon (first byte) or binary zero (first halfword) indicating that the whole lock descriptor is to be ignored. (It is counted as one of the *n* descriptors.) The data set, if specified, need not be accessible for read or write access to the user requesting the lock.
ditem	Always 8 halfwords long unless an @ is stored in the first byte. It can be one of the following: • A data item name, left-justified, 16 characters long or, if shorter, terminated with a blank or semicolon. • A data item number stored as an integer in the first halfword. It can be in the range of 1 to 1023. • An at-sign (@) stored in the first byte of the *ditem* indicating that the whole data set specified in *dset* is to be locked. All unused bytes are ignored and can be omitted if desired. 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.
relop	One halfword long, it contains one of the three relational operators represented as two ASCII characters: <= less than or equal >= greater than or equal =␣ or ␣= equal (␣ indicates space character)
value	The value of the data item to be locked. It must be stored in exactly the same way as it is stored in the database. TurboIMAGE/XL extracts as many halfwords 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/XL 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 right most 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 as shown here:

Figure 5-4. Lock Descriptor Format



This would be declared in COBOL II 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 cannot exceed 28 nibbles (7 halfwords) 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 cannot 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 database is 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.
- If the data item is type R, it is sorted based on the HP 3000 floating point number format.

Table 5-17. DBLOCK Return Status Values

Calling Errors:	-11	Bad database reference.	
	-31	Bad mode value.	
	-121	Descriptor count error.	
	-123	Illegal relop in a descriptor.	
	-124	Descriptor too short. Must be greater than or equal to 9.	
	-125	Bad set name/number.	
	-126	Bad item name/number.	
	-127	Attempt to lock using a compound item.	
	-128	Value field too short in a descriptor.	
	-129	P-type item longer than P28 specified.	
	-130	Illegal digit in a P-type value.	
	-131	Lowercase character in type U value.	
	-132	Illegal digit in type Z value.	
	-133	Illegal sign in type Z value.	
	-134	Two descriptors conflict.	
	-135	DBLOCK called when locks already in effect.	
	-136	Descriptor list exceeds 4094 bytes.	
	-222	Only DBXUNDO allowed when a dynamic transaction encounters an error.	
Communications Errors:	-102	DSWRITE failure.	
	-103	Remote 3000 stack too small.	
	-106	Remote 3000 data inconsistent.	
	-107	NS 3000 or DS 3000 system error.	

Table 5-17. DBLOCK Return Status Values

Exceptional Conditions:			Applicable Modes
	20	Database locked or contains locks.	(2,4,6)
		(Status element 3: 0 = database locked 1 = data set or entries locked)	
	22	Data set locked by another process.	(3,4,5,6)
	23	Entries locked within set.	(4)
	24	Item conflicts with current locks.	(6)
	25	Entry or entries already locked.	(6)
	26	Lock not performed since deadlock would occur.	(1,2,3,4,5,6)
	62	DBG full. (If this error occurs when multiple lock descriptors are specified, some of the descriptors may have been successfully completed. If so, they are not unlocked by TurboIMAGE/XL before returning the error. Therefore, issue a DBUNLOCK after any positive-numbered error, unless you have reason to do otherwise.)	(5,6)
	63	DBG disabled; potential damage; only DBCLOSE allowed.	
	-192	Invalid DBU.	
	-241	Bad tag for TurboLKT table.	

Appendix A contains more information about these conditions.

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 database. The first element 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 512 bytes that contains user data (ASCII or binary) to be written to the log file as part of the DBMEMO log record.

mode must be an integer equal to 1.

status is the name of an array of 10 halfwords in which TurboIMAGE/XL returns status information. If the procedure executes successfully, the status array contents are:

Element	Contents
1	If the procedure succeeds, the return status is 0. Table 5-18. describes the contents of element 1 when the procedure does not succeed.
2-4	Unchanged from previous procedure call using this array.
5-10	Procedure call information. Refer to "Library Procedure Error Messages" in appendix A for a description of this information.

textlen is an integer equal to the number of halfwords to be logged from the *text* parameter, or is a negative integer equal to the number of bytes. Length can be zero.

Discussion

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

Table 5-18. DBMEMO Return Status Values

Calling Errors:	-11	Bad database reference.
	-31	Bad mode.
	-151	Text length greater than 512 bytes.
	-222	Only DBXUNDO allowed when a dynamic transaction encounters an error.
Communications Errors:	-102	DSWRITE failure.
	-106	Remote 3000 data inconsistent.
	-107	NS 3000 or DS 3000 system error.
Logging System Failures:	-111	WRITELOG failure.
Exceptional Conditions:	-193	DBU control block is full.
	63	DBG disabled; potential damage; only DBCLOSE allowed.

Consult appendix A for more information about these conditions.

DBOPEN

INTRINSIC NUMBER 401

Initiates access to the database and establishes the user class number and access mode for all subsequent database access. DBOPEN is used in conjunction with DBCLOSE to establish and terminate access to a database.

Syntax

DBOPEN,*base*,*password*,*mode*,*status*

Parameters

base is the name of an integer array containing a string of ASCII characters. The string must consist of two blanks followed by a left-justified database name (maximum 6 characters) and terminated by a semicolon or blank (␣), for example, "␣ ␣orders;". If the database is successfully opened, TurboIMAGE/XL replaces the two blanks with a value called the base ID. The base ID uniquely identifies this access path between the database and the process calling DBOPEN. In all subsequent accesses to the database, the first halfword of *base* must be this base ID; therefore, the array should not be modified. The base ID contains a number that distinguishes between the 63 access paths allowed for each process for accessing a given database.

NOTE The base ID cannot be passed between processes in an attempt to reduce the number of required DBOPEN calls.

To access a database catalogued in a group other than the user's log-on group, the database name must be followed by a period and the group name, for example, ORDERS.GROUPX. If the database 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 can use an MPE/iX FILE command before executing the application program to equate the database name or the database-access file name to another database or database-access file name. You can use only the formal file designator, actual file designator, and the *DEV=* parameters. For additional information on the database-access file, refer to chapter 9.

password is the name of an integer 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 ␣ represents a blank):

```
␣[/USERIDENT]            Access class zero (0).
;[/USERIDENT]             Creator access.
password[/USERIDENT]    Password access.
```

If either the password or the user identifier string is less than eight characters long, it must be terminated with a semicolon or a blank.

The password establishes a user class number as described in chapter 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␣
CLERK;
CLERK;/JOE;
CLERK␣/JOE;
␣/DBA
```

mode is an integer between 1 and 8, inclusive, corresponding to the valid TurboIMAGE/XL access modes described in chapter 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

NOTE If the database is open in database access mode 1, a lock must be in effect on either the data set or the whole database when adding to or deleting from master data sets. If a data entry level lock is specified, any subsequent DBPUTs or DBDELETEs will fail with error number –12 and the following message is returned:

intrinsic name CALLED WITHOUT COVERING LOCK IN EFFECT

Lock either the entire database or data set with a data entry lock by using an @ sign to specify all data sets or all data items.

The figure in appendix B summarizes the results of multiple access to the same database. If a database cannot be opened successfully in a particular mode, this information can be used to determine the problem and to select an alternate mode.

If your database is enabled for third-party indexing (TPI), refer to your vendor documentation for additional DBOPEN mode information. The section on DBUTIL in chapter 8 of this book has a brief description of the TPI option.

status is the name of an array of 10 halfwords in which TurboIMAGE/XL returns status information about the procedure. If the procedure executes successfully, the status array contents are:

Element	Contents
1	If the procedure succeeds, the return status is 0. Table 5-19. describes the contents of element 1 when the procedure does not succeed.
2	User class number, 0 to 63 (or a 64 if the database creator enters a semicolon ";" in place of a password).
3	Current size of the DBG (in halfwords) or 32767, whichever is smaller. If it is 32767, the DBG size exceeds the maximum halfword value limit.
4	Size of the DBU (in halfwords) or 32767, whichever is smaller. If it is 32767, the DBU size exceeds the maximum half-word value limit.
5-10	Information about the current procedure call and its results. This same information is returned for all TurboIMAGE/XL procedures if an error occurs. It is described in "Library Procedure Error Messages" in appendix A.

Discussion

A process can concurrently use the database through independent, unique access paths by issuing as many as 127 calls to DBOPEN (63 calls per database) and specifying a different base array in each call. Subsequent calls to other TurboIMAGE/XL procedures must use the appropriate base array so that the correct base ID is used.

CAUTION Although a single process can call DBOPEN a maximum of 127 times (63 per database), DBOPEN can encounter an MPE/iX system limit and fail. For example, DBOPEN would fail if the limit was exceeded for mapped file space or if the process attempted to open more than the allowable number of files.

The database activity controlled on one access path relates to that controlled on other access paths in the same way the database 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.

DBOPEN performs expansion recovery if necessary. During the *first open* of the database using any open mode, DBOPEN automatically performs expansion recovery for any detail data set with the "expansion in progress" flag turned on and a previous DBPUT capacity expansion had not completed. Recovery correctly adjusts the data set free count and the root file data set capacity fields using the actual data set file size. Use the SHOW CAPACITY command in DBUTIL to detect if recovery is required for the data set. If the message, "dynamic capacity expansion in progress flag is on," is displayed for the data set, and asterisks are in the "no. of entries" and "%max cap" fields, then recovery is required for the data set(s). QUERY or any application performing the first DBOPEN of the database can be used to recover the detail data set capacity. Use DBChange Plus or a third-party application to change an existing detail data set to have parameters for dynamic expansion.

If the database is enabled for logging, and the program calls DBOPEN in one of modes 1-4, then TurboIMAGE/XL attempts to access a log file using the MPE/iX OPENLOG intrinsic. OPENLOG succeeds only if the following have been completed:

1. A valid log identifier and log password have been set into the database root file using the DBUTIL >>SET command, and
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 also fails and returns an appropriate error condition. If OPENLOG succeeds, DBOPEN causes 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 database in one of modes 1-4, and the database is enabled for logging. A program does not log if it opens in one of modes 5-8, or if the database is not enabled for logging.

If DBRECOV roll-back recovery is enabled, the first DBOPEN checks if the user logging file and the database are attached to the same Transaction Management (XM) user log set. The database and the user logging file must be kept synchronized at the XM level in order for DBRECOV roll-back recovery to work.

DBOPEN initiates recovery of the incomplete dynamic transactions, if necessary; then DBRECOV rolls back the incomplete static transactions.

Dynamic transactions are not allowed with DBOPEN mode 2.

Table 5-19. DBOPEN Return Status Values

File System, Memory Management, and Transaction Management Failures:	-1	HPFOPEN or FOPEN failure.
	-2	FCLOSE failure.
	-3	FREADDIR failure.
	-4	FREADLABEL failure.
	-5	FWRITEDIR error.
	-6	FWRITELABEL error.
	-8	FUNLOCK failure.
	-9	Cannot create a control block.
	-10	FFILEINFO failure.
	-168	Cannot attach *n* to MPE XL XM: file system error *nn*.
	-169	Invalid mode for XM attach options.
	-175	Cannot attach *n* to MPE XL XM: XM error *nn*.
	-176	Cannot detach *n* from MPE XL XM: XM error *nn*.
	-177	User log file is not in the same volume set as database.
	-178	Cannot detach *n* from MPE XL XM: file system error *nn*.
	-179	Cannot begin MPE XL XM transaction for attach.
	-198	Total DBOPEN count/user exceeds limit.
	-208	FLABELINFO failure.
	-209	Invalid mode for XM detach.
	-210	MPE error *decimal integer* while getting log file name.
	-211	Invalid or no userlabel.

Table 5-19. DBOPEN Return Status Values

Calling Errors:	-11	Bad database reference.
	-13	Must be creator of root file or database.
	-21	Bad password.
	-22	Maintenance word required.
	-31	Bad mode.
	-32	Unobtainable mode.
	-34	Database must be recovered before access is allowed.
	-90	Root file bad: unrecognized state: *% octal integer*.
	-91	Bad root modification level.
	-92	Database not created.
	-94	Database bad: Was being modified with output deferred, may not be accessed in mode *decimal integer*.
	-95	Database bad: Creation was in process (create again).
	-96	Database bad: Erase was in process (erase again).
	-220	Database and user log not attached to the same XM log set.
Communications Errors:	-15	DSLINE or remote HELLO failure; setup for RDBA failed.
	-60	Illegal file equation on root file.
	-61	Error while obtaining information about file equation.
	-100	DSOPEN failure.
	-101	DSCLOSE failure.
	-102	DSWRITE failure.
	-103	Remote 3000 space insufficient.
	-104	Remote system does not support TurboIMAGE/XL.
	-105	Remote 3000 cannot create control block.
	-106	Remote 3000 data inconsistent.
	-107	NS 3000 or DS 3000 system error.
Logging System Failures:	-110	OPENLOG failure.
	-111	WRITELOG failure.
	-114	Roll-back enabled without logging.

Table 5-19. DBOPEN Return Status Values

Native Language Support Errors:	-200	Database language not system supported.
	-201	Native Language Support not installed.
	-202	MPE Native Language Support error *decimal integer* returned by NLINFO.
Exceptional Conditions:	-167	Cannot begin MPE XL XM transaction: XM error *nn*.
	-191	DBS control block is full.
	-199	Cannot end MPE XL XM transaction: XM error *nn*.
	-220	User log and database not attached to same XM log.
	-226	Error occurred when the 00 file was created.
	-227	Error occurred in 00 file recovery.
	-234	Cannot purge the 00 file.
	-236	Internal error occurred when opening the AUX file: error *nn*.
	-250	Failure in semaphore initialization.
	-253	Database enabled for indexing, but third-party indexing is not configured.
	-331	Invalid DSET Capacity.
	-332	Error in QLOCK table operation.
	-333	Error in QOPEN table operation.
	60	Database access disabled.
	61	This database opened more than 63 times by the same process.
	62	DBG full.
	63	DBG disabled; potential damage; only DBCLOSE allowed.
	64	PCBX full.
	66	The current DBG for the database does not appear correct (TurboIMAGE internal error).
	68	DBB disabled by an abort.

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 database must be opened 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 database. The first element 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 can be up to 16 characters long. If shorter, it must be terminated by a semicolon or a blank.

mode must be an integer equal to 1.

If your database is enabled for third-party indexing (TPI), refer to your vendor documentation for additional DBPUT mode information. The section on DBUTIL in chapter 8 of this book has a brief description of the TPI option.

status is the name of an array of 10 halfwords in which TurboIMAGE/XL returns status information about the procedure. If the procedure executes successfully, the status array contents are:

Element	Contents
1	If the procedure succeeds, the return status is 0. Table 5-21. describes the contents of element 1 when the procedure does not succeed.
2	Length of logical entry in buffer array (in halfwords).
3-4	Word record number of new entry.
5-6	Word 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, word record address of predecessor on synonym chain. If detail, word record number of predecessor on current detail chain.
9-10	If detail, word record number of successor on current chain. If master, word zero.

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. For example:

```
ACCOUNT,LAST-NAME,CITY,STATE;
```

When referencing by number, the first halfword 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/XL 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 programs in chapter 6. List processing is a relatively high overhead operation which can 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 must be in the same order as their data item identifiers in the list array. The number of halfwords for each value must correspond to the number required by its type; for example, I2 values must be 2 halfwords long.

Table 5-20. Special *list* Parameter Constructs

Construct	*list* Array Contents	Purpose
Empty	0; or 0␣ or ; or ␣ (0 must be ASCII.)	Request no data transfer.
Empty Numeric	0 (*n*, length of data item identifier list, is zero)	Request no data transfer.
Asterisk	*; or *␣	Request procedure to use previous *list* and apply it to same data set. This construct saves TurboIMAGE/XL processing time, especially if more than one or two items are involved. If "*" is used to define the list in the first call to DBGET and DBPUT, TurboIMAGE/XL treats it as a zero.

Table 5-20. Special *list* Parameter Constructs

Construct	*list* Array Contents	Purpose
Commercial At-Sign	@; or @␣	Request procedure to use all data items of the data set in the order of their occurrence in the entry.
NOTE:	␣ indicates blank	

Discussion for Master Data Sets

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

- The data set must be a manual master.
- The key 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.
- Space must be available in the master set, or must be dynamically expandable to add an entry.
- If dynamic capacity expansion parameters are specified for the master data, when the master data set is almost full, expansion is done by the incremental amount. If the expansion is successful, the new record is added to the database. If the expansion is not successful, an error message is displayed, and the record is not added to the database. If there is insufficient disc space to expand the data set to the *full* incremental amount, DBPUT will perform a *partial* expansion up to the disc space available. DBPUT will terminate if there is no available group or account disc space even if there is enough system disc space. (The current capacity for a data set can be displayed by the SHOW CAPACITY command in DBUTIL or the output buffer from DBINFO modes 202 and 205.)
- 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.
- Values for data items not included in the list array are filled with binary zeros.
- The caller must have a lock on the data set or the database if the database is open in access mode 1.
- DBPUT to an indexed master triggers a similar operation to indexed master's B-Tree index file.

Discussion for Detail Data Sets

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

- The data set must have free space for the entry.
- If the database 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.
- Values for data items not included in the list array 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.
- Proper Native Language collating sequence must be maintained for chain sorting.
- If dynamic capacity expansion is allowed for the detail data set, when the detail data set reaches the end of the current allocation (that is, data set free count is zero), expansion is requested by the incremental amount. If the expansion is successful, the new record is added to the database. If the expansion is not successful, an error message is displayed, and the record is not added to the database. If there is insufficient disc space to expand the data set to the *full* incremental amount, DBPUT will perform a *partial* expansion up to the disc space available. DBPUT will terminate if there is no available group or account disc space even if there is enough system disc space. (The current capacity for a data set can be displayed by the SHOW CAPACITY command in DBUTIL or the output buffer from DBINFO modes 202 and 205.)

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 elements 7 through 10.

The record number of the new data entry is returned to status halfwords 3-4; and its forward and backward pointers are returned in status halfwords 7-8 and 9-10, respectively. If you intend to use these numbers for directed reads (see "Directed Access" in chapter 4), save them because subsequent TurboIMAGE/XL procedure calls can overwrite the status area.

The execution of a call to DBPUT could require extensive resources depending on the amount of chain maintenance required. For example, when an entry is added to a detail data set, the new entry must be linked to all other related entries with the same key values and to all of its related master entries. This operation could involve many blocks of data. TurboIMAGE/XL prevents data block access conflicts with all other users and ensures data integrity by applying a temporary lock on other processes until the call to DBPUT completes. The timing of this temporary lock can be controlled with the PREFETCH option of DBUTIL. Refer to "Coordinating Additions to a Database" in chapter 4 for considerations when enabling or disabling this option.

Performance may be impacted by the number of entries incremented when DBPUT is used to dynamically expand the detail data set. The number of disc extents used for the data set file may also impact the performance of TurboIMAGE/XL.

If the process is logging, a call to DBPUT causes a log record to be written with such

information as the time, date, user identification number, and a copy of the new record to be added.

If DBPUT is called within a dynamic transaction, a log record is written after the physical transaction has been successfully completed. If the intrinsic cannot be completed, an error is returned. This error condition must be checked, and you must decide to use DBXUNDO, DBXEND, or continue with the remainder of the dynamic transaction. DBXUNDO will abort the entire dynamic transaction. DBXEND will terminate the dynamic transaction; the modifications completed thus far within the transaction will remain in the database.

Table 5-21. DBPUT Return Status Values

File System, Memory Management, and Transaction Management Failures:	-1	FOPEN failure.
	-3	FREADDIR failure.
	-4	FREADLABEL failure.
	-5	FWRITEDIR failure.
	-6	FWRITELABEL failure.
	-167	Cannot begin MPE XL XM transaction: XM error *nn*.
	-168	Cannot attach *n* to MPE XL XM: file system error *nn*.
	-169	Invalid mode for XM attach options.
	-175	Cannot attach *n* to MPE XL XM: XM error *nn*.
	-176	Cannot detach *n* from MPE XL XM: XM error *nn*.
	-178	Cannot detach *n* from MPE XL XM: file system error *nn*.
	-199	Cannot end MPE XL XM transaction: XM error *nn*.
	-209	Invalid mode for XM detach options.
Calling Errors:	-11	Bad database reference.
	-12	No lock covers the data entry to be added. (Occurs only if database is open in access mode 1.)
	-14	Illegal intrinsic in current access mode.
	-21	Bad data set reference.
	-23	Data set not writable.
	-24	Operation not allowed on automatic master data set.
	-31	Bad mode.
	-51	Bad list length.
	-52	Bad list or bad item.
	-53	Missing search or sort item.
	-222	Only DBXUNDO allowed when a dynamic transaction encounters an error.

Table 5-21. DBPUT Return Status Values

Communications Errors:	-102	DSWRITE failure.
	-106	Remote 3000 data inconsistent.
	-107	NS 3000 or DS 3000 system error.
Logging System Failures:	-111	WRITELOG failure.
Exceptional Conditions:	-193	DBU control block is full.
	-196	DBB control block is full.
	-212	Database corruption detected.
	-264	XM write procedure returned 1030 or 1040.
	-312	Internal error encountered while reading database block.
	-314	Error while obtaining path information for set.
	-322	Error returned by process list.
	-332	Error in QLOCK table operation.
	16	Data set full.
		(In the following messages, *dataset#* and FSERR # will be replaced by the actual number.)
		DBPUT cannot expand *dataset#*: dataset at maximum capacity.
		DBPUT *dataset#* incomplete expansion: File system error #.
		DBPUT cannot expand *dataset#*: Out of disc space (FSERR #).
	18	Broken chain; forward and backward pointers not consistent.
	43	Duplicate key item value.
	62	DBG control block is full.
	63	DBG disabled; potential damage; only DBCLOSE allowed.
	1*nn*	Missing chain head for path number *nn*.
	2*nn*	Full chain for path number *nn*.
	3*nn*	Internal error.
	4*nn*	Full synonym chain.

Refer to appendix A for more information about these conditions.

DBUNLOCK

INTRINSIC NUMBER 410

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

If DBUNLOCK is called when a dynamic transaction is active and a modify intrinsic (DBPUT, DBDELETE, or DBUPDATE) has already been used in the dynamic transaction (that is, the database is modified), the DBUNLOCK fails. You must check the error condition. You may use DBERROR or DBEXPLAIN to display the error message. When a DBUNLOCK fails within the dynamic transaction, dynamic intrinsic rollback allows the following choices:

- Use DBXEND to end the dynamic transaction.
- Continue with the remainder of the dynamic transaction taking into account that DBUNLOCK failed and locks are still in place.
- Use DBXUNDO to rollback the entire dynamic transaction.

Syntax

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

Parameters

base is the name of the array used for the *base* parameter when opening the database. The first element of the array must contain the base ID returned by DBOPEN.

dset is currently unused. Use the Not_Used_Parm or DUMMY variable as recommended at the beginning of this chapter or any *dset* array used for other procedures.

mode must be an integer equal to 1.

status is the name of an array of 10 halfwords in which TurboIMAGE/XL returns status information about the procedure. If the procedure executes successfully, the status array contents are:

Element	Contents
1	If the procedure succeeds, the return status is 0. Table 5-22. describes the contents of element 1 when the procedure does not succeed.
2	Number of lock descriptors released by this call. Each data set lock or database lock is counted as one descriptor.
3-4	Reserved for internal use.
5-10	Information about the procedure call and its results. Refer

to "Library Procedure Error Messages" in appendix A for a description of this information.

Table 5-22. DBUNLOCK Return Status Values

File System, Memory Management, and Transaction Management Failures:	-4	MPE file error *nn* returned by DBUNLOCK on FREADLABEL.
	-6	MPE file error *nn* returned by DBUNLOCK on FWRITELABEL.
	-167	Cannot begin MPE XL XM transaction: XM error *nn*.
	-199	Cannot end MPE XL XM transaction: XM error *nn*.
Calling Errors:	-11	Bad database reference.
	-31	Bad mode.
	-215	XM error *nn* encountered when rolling out dynamic transaction.
	-222	Only DBXUNDO allowed when a dynamic transaction encounters an error.
	-230	A DBUNLOCK inside a dynamic transaction is not allowed.
	-231	During Dynamic Rollback recovery, internal procedure failed; error *nn*.
Communications Errors:	-102	DSWRITE failure.
	-106	Remote 3000 data inconsistent.
	-107	NS 3000 or DS 3000 system error.
Exceptional Conditions:	63	DBG disabled; potential damage; only DBCLOSE allowed.

Appendix A contains more information about these conditions.

DBUPDATE

INTRINSIC NUMBER 406

Modifies values of data items in the entry residing at the current record address of a specified data set. To call DBUPDATE, you must open the database in access mode 1, 2, 3, or 4. The update is always carried out correctly against the latest version of the data, regardless of modifications made by other users.

In database access mode 1, 3, or 4, you can use DBUPDATE to modify the values of detail data set search and sort items if permitted by the critical item update (CIUPDATE) option settings for the database and the current process. Master data set key item values *cannot* be modified even if CIUPDATE is permitted.

Syntax

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

Parameters

base is the name of the array used as the *base* parameter when opening the database. The first element 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 can be up to 16 characters long. If shorter, it must be terminated by a semicolon or a blank.

mode must be an integer equal to 1.

If your database is enabled for third-party indexing (TPI), refer to your vendor documentation for additional DBUPDATE mode information. The section on DBUTIL in chapter 8 of this book has a brief description of the TPI option.

status is the name of an array of 10 halfwords in which TurboIMAGE/XL returns status information about the procedure. If the procedure operates successfully, the status array contents are:

Element	Contents
1	If the procedure succeeds, the return status is 0. Table 5-23. describes the contents of element 1 when the procedure does not succeed.
2	Length of the values in buffer (in halfwords).
3-10	Same word values set by preceding procedure call which positioned the data set at the current entry. If critical item update is permitted, the value contained in element 3 determines the message returned.

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 database is opened must allow at least read access to all the items included in the *list* array.

If the corresponding buffer array values are the same as the current data item values, the *list* array can include data items to which the user has read access only, such as, key, search and sort items. This feature permits reading and updating with the same *list* array contents. Those items to be updated must allow write access and *cannot* be key, search, or sort items.

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

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

The *list* not only specifies the data items to be updated immediately but is saved internally by TurboIMAGE/XL 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 that can 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 halfwords for each value must correspond to the number of halfwords required by its type multiplied by the sub-item count. Search and sort item values can be included in this update list if their values will not change.

Discussion

Before performing an update for a database opened in access mode 1, TurboIMAGE/XL 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 causes a log record to be written with such information as the time, date, user identification number, and a copy of both the old and new data item values.

When DBUPDATE is called within a dynamic transaction, a log record is written after the successful completion of the physical transaction. If the intrinsic cannot be completed, an error is returned. This error condition must be checked, and you must decide to use

DBXUNDO, DBXEND, or continue with the remainder of the dynamic transaction. DBXUNDO will abort the entire transaction. DBXEND will terminate the dynamic transaction; the modifications completed thus far within the transaction will remain in the database.

Table 5-23. DBUPDATE Return Status Values

File System, Memory Management, and Transaction Management Failures:	-1	FOPEN failure.
	-3	FREADDIR failure.
	-4	FREADLABEL failure.
	-5	FWRITEDIR failure.
	-167	Cannot begin MPE XL XM transaction: XM error.
	-168	Cannot attach *n* to MPE XL XM: file system error *nn*.
	-169	Invalid mode for XM attach options.
	-175	Cannot attach *n* to MPE XL XM: XM error *nn*.
	-176	Cannot detach *n* from MPE XL XM: XM error *nn*.
	-178	Cannot detach *n* from MPE XL XM: file system error *nn*.
	-199	Cannot end MPE XL XM transaction: XM error *nn*.
	-209	Invalid mode for XM detach options.
Calling Errors:	-11	Bad database reference.
	-12	No locks cover the data entry to be updated. (Occurs only if database is open in access mode 1.)
	-14	Illegal intrinsic in current access mode.
	-21	Bad data set reference.
	-31	Bad mode.
	-51	Bad *list* length.
	-52	Bad list or bad item.
	-82	CIUPDATE is set to DISALLOWED; cannot use critical item update.
	-222	Only DBXUNDO allowed when a dynamic transaction encounters an error.
Communications Errors:	-102	DSWRITE failure.
	-106	Remote 3000 data inconsistent.
	-107	NS 3000 or DS 3000 system error.
Logging System Failures:	-111	WRITELOG failure.

Table 5-23. DBUPDATE Return Status Values

Exceptional Conditions:	-193	DBU control block is full.
	-264	Error while writing to TPI files.
	-3*nn*	Internal error.
	-312	Error while reading database file.
	-314	Error while getting path information for set.
	-332	Error in QLOCK table operation.
	17	No entry.
	41	DBUPDATE attempted to modify value of critical item--key, search or sort.
	42	Read only item.
	49	Illegal buffer address.
	50	Buffer too small.
	62	DBG full.
	63	DBG disabled; potential damage; only DBCLOSE allowed.
	68	DBB disabled.

Appendix A contains more information about these conditions.

DBXBEGIN

INTRINSIC NUMBER 420

Designates the beginning of a sequence of TurboIMAGE/XL procedure calls that are to be regarded as a dynamic transaction of a single database or dynamic transaction spanning multiple databases (DMDBX) for the purposes of logging and dynamic roll-back recovery. The *text* parameter can be used to log user information to the log file. DBXBEGIN is used in conjunction with DBXEND to begin and end a dynamic transaction.

Syntax

```
DBXBEGIN,{ base
           baseidlist} ,te0xt,mode,status,textlen
```

Parameters

base is the name of the array used as the base parameter when opening the database. The first element of the array must contain the base ID returned by DBOPEN. (Refer to DBOPEN for more information about the base ID.)

baseidlist is the name of the integer array containing the base IDs of the databases which are involved in a DMDBX. Use *baseidlist* when calling DBXBEGIN mode 3 (DMDBX). The layout of this array is shown here (each element is a halfword or two bytes):

Element	Contents
1-2	Application program must set these two halfwords to binary 0s before calling DBXBEGIN. After returning to the calling program, these two halfwords contain the transaction ID. Use this same *baseidlist* with the corresponding DBXEND or DBXUNDO intrinsics.
3	Number of base IDs involved in the DMDBX. This must be a number between 1 and 15 inclusive.
4-*n*	Base IDs of the databases involved in the DMDBX. Each base ID occupies one half-word or 2 bytes and it is the first halfword of the *base* parameter used to call TurboIMAGE/XL intrinsics.

text is the name of an array up to 256 halfwords long that contains user ASCII or binary data to be written to the log file as part of the DBXBEGIN log record. The *text* argument is used to assign each particular transaction a distinct name. (Refer to "Discussion" below for more information.)

mode is an integer indicating the transaction type:

Mode 1: Indicates a dynamic transaction which spans only one database.

Mode 3: Indicates a dynamic transaction spanning multiple databases (DMDBX). If user logging is enabled for the databases, mode 3 generates *multiple* entries in the log file, one for each database.

status is the name of an array of 10 halfwords in which TurboIMAGE/XL returns status information. If the procedure executes successfully, the status array contents are:

Element	Contents
1	If the procedure succeeds, the return status is 0. Table 5-24. describes the contents of element 1 when the procedure does not succeed.
2-4	Unchanged from previous procedure call using this array.
5-10	Procedure call information. Refer to "Library Procedure Error Messages" in appendix A for a description of this information.

textlen is an integer equal to the number of halfwords to be logged from the *text* parameter, or is a negative integer equal to the number of bytes. Length can be zero.

Discussion

DBXBEGIN is called to designate the beginning of a sequence of TurboIMAGE/XL procedure calls that are jointly considered a single dynamic transaction. The end of such a sequence is designated by a matching call to DBXEND. DBXBEGIN cannot be called if another transaction started by DBXBEGIN or DBBEGIN is active. The intrinsic does not begin a dynamic transaction if AUTODEFER is enabled for the database.

Before including a database in a DMDBX, DBXBEGIN mode 3, DBCONTROL mode 7 needs to have been done *once* for each of the databases in the DMDBX. DBCONTROL mode 7 remains active until the database is closed or the application terminates. DBCONTROL mode 7 also enables the database for deadlock detection. If deadlock is encountered, it returns an error 26, instead of triggering a process hang.

NOTE DBXBEGIN is not allowed with DBOPEN mode 2 nor with AUTODEFER enabled.

Logging and DBRECOV are not needed with dynamic transactions, because the database can be recovered dynamically. However, if the calling process is logging, DBXBEGIN causes a record to be written to the log file to identify the beginning of a dynamic transaction.

Table 5-24. DBXBEGIN Return Status Values

File System, Memory Management, and Transaction Management Failures:	-4	FREADLABEL error.
	-228	DBXBEGIN encountered XM error *nn* when starting dynamic transaction.
Calling Errors:	-11	Bad database reference.
	-31	Bad (unrecognized) DBXBEGIN mode: *n*.
	-139	Invalid number of base IDs.
	-140	Bad base ID list.
	-151	Text length greater than 512 bytes.
	-152	DBXBEGIN called while a transaction is in progress.
	-217	DBOPEN mode *n* incompatible with Dynamic Rollback.
	-218	Output deferred not compatible with DBX.
	-219	Remote database access incompatible with Dynamic Rollback.
	-221	Cannot begin transaction when a dynamic transaction is active.
	-222	Only DBXUNDO allowed when a dynamic transaction encounters an error.
	-242	Error in TurboGTX file operation.
Logging System Failures:	-111	WRITELOG intrinsic failure.
Exceptional Conditions:	67	DBU disabled; potential damage; only DBCLOSE allowed.
	-332	Error in QLOCK table operation.

Consult appendix A for more information about these conditions.

DBXEND

INTRINSIC NUMBER 421

Designates the end of a sequence of TurboIMAGE/XL procedure calls regarded as a dynamic transaction for the purposes of logging and dynamic roll-back recovery. The *text* parameter can be used to log user information to the log file. DBXEND is used in conjunction with DBXBEGIN to begin and end a dynamic transaction.

Syntax

```
DBXEND ,{ base
          baseidlist} text,mode,status,textlen
```

Parameters

base is the name of the array used as the base parameter when opening the database. The first element of the array must contain the base ID returned by DBOPEN. (Refer to DBOPEN for more information about the base ID.)

baseidlist is the name of the integer array containing the base IDs of the databases which are involved in the DMDBX. Use the same *baseidlist* used with DBXBEGIN when calling DBXEND mode 3 to end the DMDBX. The layout of this array is same as in DBXBEGIN, except that the transaction ID is already set in the first two halfwords by DBXBEGIN.

text is an array up to 256 halfwords long that contains user ASCII or binary data to be written to the log file as part of the DBXEND log record.

mode must be a halfword equal to 1, 2 or 3.

Mode 1: End of dynamic transaction spanning one database.

Mode 2: End of dynamic transaction spanning one database, started with DBXBEGIN mode 1, and write contents of the transaction management (XM) logging buffer in memory to disk. If logging is enabled, the contents of the logging buffer in memory will also be written to disk.

Mode 3: Indicates the end of a DMDBX started with DBXBEGIN mode 3. If user logging is enabled for the databases, mode 3 generates *multiple* entries in the log file, one for each database in the DMDBX, in order to mark the end of a dynamic transaction.

status is the name of an array of 10 halfwords in which TurboIMAGE/XL returns status information about the procedure. If the procedure executes successfully, the status array contents are as follows:

Element	Contents
1	If the procedure succeeds, the return status is 0. Table

		5-25. describes the contents of element 1 when the procedure does not succeed.
	2-4	Unchanged from previous procedure call using this array.
	5-10	Procedure call information. Refer to "Library Procedure Error Messages" in appendix A for a description of this information.

textlen is an integer equal to the number of halfwords to be logged from the *text* parameter, or is a negative integer equal to the number of bytes. Length can be zero.

Discussion

DBXEND is called to designate the end of a sequence of TurboIMAGE/XL procedure calls that are collectively considered a dynamic transaction. If an intrinsic fails within a dynamic transaction, you may use DBXEND to end the dynamic transaction at that point. The modifications completed until that point will remain in the database(s). The beginning of such a sequence is designated by a previous call to DBXBEGIN. DBXEND cannot be called to end a transaction started by DBBEGIN.

Logging and DBRECOV are not needed with dynamic transactions, because the database can be recovered dynamically. However, if the calling process is logging, DBXEND causes a record to be written to the log file to identify the end of a dynamic transaction.

If you call DBXEND with mode 2, DBXEND forces the XM log buffer (and user log buffer if the process is logging) to be written from memory to disk before returning to the calling process. Use this option only for critical transactions; too many mode 2 DBXEND calls can degrade performance by causing a disk access each time a dynamic transaction ends.

DBXEND returns an error condition if it is called without a prior matching call to DBXBEGIN. DBXEND is not necessary after a DBXUNDO.

Table 5-25. DBXEND Return Status Values

File System, Memory Management, and Transaction Management Failures:	-213	DBXEND encountered XM error *nn* when ending dynamic transaction.
Calling Errors:	-11	Bad database reference.
	-31	Bad (unrecognized) DBXEND mode: *n*.
	-140	Bad baseid list.
	-151	Text length greater than 512 bytes.
	-219	Remote database access incompatible with Dynamic Rollback.
	-222	Only DBXUNDO allowed when a dynamic transaction encounters an error.
	-223	Cannot DBXEND or DBXUNDO a transaction which was not active.
	-237	Cannot DBXEND or DBXUNDO a DBBEGIN transaction.
	-238	MDBX DBXBEGIN, DBXEND mode mismatch.
Logging System Failures:	-111	WRITELOG failure.
Exceptional Conditions:	67	DBU disabled; potential damage; only DBCLOSE allowed.
	-213	XM error on DBXEND.
	-242	Error in TurboGTX file operation.
	-332	Error in QLOCK table operation.

Consult appendix A for more information about these conditions.

DBXUNDO

INTRINSIC NUMBER 422

Rolls back the active sequence of TurboIMAGE/XL procedure calls which are considered a dynamic transaction.

Syntax

```
DBXUNDO ,{ base
         baseidlist} ,text,mode,status,textlen
```

Parameters

base is the name of the array used as the base parameter when opening the database. The first element of the array must contain the base ID returned by DBOPEN. (Refer to DBOPEN for more information about the base ID.)

baseidlist Name of the integer array containing the base IDs of the databases which are involved in the DMDBX. Use the same *baseidlist* parameter of DBXBEGIN mode 3, when calling DBXUNDO to roll back a DMDBX.

text is an array up to 256 halfwords long that contains user ASCII or binary data to be written to the log file as part of the DBXUNDO log record.

mode Must be a halfword equal to 1 when employing *base* as the base parameter or 3 when using *baseidlist* with the matching DBXBEGIN call.

Mode 1: Dynamically roll back DBPUT, DBDELETE, and DBUPDATE intrinsics which completed successfully since the matching DBXBEGIN mode 1 call.

Mode 3: Dynamically roll back DBPUT, DBDELETE, and DBUPDATE intrinsics which completed successfully inside the DMDBX to their respective databases since the matching DBXBEGIN mode 3 call. If user logging is enabled for the databases, mode 3 generates *multiple* entries in the log file, one for each database, in order to mark the roll back of the dynamic transaction.

status is the name of an array of 10 halfwords in which TurboIMAGE/XL returns status information about the procedure. If the procedure executes successfully, the status array contents are as follows:

Element	Contents
1	If the procedure succeeds, the return status is 0. Table 5-26. describes the contents of element 1 when the procedure does not succeed.
2-4	Unchanged from previous procedure call using this array.

	5-10	Procedure call information. Refer to "Library Procedure Error Messages" in appendix A for a description of this information.

textlen is an integer equal to the number of halfwords to be logged from the *text* parameter, or is a negative integer equal to the number of bytes. Length can be zero.

Discussion

DBXUNDO is called to dynamically roll back a sequence of TurboIMAGE/XL procedure calls that completed successfully inside a dynamic transaction. The beginning of such a sequence is designated by a previous call to DBXBEGIN. DBXUNDO cannot be called to roll back a transaction started by DBBEGIN or if the database is enabled for AUTODEFER.

Logging and DBRECOV are not needed with dynamic transactions, because the database can be recovered dynamically. However, if the calling process is logging, DBXUNDO causes a record to be written to the log file to identify the transaction in the event it needs to be recovered.

CAUTION After DBXUNDO is called, the current record pointer, current path, current list, and chronological order prior to the call to DBXBEGIN may not be restored.

DBXUNDO cannot be called to roll back a transaction started by DBBEGIN. DBXUNDO returns an error condition if it is called without a prior matching call to DBXBEGIN. DBXEND is not necessary after DBXUNDO. DBXUNDO rolls back the entire transaction, and then transactions continue according to the logic of the program. In general, DBXUNDO or DBXEND must be the last intrinsic for a dynamic transaction to be executed. It designates the end of that dynamic transaction.

Table 5-26. DBXUNDO Return Status Values

File System, Memory Management, and Transaction Management Failures:	-215	XM error *nn* encountered when rolling out dynamic transaction.
Calling Errors:	-11	Bad database reference.
	-31	Bad (unrecognized) DBXUNDO mode *n*.
	-140	Bad baseid list.
	-151	Text length greater than 512 bytes.
	-218	Output deferred is incompatible with Dynamic Rollback.
	-219	Remote database access is incompatible with Dynamic Rollback.
	-223	Cannot DBXEND or DBXUNDO a transaction which was not active.
	-237	Cannot DBXEND or DBXUNDO a DBBEGIN transaction.
	-238	MDBX DBXBEGIN, DBXEND mode mismatch.
	-240	MDBX mode mismatch.
Logging System Failures:	-111	WRITELOG failure.
Exceptional Conditions:	-231	During DBX recovery, internal procedure failed; error *nn*.
	-242	Error in TurboGTX file operation.
	-332	Error in QLOCK table operation.
	67	DBU disabled; potential damage; only DBCLOSE allowed.

Consult appendix A for more information about these conditions.

6 Host Language Access

You can access TurboIMAGE/XL from Compatibility Mode or Native Mode application programs. Compilers are available on MPE/iX in one or both modes in BASIC, BBASIC, C, COBOL, COBOL II, FORTRAN 66, FORTRAN 77, Pascal, RPG, and SPL. This chapter focuses on specific programming languages for use with Native Mode only. For Compatibility Mode examples, refer to the *TurboIMAGE/V Database Management System Reference Manual*.

The first section of this chapter presents a model program written in pseudo code; the subsequent sections discuss using TurboIMAGE/XL with specific programming languages.

NOTE If you are an experienced TurboIMAGE/XL programmer, you can skip the model program and go directly to the section containing the language in which you write your application.

The following languages are presented in alphabetical order in this chapter:

- C
- COBOL II
- FORTRAN 77
- Pascal
- RPG

Each discussion includes:

- A presentation of any needed language-specific TurboIMAGE/XL information.
- An example showing some of the various model program routines written in the particular language. Note that the COBOL II program example shows all of the routines included in the model program.

The COBOL II program is a complete, executable program. The RPG program is also executable, although it contains only a subset of the model program routines. The other language examples show only portions of the model program but do demonstrate many of the TurboIMAGE/XL procedures. All of the examples are designed to illustrate the most simple and direct way TurboIMAGE/XL procedures are called. They are not intended as examples of the best way to code the tasks that are illustrated; 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. For information on the TurboIMAGE/XL data types to be used with these languages, refer to chapter 3.

NOTE In this manual a **word** is a 32-bit storage unit and a **halfword** is a 16-bit storage unit. One byte is 8 bits.

Model Program

This section shows the model for the example programs that run against the ORDERS database. You may wish to skip this section if you are an experienced TurboIMAGE/XL programmer.

The main entry point for the application is a numbered list of functions that can be performed by calling the various routines. Each routine is made up of one or more tasks that are implemented through TurboIMAGE/XL intrinsic calls. Each call contains one or more parameters, some of which are used to pass their assigned values to TurboIMAGE/XL intrinsics. These values determine the outcome of the intrinsic. Other parameters are used to receive data and status information from TurboIMAGE/XL intrinsics.

This model attempts to familiarize you with generic techniques for making TurboIMAGE/XL intrinsic calls. Additionally, the model reiterates the importance of defining TurboIMAGE/XL transactions for logging and recovery.

The conventions used throughout the model and the structure of the program are explained immediately preceding the model program.

ORDERS Database Schema

Figure 6-1. contains the list file output printed when the schema of the sample ORDERS database is processed. This schema is shown earlier in chapter 3, but is repeated here for easy reference when reading through the model program. Use it to refer to the data set and data item names used in the model program.

NOTE Because the Schema Processor, DBSCHEMA, upshifts alphabetic characters, programs must specify data set and data item names in all uppercase characters. Take note of this if the programming language you use does not require uppercase characters.

Figure 6-1. ORDERS Database Schema Listing

```
PAGE 1

HEWLETT-PACKARD 30391C.05.00 TurboIMAGE/3000: DBSCHEMA TUE, JAN 11

    $CONTROL LIST,LINES=46
    $PAGE "SCHEMA FOR DATABASE ORDERS"

BEGIN DATABASE ORDERS;

PASSWORDS:
 11 CREDIT;      << CUSTOMER CREDIT OFFICE >>
 12 BUYER;       << BUYER - RESPONSIBLE FOR PARTS INVENTORY >>
 13 SHIP-REC;    << WAREHOUSE - SHIPPING AND RECEIVING >>
 14 CLERK;       << SALES CLERK >>
 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-ADDRESS, X26 (12,13,14/11); << NUMBER AND STREET ADDRESS>>
 SUPPLIER,       X16 (12,13/);      << SUPPLYING COMPANY NAME>>
 TAX,            J2 (14/);          << SALES TAX (PENNIES)>>
 TOTAL,          J2 (11,14/);       << TOTAL AMOUNT OF SALE (PENNIES)>>
 UNIT-COST,      P8 (/12);          << UNIT COST OF PRODUCT>>
 ZIP,            X6 (12,13,14/11);  << ZIP CODE>>

SETS:
 NAME:       DATE-MASTER,AUTOMATIC,DISC1;      <<DATE MASTER>>
 ENTRY:      DATE(3);
 CAPACITY:   365;

 NAME:       CUSTOMER,MANUAL(14/11,18),DISC1;  <<CUSTOMER MASTER>>
 ENTRY:      ACCOUNT(1),
             LAST-NAME,
             FIRST-NAME,
             INITIAL,
             STREET-ADDRESS,
             CITY,
             STATE,
             ZIP,
             CREDIT-RATING;
 CAPACITY:   201;

 PAGE 2  SCHEMA FOR DATABASE ORDERS


 NAME:       PRODUCT,MANUAL(13,14/12,18),DISC1;<<PRODUCT MASTER>>
 ENTRY:      STOCK#(2),
             DESCRIPTION;
 CAPACITY:   300;

 NAME:       SUP-MASTER,MANUAL(13/12,18),DISC1; <<SUPPLIER MASTER>>
 ENTRY:      SUPPLIER(1),
             STREET-ADDRESS,
             CITY,
             STATE,
             ZIP;
 CAPACITY:   201;
```

```
 NAME:       INVENTORY,DETAIL(12,14/13,18),DISC2; <<INVENTORY DETAIL>>
 ENTRY:      STOCK#(PRODUCT),
             ONHANDQTY,
             SUPPLIER(!SUP-MASTER),        <<PRIMARY PATH>>
             UNIT-COST,
             LASTSHIPDATE(DATE-MASTER),
             BINNUM;
 CAPACITY:   1800, 450, 10%;

 NAME:       SALES,DETAIL(11/14,18),DISC2; <<SALES DETAIL>>
 ENTRY:      ACCOUNT(CUSTOMER(PURCH-DATE)),
             STOCK#(PRODUCT),
             QUANTITY,
             PRICE,
             TAX,
             TOTAL,
             PURCH-DATE(DATE-MASTER),
             DELIV-DATE(DATE-MASTER);
 CAPACITY:   1800, 504, 112;
END.

 DATA SET      TYPE  FLD  PT  ENTR  MED  MAXIMUM   BLK  BLK  DISC
   NAME              CNT  CT  LGTH  REC  CAPACITY  FAC  LGTH SPACE

DATE-MASTER       A  1    3   3     26   365       19   496  96
CUSTOMER          M  9    1   41    52   201       7    365  96
PRODUCT           M  2    2   14    31   300       16   497  80
SUP-MASTER        M  5    1   31    42   201       12   505  80
INVENTORY         D  6    3   20    32  1800       15   481  128
                INITIAL CAPACITY: 450     INCREMENT ENTRIES:  45
SALES             D  8    4   19    35  1008       14   491  160
                INITIAL CAPACITY: 504     INCREMENT ENTRIES: 112

                         TOTAL DISC SECTORS INCLUDING ROOT:  672

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.
```

Model Program Conventions

The following conventions are used throughout the model.

n	Refers to the option number assigned to each function shown in the main body of the program.
	Indicates an intentional blank.
ACCESS	Indicates the user access mode, or how the database was opened for a particular routine. Note that for this model the user access mode is always 1 (shared modify access).

Buffer_Name **is made up of** *data item name* *data item name* *data item name* *data item name*	Indicates that a data structure is being used. *Buffer_Name* is the name given to the structure, and *data item name* is the name given to the individual parts of that record.
BEGIN MAIN LINE	Indicates the beginning of the main body of the program.
BEGIN ROUTINE	Indicates the beginning of a routine.
CALL *procedure*	Calls the specific TurboIMAGE/XL intrinsic or internal procedure to be used in this particular portion of the routine, and specifies the values for each parameter.
CALLED BY	Refers to that portion of the program which called the routine.
CALLS	Specifies the TurboIMAGE/XL intrinsic (library procedure) or the internal program procedure called by the routine. Note that the intrinsics are listed in the order in which they are used in the routine; as a result, some may be listed more than once. Note that the mode for each call is also shown.
DISPLAY	Displays any specified data.
END LOOP	Indicates the end of an iterative block.
END MAIN LINE	Indicates the end of the main body of the program.
END ROUTINE	Indicates the end of a routine.
ERROR CHECKING	If error found, calls a routine that checks the contents of the status array to determine what action to take.
EXIT LOOP *condition*	Indicates the condition which terminates an iteration.
Not_Used_Parm	Indicates a dummy parameter when a particular intrinsic call does not use a specific parameter.
OBJECTIVE	States the purpose of each routine.
OBTAIN ←user input	Allows the user to enter the required information interactively.
OBTAIN *parameter←value*	Indicates the assignment of a value to a parameter.

NOTE	**For Pascal programmers only: Note that parameters cannot be odd-byte aligned.**

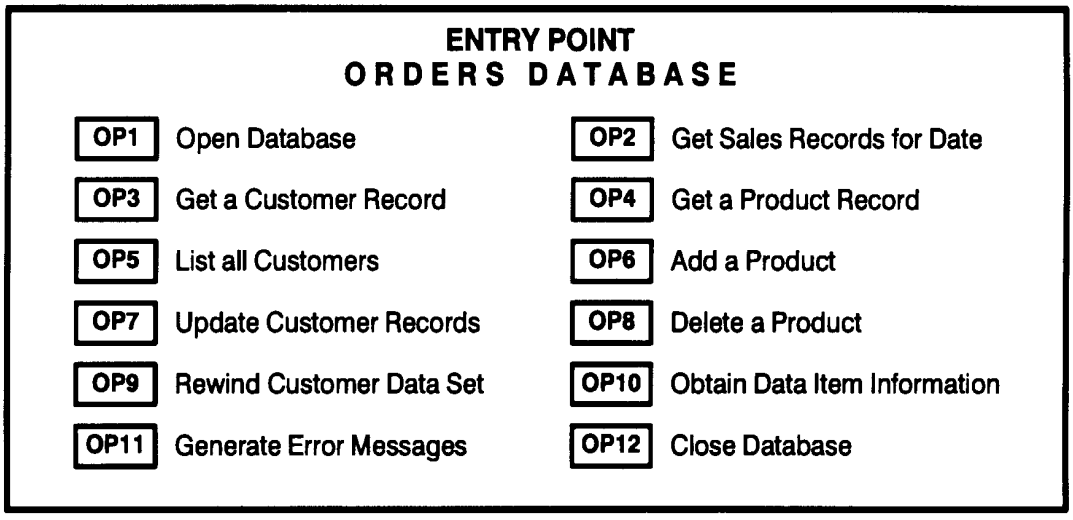
RETURN	Transfers control to the beginning of the iterative block in which it is used.
START LOOP	Indicates the beginning of an iterative block.

ORDERS Database Model Program

Main Body of Program

```
BEGIN MAIN LINE
*    OBJECTIVE:  The main line contains the logic of this application.
*                It displays all functions, accepts a selection,
*                then calls the appropriate routines.  The execution
*                of this program stops after the database is closed.
*
*    ACCESS:     Not applicable.
*
*    CALLED BY:  Not applicable.
*
*    CALLS:      Open_The_Database
*                Get_Sales_For_Date
*                Get_A_Customer_Record
*                Get_A_Product_Record
*                List_All_Customers
*                Add_A_Product
*                Update_A_Customer
*                Delete_A_Product
*                Rewind_Customer_Set
*                Get_Data_Item_Info
*                Get_Error_And_Explain
*                Close_The_Database
     START LOOP
          DISPLAY the list of functions
```

The following illustration depicts one way of displaying the list of functions for this program.



LG200137_032

```
          OBTAIN option  ←  user input

          If option = 1
               Then CALL Open_The_Database
                   and RETURN
          If option = 2
               Then CALL Get_Sales_For_Date
                    and RETURN
          If option = 3
               Then CALL Get_A_Customer_Record
                    and RETURN
          If option = 4
               Then CALL Get_A_Product_Record
                    and RETURN
          If option = 5
               Then CALL List_All_Customers
                     and RETURN
          If option = 6
               Then CALL Add_A_Product
                    and RETURN
          If option = 7
               Then CALL Update_A_Customer
                    and RETURN
          If option = 8
               Then CALL Delete_A_Product
                    and RETURN
          If option = 9
               Then CALL Rewind_Customer_Set
                    and RETURN
          If option = 10
               Then CALL Get_Data_Item_Info
                    and RETURN
          If option = 11
               Then CALL Get_Error_And_Explain
                    and RETURN
          If option = 12
               Then CALL Close_The_Database
                    and RETURN
          EXIT LOOP if option = 12

     END LOOP
END MAIN LINE
```

Opening the Database

```
   (USER SELECTS 1 TO OPEN THE DATABASE)

   ROUTINE:  Open_The_Database
   *    OBJECTIVE:  This routine opens the ORDERS database in mode 1
   *                for this application.
   *
   *    ACCESS:     Mode 1 - Shared Modify Access (SMA) with locking required
   *
   *    CALLED BY:  Main Line
   *
   *    CALLS:      DBOPEN in mode 1 (SMA)
```

```
BEGIN ROUTINE
     OBTAIN DBname       ← "  ORDERS;"
     OBTAIN Password     ← "DO-ALL;"
     CALL DBOPEN (DBname, Password, Mode1_SMA, Status)

          ERROR CHECKING
END ROUTINE
```

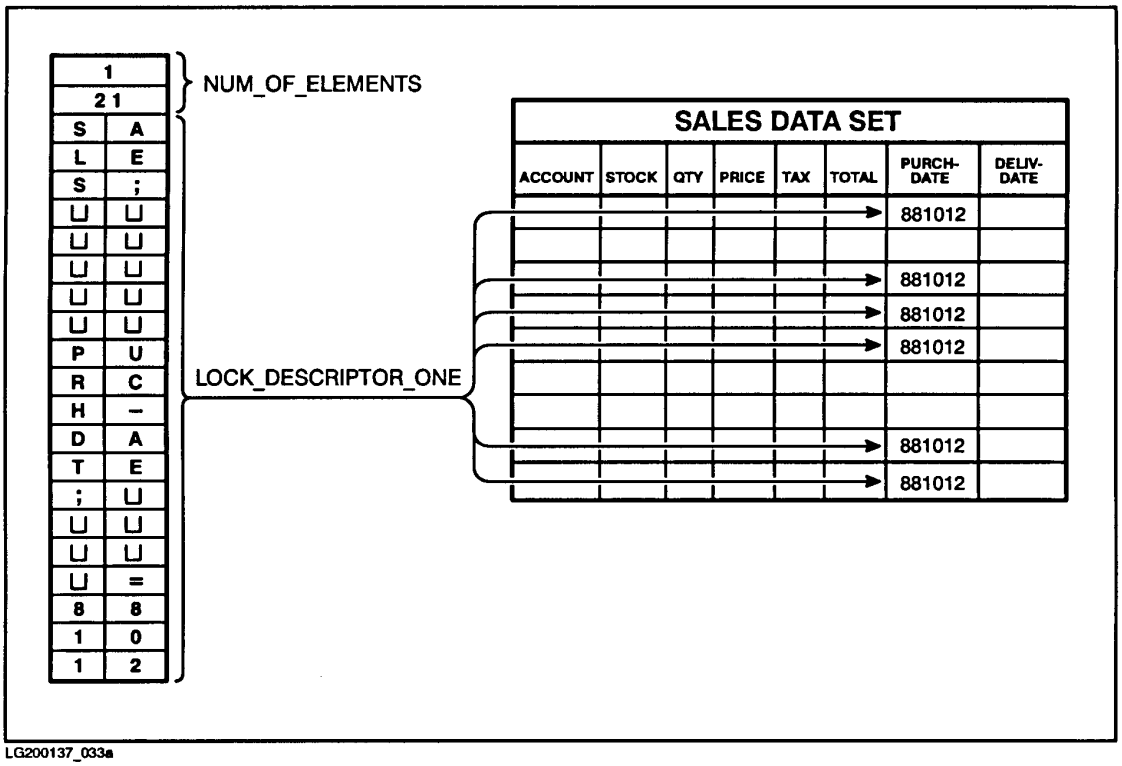
Retrieving All the Records on a Chain (with Item Level Locking)

```
(USER SELECTS 2 TO RETRIEVE SALES DATA)

ROUTINE:  Get_Sales_For_Date
*    OBJECTIVE:  This routine demonstrates chained access, forward chained
*                read, and data item locking.
*
*                The routine retrieves all sales records generated
*                on a particular purchase date.  The value for date is
*                provided by the user and is used as the search item.
*                Due to concurrency issues, a data item lock is acquired
*                on all sales records identified by the date.
*
*    ACCESS:     Mode 1 - Shared Modify Access
*
*    CALLED BY:  Main Line
*
*    CALLS:      DBLOCK in mode 5 (unconditional)
*                DBFIND in mode 1 (chained access)
*                DBGET in mode 5 (forward chained read)
*                DBUNLOCK in mode 1 (unlock)

BEGIN ROUTINE
     Sales_Buffer is made up of:
          Account
          Stock#
          Quantity
          Price
          Tax
          Total
          Purch-Date
          Deliv-Date
     Lock_Descriptor_Sales_Type is made up of:
          Length_Of_Descriptor
          Data_Set_Of_Descriptor
          Data_Item_Of_Descriptor
          Relative_Operator_For_Data_Item
          Value_For_Data_Item
     Lock_Descriptor_Sales_Array_Type is made up of:
          Number_Of_Elements
          Lock_Descriptor_Sales_Type
```

The following illustration shows the layout for a lock descriptor array formats after the actual values have been assigned. Note that the date is stored in YYMMDD format.



LG200137_033a

```
     OBTAIN Number_Of_Elements                ← 1
     OBTAIN Length_Of_Descriptor              ← 21
     OBTAIN Data_Set_Of_Descriptor            ← "SALES;"
     OBTAIN Data_Item_Of_Descriptor           ← "PURCH-DATE;"
     OBTAIN Relative_Operator_For_Data_Item   ← " ="
     OBTAIN Value_For_Data_Item               ← "881012"
     OBTAIN List                              ← "@;"
     OBTAIN Search_Item_Name                  ← "PURCH-DATE;"
     OBTAIN Search_Item_Value                 ← "881012"
     CALL DBLOCK (DBname, Lock_Descriptor_Sales_Array_Type,
          Mode5_Unconditional, Status)

          ERROR CHECKING
     CALL DBFIND (DBname, Sales_Detail, Mode1_Chained_Read,
          Status, Search_Item_Name, Search_Item_Value)

          ERROR CHECKING
     START LOOP
     CALL DBGET (DBname, Sales_Detail, Mode5_Forward, Status,
          List, Sales_Buffer, Not_Used_Parm)

          ERROR CHECKING
     DISPLAY the Sales_Buffer
_____________________________________________________________________
     Account  Stock#  Quantity  Price  Tax  Total  Purch-Date  Deliv-Date
          EXIT LOOP if end of chain
     END LOOP
     CALL DBUNLOCK (DBname, Lock_Desc_Array, Mode1_Unlock, Status)

          ERROR CHECKING
END ROUTINE
```

Retrieving a Data Entry Using a Record Number

```
(USER SELECTS 3 TO RETRIEVE CUSTOMER DATA)

ROUTINE:  Get_A_Customer_Record
*    OBJECTIVE:  This routine demonstrates directed access by retrieving
*                a customer record with a known record number.  Note
*                that the record number is first obtained using a DBGET
*                call, which in this case is a calculated mode 7.
*
*    ACCESS:     Mode 1 - Shared Modify Access
*
*    CALLED BY:  Main Line
*
*    CALLS:      DBGET in mode 7 (calculated read)
*                DBGET in mode 4 (directed read)

BEGIN ROUTINE
     Customer_Buffer is made up of:
          Account
          Last-Name
          First-Name
          Initial
          Street-Address
          City
          State
          Zip
          Credit-Rating
     OBTAIN List            ← "@;"
     OBTAIN Key_Item_Value  ← 315578

     CALL DBGET (DBname, Customer_Master, Mode7_Calculated, Status, List,
          Customer_Buffer, Key_Item_Value)

          ERROR CHECKING
     OBTAIN Record_Num ← Status [element 3]

     CALL DBGET (DBname, Customer_Master, Mode4_Directed, Status, List,
          Customer_Buffer, Record_Num)

          ERROR CHECKING
     DISPLAY the Customer_Buffer
_____________________________________________________________________
     Account  Last-Name  First-Name  Initial  Street-Address  City. . .
END ROUTINE
```

Retrieving Master Data Using a Key Value

```
(USER SELECTS 4 TO RETRIEVE PRODUCT DATA)

ROUTINE:Get_A_Product_Record
*    OBJECTIVE:  This routine demonstrates calculated access by
*                retrieving a product record from a master data
*                set based on a user-defined key item value.
*
*    ACCESS:     Mode 1 - Shared Modify Access
```

```
*
*    CALLED BY:  Main Line
*
*    CALLS:      DBGET in mode 7 (calculated read)

BEGIN ROUTINE
     Product_Buffer is made up of:
          Stock#
          Description
     OBTAIN List            ← "@;"
     OBTAIN Key_Item_Value  ← "STK30040"
     CALL DBGET (DBname, Product_Master, Mode7_Calculated, Status, List,
          Product_Buffer, Key_Item_Value)

          ERROR CHECKING
     DISPLAY the Product_Buffer
_____________________________________________________________________
     Stock#  Description
END ROUTINE
```

Retrieving Data Serially (with Set Level Locking)

```
(USER SELECTS 5 TO RETRIEVE CUSTOMER DATA)

ROUTINE:  List_All_Customers
*    OBJECTIVE:  This routine demonstrates serial access by listing
*                all customer records.  For the sake of consistency,
*                the data set is locked for exclusive access,
*                then the data is read serially.
*
*    ACCESS:     Mode 1 - Shared Modify Access
*
*    CALLED BY:  Main Line
*
*    CALLS:      DBLOCK in mode 3 (unconditional)
*                DBGET in mode 2 (forward read)
*                DBUNLOCK in mode 1 (unlock)

BEGIN ROUTINE
     Customer_Buffer is made up of:
          Account
          Last-Name
          First-Name
          Initial
          Street-Address
          City
          State
          Zip
          Credit-Rating
     CALL DBLOCK (DBname, Customer_Master, Mode3_Unconditional, Status)

          ERROR CHECKING
     OBTAIN List  ← "Account, Last-Name, First-Name, Initial;"
     START LOOP<
     CALL DBGET (DBname, Customer_Master, Mode2_Forward, Status, List,
          Customer_Buffer, Not_Used_Parm)
```

```
          ERROR CHECKING
     DISPLAY List
_____________________________________________________________________
     Account  Last-Name  First-Name  Initial
          EXIT LOOP if first word of Status buffer <gt; 0

     END LOOP
     CALL DBUNLOCK (DBname, Customer_Master, Mode1_Unlock, Status)

          ERROR CHECKING
END ROUTINE
```

Adding an Entry

```
(USER SELECTS 6 TO DO A PUT)

ROUTINE:  Add_A_Product
*    OBJECTIVE:  This routine adds one entry to the Product master data
*                set.  After obtaining user input, the data set is locked
*                for exclusive access.  A transaction starts, and
*                new values are added by a call to DBPUT.  At the end
*                of the routine, the transaction is ended and locks
*                are released.
*
*    ACCESS:     Mode 1 - Shared Modify Access
*
*    CALLED BY:  Main Line
*
*    CALLS:      DBLOCK in mode 3 (unconditional)
*                DBBEGIN in mode 1 (transaction begin)
*                DBPUT in mode 1 (put)
*                DBEND in mode 1 (transaction end)
*                DBUNLOCK in mode 1 (unlock)

BEGIN ROUTINE
     Product_Buffer is made up of:
         Stock#
         Description
     OBTAIN Stock#       ← user input
     OBTAIN Description  ← user input
     OBTAIN List         ← "@;"
     CALL DBLOCK (DBname, Product_Master, Mode3_Unconditional, Status)

         ERROR CHECKING
     OBTAIN Text     ← "Add entry to Product set Begin  "
     OBTAIN Textlen  ← 16

     CALL DBBEGIN (DBname, Text, Mode1_Xbegin, Status, TextLen)

          ERROR CHECKING
     CALL DBPUT (DBname, Product_Master, Mode1_Put, Status, List,
     Product_Buffer)

          ERROR CHECKING
     OBTAIN Text     ← "Add entry to Product set End"
```

```
     OBTAIN Textlen  ← 14

     CALL DBEND (DBname, Text, Mode1_Xend, Status, Textlen)

         ERROR CHECKING
     CALL DBUNLOCK (DBname, Product_Master, Mode1_Unlock, Status)

         ERROR CHECKING
END ROUTINE
```

Updating an Entry

```
(USER SELECTS 7 TO DO AN UPDATE)

ROUTINE:  Update_A_Customer
*    OBJECTIVE:  This routine updates a customer record interactively.
*                Updating is achieved by using the key item value to
*                locate the proper entry.  It then displays the contents
*                to be updated.
*
*                To perform the actual update, a TurboIMAGE/XL
*                transaction is started.  The entry is retrieved using
*                a re-read mode, and an item level lock is obtained using
*                the search value.  A contents check of the new values
*                is done against the old values. If the contents have
*                changed, the user can choose to abort the process.  Otherwise,
*                the transaction proceeds and the update takes place.
*
*    ACCESS:     Mode 1 - Shared Modify Access
*
*    CALLED BY:  Main Line
*
*    CALLS:      DBGET in mode 7 (calculated)
*                DBLOCK in mode 5 (unconditional)
*                DBBEGIN in mode 1 (transaction begin)
*                DBGET in mode 1 (re-read)
*                DBUPDATE in mode 1 (update)
*                DBEND in mode 1 (transaction end)
*                DBUNLOCK in mode 1 (unlock)

BEGIN ROUTINE
     Customer_Buffer,
     Customer_Buffer_New, and
     Customer_Buffer_Old are made up of:
          Account
          Last-Name
          First-Name
          Initial
          Street-Address
          City
          State
          Zip
          Credit-Rating
     Lock_Descriptor_Customer_Type is made up of:
          Length_Of_Descriptor
          Data_Set_Of_Descriptor
          Data_Item_Of_Descriptor
```

```
          Relative_Operator_For_Data_Item
          Value_For_Data_Item
     Lock_Descriptor_Customer_Array_Type is made up of:
          Number_Of_Elements
          Lock_Descriptor_Customer_Type
     OBTAIN List            ← "@;"
     OBTAIN Key_Item_Value  ← user input

     CALL DBGET (DBname, Customer_Master, Mode7_Calculated, Status, List,
          Customer_Buffer, Key_Item_Value)

          ERROR CHECKING
     Customer_Buffer_Old  ← Customer_Buffer
     DISPLAY Customer_Buffer
     OBTAIN Customer_Buffer_New   ← user input

     OBTAIN Number_Of_Elements               ← 1
     OBTAIN Length_Of_Descriptor             ← 22
     OBTAIN Data_Set_Of_Descriptor           ← "CUSTOMER;"
     OBTAIN Data_Item_Of_Descriptor          ← "ACCOUNT;"
     OBTAIN Relative_Operator_For_Data_Item  ← " ="
     OBTAIN Value_For_Data_Item              ← Key_Item_Value
     CALL DBLOCK (DBname, Lock_Descriptor_Customer_Array_Type,
          Mode5_Unconditional, Status)

          ERROR CHECKING
     OBTAIN Text     ← "Update entry on Customer set Begin  "
     OBTAIN Textlen  ← 18

     CALL DBBEGIN (DBname, Text, Mode1_Xbegin, Status, Textlen)

          ERROR CHECKING
     CALL DBGET (DBname, Customer_Master, Mode1_Reread, Status, List,
          Customer_Buffer, Not_Used_Parm)

          ERROR CHECKING
               If Customer_Buffer is the same as Customer_Buffer_Old
                    Then continue
               Otherwise
                    Let the user know that the entry has been modified by
                    another user, end the transaction, and release the locks.
     CALL DBUPDATE (DBname, Customer_Master, Mode1_Update, Status, List,
          Customer_Buffer_New)

          ERROR CHECKING
     OBTAIN Text     ← "Update entry on Customer set End"
     OBTAIN Textlen  ← 16

     CALL DBEND (DBname, Text, Mode1_Xend, Status, Textlen)

          ERROR CHECKING
     CALL DBUNLOCK (DBname, Customer_Master, Mode1_Unlock, Status)

          ERROR CHECKING
END ROUTINE
```

Deleting an Entry

```
(USER SELECTS 8 TO DELETE AN ENTRY)

ROUTINE:  Delete_A_Product
*    OBJECTIVE:  This routine deletes an entry from the Product master
*                data set.  The entry is specified by its key item value.
*                Identifying the entry and deleting it are preceded by
*                calls to DBLOCK and DBBEGIN to obtain locks and to start
*                a new transaction.
*
*                When the entry is located, the deletion of the record
*                at the current record pointer is done by a call to
*                DBDELETE.
*
*                The completion of a transaction is achieved by a call
*                to DBEND, and outstanding locks on this data set are
*                released by a call to DBUNLOCK.
*
*    ACCESS:     Mode 1 - Shared Modify Access
*
*    CALLED BY:  Main Line
*
*    CALLS:      DBLOCK in mode 3 (unconditional)
*                DBBEGIN in mode 1 (transaction begin)
*                DBGET in mode 7 (calculated read)
*                DBDELETE in mode 1 (delete)
*                DBEND in mode 1 (transaction end)
*                DBUNLOCK in mode 1 (unlock)

BEGIN ROUTINE
     Product_Buffer is made up of:
          Stock#
          Description
     CALL DBLOCK (DBname, Product_Master, Mode3_Unconditional, Status)

          ERROR CHECKING
          OBTAIN List           ← "@;"
          OBTAIN Text           ← "Delete entry from Product set Begin "
          OBTAIN Textlen        ← 18
          OBTAIN Key_Item_Value ← "STK30040"
     CALL DBBEGIN (DBname, Text, Mode1_Xbegin, Status, Textlen)

          ERROR CHECKING
     CALL DBGET (DBname, Product_Master, Mode7_Calculated, Status, List,
          Product_Buffer, Key_Item_Value)

          ERROR CHECKING
     CALL DBDELETE (DBname, Product_Master, Mode1_Delete, Status)

          ERROR CHECKING
     OBTAIN Text     ← "Delete entry from Product set End   "
     OBTAIN Textlen  ← 18

     CALL DBEND (DBname, Text, Mode1_Xend, Status, Textlen)

          ERROR CHECKING
     CALL DBUNLOCK (DBname, Product_Master, Mode1_Unlock, Status)
```

```
          ERROR CHECKING
END ROUTINE
```

Rewinding a Data Set

```
(USER SELECTS 9 TO REWIND A DATA SET)

ROUTINE:  Rewind_Customer_Set
*    OBJECTIVE:  This routine rewinds the customer data set by calling
*                DBCLOSE in mode 2.
*
*    ACCESS:     Mode 1 - Shared Modify Access
*
*    CALLED BY:  Main Line
*
*    CALLS:      DBCLOSE in mode 2 (rewind)

BEGIN ROUTINE
     CALL DBCLOSE (DBname, Customer_Master, Mode2_Rewind, Status)

          ERROR CHECKING
END ROUTINE
```

Obtaining Database Information

```
(USER SELECTS 10 TO OBTAIN INFORMATION ABOUT A DATA ITEM)

ROUTINE:  Get_Data_Item_Info
*    OBJECTIVE:  This routine obtains information about a data item
*                by calling DBINFO in mode 102.
*
*    ACCESS:     Mode 1 - Shared Modify Access
*
*    CALLED BY:  Main Line
*
*    CALLS:      DBINFO in mode 102 (item access)

BEGIN ROUTINE
     DBINFO_Buffer is made up of:
          Data_Item_Name
          Data_Type
          Sub_Item_Length
          Sub_Item_Count
     OBTAIN Data_Item_Name  ← "PURCH-DATE;"
     CALL DBINFO (DBname, Data_Item_Name, Mode102_Item, Status, DBINFO_Buffer)

          ERROR CHECKING
     DISPLAY the DBINFO_Buffer
_____________________________________________________________________________
     Data_Item_Name  Data_Type  Sub_Item_Length  Sub_Item_Count
END ROUTINE
```

Obtaining Error Messages and Explanations

```
(USER SELECTS 11 TO OBTAIN ERROR MESSAGES AND ADDITIONAL
ERROR-RELATED INFORMATION)

ROUTINE:Get_Error_And_Explain
*    OBJECTIVE:  This routine generates an error message, corresponding
*                to the existing value in the first word of the status
*                parameter, by calling the DBERROR intrinsic.
*                Additionally, the routine generates a description
*                regarding the outstanding error message by calling the
*                DBEXPLAIN intrinsic.
*
*    ACCESS:     Mode 1 - Shared Modify Access
*
*    CALLED BY:  Main Line
*
*    CALLS:      DBERROR
*                DBEXPLAIN
BEGIN ROUTINE
     Error_Buffer is made up of:
          Error_Message
     CALL DBERROR (Status, Error_Buffer, Error_Length)

     DISPLAY the Error_Buffer
_____________________________________________________________________
     Error_Message
     CALL DBEXPLAIN (Status)

END ROUTINE
```

Closing the Database

```
(USER SELECTS 12 TO CLOSE THE DATABASE)

ROUTINE:  Close_The_Database
*    OBJECTIVE:  This routine closes the ORDERS database by calling
*                the DBCLOSE intrinsic.
*
*    ACCESS:     Mode 1 - Shared Modify Access
*
*    CALLED BY:  Main Line
*
*    CALLS:      DBCLOSE in mode 1 (close)

BEGIN ROUTINE
     CALL DBCLOSE (DBname, Not_Used_Parm, Mode1_Close, Status)

          ERROR CHECKING
END ROUTINE
```

C

This section shows, in C, portions of the model program presented at the beginning of this chapter. The examples perform specific tasks to illustrate the use of TurboIMAGE/XL intrinsics. The C example does not illustrate everything in the COBOL example. Some blocks of code may be appropriate only if expanded to a full program.

Data items are defined at the beginning of the sample program. TurboIMAGE/XL intrinsics must be declared for C as external procedures. The procedure name is identified by the word "Intrinsic."

Type declarations declare names for data structure forms that will be used in allocating variables. Variable declarations allocate the variables of the program. Variables are defined with precise types or forms. C string literals are delimited with double quotation marks (" "). Field and record names are separated with a dot (.) when referenced (for example, base_name.baseid).

NOTE Because the Schema Processor, DBSCHEMA, upshifts alphabetic characters, programs must specify data set and data item names in all uppercase characters. Take note of this because C does not require that you use uppercase characters.

For information on TurboIMAGE/XL data item lengths and type designators, refer to chapter 3. Tables 3-2 and 3-3 show the TurboIMAGE/XL type designators, sub-item lengths, and data types typically used to process them in C.

NOTE All parameters must be on halfword boundaries.

Defining Data Types, Variables, and Intrinsics

The following is part of the C example program; it defines type declarations, variable declarations, and TurboIMAGE/XL intrinsics.

```
#pragma list off
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#pragma list on
/*         Define all TurboIMAGE/XL procedure calls that        */
/*         will be used in your application program             */
#pragma intrinsic DBBEGIN, DBEND, DBOPEN, DBCLOSE, DBGET, DBPUT,DBFIND, DBINFO
#pragma intrinsic DBEXPLAIN, DBERROR, DBDELETE, DBUPDATE, DBLOCK,DBUNLOCK

/*         Define all your TurboIMAGE/XL constants              */
#define End_Of_Chain 15       /* For DBGET Mode 5 */
#define End_Of_Data_Set 11    /* For DBGET Mode 2 */
#define No_Chain_Head 17      /* For DBFIND */
#define No_Such_Entry 17      /* For DBGET Mode 7 */
#define Entry_Has_No_Data 17  /* For DBGET Mode 4 */
```

```
short   DBname[6]
        Password[4]
        Sales_D_Set[4];
char
        *Purch_Date   = "PURCH-DATE;",
        *Equal_Op     = " =",
        *Item_List    ="ACCOUNT,STOCK#,PRICE,TAX,TOTAL,PURCH-DATE;";

/*         Define all your global variables.                    */
struct
     Database_Status_Type   { short Condition;
                              short Length;
                              int   Record_Number;
                              int   Chain_Count;
                              int   Back_Pointer;
                              int   Forward_Pointer;
                             } Status;
struct
     Sales_Data_Set_Type     {int   Account_Number;
                              char  Stock_Number[8];
                              int   Price;
                              int   Tax;
                              int   Total;
                              char  Purch_Date[6];
                             };
struct
     Lock_Descriptor_Type    {short Num_Of_Elements;
                              short Length_Of_Descriptor;
                              char  Data_Set_Of_Descriptor[16];
                              char  Data_Item_Of_Descriptor[16];
                              char  Relop_For_Data_Item[2];
                              char  Value_For_Data_Item[6];
                             };
short Mode;
```

Main Body of Program

```
/*     Beginning of the main program                 */
main()
{
/*     Initialize the database and set information   */
  strcpy ((char *)DBname,"  ORDERS; ");
  strcpy ((char *)Password,"DO-ALL;");
  strcpy ((char *)Sales_D_Set,"SALES;");

  Open_The_Database();
  Get_Sales_For_Date();
  exit (0);
}
```

Obtaining Error Messages and Explanations

The following paragraph implements the GET-ERROR-AND-EXPLAIN routine of the sample program. This routine calls DBEXPLAIN and DBERROR. DBEXPLAIN interprets the contents of the status parameter and prints a message on $STDLIST. DBERROR returns a message in the ERROR-BUFFER, explaining the condition code returned by TurboIMAGE/XL. At the end the routine, users can choose to abort or continue the execution of this program.

```
/*     Beginning of subroutines                    */
Get_Error_And_Explain()
{
/*
      Access   :     Mode 1 - Shared Modified Access
                              The Orders database was opened in mode 1


      Called by:     Open_The_Database
                     Get_Sales_For_Date
                     Get_A_Product_Record
                     List_All_Customers
                     Add_A_Product
                     Update_A_Customer
                     Delete_A_Product
                     Rewind_Customer_Set
                     Get_Data_Item_Info
                     Close_The_Database
      Calls    :     DBERROR
                     DBEXPLAIN
*/
  short        Error_Buffer[80];
  short        Error_Length;
  int          Answer;

  DBERROR(&Status,Error_Buffer,&Error_Length);

  printf("-----------------------------------------\n");
  printf("%.*s\n",Error_Length, (char *)Error_Buffer);
  printf("-----------------------------------------\n");

  DBEXPLAIN(&Status);

  Answer=0;
  printf("---Enter, <1> to ABORT..., <2> to Continue >\n");
  scanf("%d",&Answer);
  if (Answer != 1)
     printf(" Continuing .........\n");
  else
     exit(0);
}
```

Opening the Database

This paragraph implements the OPEN-THE-DATABASE routine of the sample program in C. All required values, such as the password, are defined in the "static char" section of the program. Note that the password DO-ALL establishes user class number 18. The password DO-ALL is followed by a semicolon because it is less than eight characters long; a blank can be substituted for the semicolon. OPEN-THE-DATABASE uses open mode 1, which is the shared modify access mode. Error trapping is done by referring all non-zero conditions to the GET-ERROR-AND-EXPLAIN procedure.

```
Open_The_Database()
{

/*

        ACCESS   :  Mode 1 - Shared Modify Access (SMA) with locking required

        Called By:  Main Line

        Calls    :  DBOPEN in mode 1 (SMA)
                    Get_Error_And_Explain
*/

  Mode =1;

  DBOPEN(DBname,Password,&Mode,&Status);
  if (Status.Condition != 0)
     Get_Error_And_Explain();
}
```

Retrieving All the Records on a Chain (with Item Level Locking)

This paragraph implements the GET-SALES-FOR-DATE routine of the sample program. Chain access is achieved using a call to DBFIND to determine the location of the first and last entries in the chain. The search item used for this call is PURCH-DATE. An item level lock is obtained on the value of the search item before the DBFIND call. After that, individual chain items are retrieved, until the end of the chain is encountered. This is done using multiple calls to the DBGET procedure.

The routine traps two exceptional conditions:

1. Status condition 17 from the DBFIND call, indicating that the chain head cannot be located.
2. Status 15 from the DBGET call, indicating the end of the chain.

The status interpretation routine permits you to either abort or continue with the execution after viewing all error messages.

```
Get_Sales_For_Date()
/*
        ACCESS   :  Mode 1 - Shared Modify Access
                               The Orders database was opened in mode 1

        Called By:  Main Line
        Calls  :  DBLOCK in mode 5 (unconditional item level locking)
```

```
                    DBFIND in mode 1 (chained access)
                    DBGET  in mode 5 (forward chained read)
                    DBUNLOCK in mode 1 (unlock)
                    Get_Error_And_Explain
   */
   {
     struct  Lock_Descriptor_Type       Lock_Descriptor;
     struct  Sales_Data_Set_Type        Sales_Buffer;
     short                              Search_Item_Value[3];
     short                              Search_Item_Name[8];
     short                              List[40];
     short                              Dummy;
     size_t                             srch_len = 6;

     /*  Prepare the lock descriptor buffer for obtaining item
         level locks on the Sales data set.
     */
     Lock_Descriptor.Num_Of_Elements        = 1;
     Lock_Descriptor.Length_Of_Descriptor   = 21;
     strcpy(Lock_Descriptor.Data_Set_Of_Descriptor,(char *)Sales_D_Set);
     strcpy(Lock_Descriptor.Data_Item_Of_Descriptor,(char *)Purch_Date);
     Lock_Descriptor.Relop_For_Data_Item[0] = Equal_Op[0];
     Lock_Descriptor.Relop_For_Data_Item[1] = Equal_Op[1];

     printf("Enter The Date of Purchase as (YYMMDD) >>> \n");
     scanf("%6c", (char *)Search_Item_Value);

     /*  Request item level locks (mode 5) */
     Mode  = 5;
     /*  Append the user's input to the lock descriptor buffer */
     strncpy(Lock_Descriptor.Value_For_Data_Item,
           (char *)Search_Item_Value,srch_len);

     /*  Place item level locks on all entries identified by
       the value in the Search_Item_Value
     */
     DBLOCK(DBname,&Lock_Descriptor,&Mode,&Status);
         if (Status.Condition != 0)
        Get_Error_And_Explain();

     Mode  = 1;
     strcpy((char *)Search_Item_Name, Purch_Date);
     /*  Locate the chain identified by the value in the
         Search_Item_Value
     */
     DBFIND(DBname,Sales_D_Set,&Mode,&Status,
            Search_Item_Name, Search_Item_Value);
     if (Status.Condition != 0)
       {
        if (Status.Condition == No_Chain_Head)
           {
            printf("*************************************\n");
            printf("* No Such Entry in the Sales Dataset  *\n");
            printf("* Please Try Again.                   *\n");
            printf("*************************************\n");
           }
        else
           Get_Error_And_Explain();
```

```
    }
  else
    {
     /*  Start retrieving all records in the current chain */
     printf("\n");
     printf("Acct-Number Stock_Number   Price   Tax  Total Purch-Date \n");
     printf("-------------------------------------------------------\n");

     Mode  = 5;
     strcpy((char *)List,Item_List);
     while (Status.Condition != End_Of_Chain)
      {
       DBGET(DBname,Sales_D_Set,&Mode,&Status,List,&Sales_Buffer,
             &Dummy);

       if (Status.Condition == 0)
        {
          printf("\n");
          printf("%11d",Sales_Buffer.Account_Number);
          printf("%13.8s",Sales_Buffer.Stock_Number);
          printf("%8d",Sales_Buffer.Price);
          printf("%6d",Sales_Buffer.Tax);
          printf("%7d",Sales_Buffer.Total);
          printf("%12.6s",Sales_Buffer.Purch_Date);
        }
        else
        {
          if (Status.Condition == End_Of_Chain)
          {
            printf("\n\n\n");
            printf ("----> End Of Chain.\n");
          }
          else
            Get_Error_And_Explain();
        }
     } /*  while  */
   } /*  else  */

  /*  Release all locks acquired at the beginning of the process */
  Mode = 1;                                  = 1;
  DBUNLOCK (DBname,Sales_D_Set,&Mode,&Status);
  if (Status.Condition != 0)
      Get_Error_And_Explain();
}
```

COBOL II

The model program presented at the beginning of this chapter is now shown here in COBOL II. The program performs specific tasks to illustrate the use of TurboIMAGE/XL intrinsics. Note that the code, although broken out by task, can be combined to make up a complete, executable program.

Data items are defined at the beginning of the sample program. The parameters for the TurboIMAGE/XL intrinsics are defined in the data division, and their values are defined when the procedure is called or, in some cases, after it is executed.

The database identifier is described as follows:

```
01  DBNAME.
    05  BASEID                          PIC  X(02).
    05  BASENAME                        PIC  X(06).
    05  TERMINATOR                      PIC  X(02).
```

To access a database catalogued in a group other than the user's log-on group, the database name must be followed by a period and the group name, for example, `ORDERS.GROUPX`. If the database 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`.

Once the database has been opened and the database identifier has been moved to the first halfword of the element (as shown in "Opening the Database"), it remains the same for all subsequent calls illustrated.

The status record is defined in the same way for all tasks but its content varies depending upon which procedure is called and the results of that procedure. The status record is defined as follows:

```
01  STATUS1.
    05  CONDITION          PIC S9(4)  COMP.
    05  LENGTH1            PIC S9(4)  COMP.
    05  RECORD-NUMBER      PIC S9(9)  COMP.
    05  CHAIN-COUNT        PIC S9(9)  COMP.
    05  BACK-POINTER       PIC S9(9)  COMP.
    05  FORWARD-POINTER    PIC S9(9)  COMP.
```

NOT-USED-PARM appears as a reminder when a parameter is not used by a procedure performing the task being illustrated. NOT-USED-PARM is defined in this program as follows:

```
01  NOT-USED-PARM-16       PIC S9(4)  COMP.
01  NOT-USED-PARM-32       PIC S9(9)  COMP.
```

NOTE Because the Schema Processor, DBSCHEMA, upshifts alphabetic characters, programs must specify data set and data item names in all uppercase characters. Take note of this because COBOL II does not require that you use uppercase characters.

For information on TurboIMAGE/XL data item lengths and type designators, refer to

chapter 3. Tables 3-2 and 3-3 show the TurboIMAGE/XL type designators, sub-item lengths, and data types typically used to process them in COBOL II.

NOTE All parameters must be on halfword boundaries.

Defining Data Types, Variables, and Intrinsics

The following is part of the COBOL II program; it defines all the data items and records.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  RECEIVE.
DATE-COMPILED.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
  SYMBOLIC CHARACTERS CLEAR, SCREEN IS 28, 86.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  END-OF-CHAIN                        PIC  S9(4)  COMP VALUE 15.
01  END-OF-DATA-SET                     PIC  S9(4)  COMP VALUE 11.
01  NO-CHAIN-HEAD                       PIC  S9(4)  COMP VALUE 17.
01  NO-SUCH-ENTRY                       PIC  S9(4)  COMP VALUE 17.
01  ENTRY-HAS-NO-DATA                   PIC  S9(4)  COMP VALUE 17.
01  DBNAME.
    05  BASEID                          PIC  X(02).
    05  BASENAME                        PIC  X(06).
    05  TERMINATOR                      PIC  X(02).
01  PASSWORD                            PIC  X(10).
01  STATUS1.
    05  CONDITION                       PIC  S9(4)  COMP.
    05  LENGTH1                         PIC  S9(4)  COMP.
    05  RECORD-NUMBER                   PIC  S9(9)  COMP.
    05  CHAIN-COUNT                     PIC  S9(9)  COMP.
    05  BACK-POINTER                    PIC  S9(9)  COMP.
    05  FORWARD-POINTER                 PIC  S9(9)  COMP.
01  OPTION                              PIC  S9(4)  COMP.
01  DB-MODE                             PIC  S9(4)  COMP.
01  LIST                                PIC  X(80).
01  ERROR-BUFFER                        PIC  X(80).
01  ERROR-LENGTH                        PIC  S9(9)  COMP.
01  ANSWER                              PIC  S9(4)  COMP.
01  LOCK-DESCRIPTOR-ARRAY.
    05  NUM-OF-ELEMENTS                 PIC  S9(4) COMP.
    05  LOCK-DESCRIPTOR-SALES.
        10  LENGTH-OF-DESCRIPTOR        PIC  S9(4)  COMP.
        10  DATA-SET-OF-DESCRIPTOR      PIC  X(16).
        10  DATA-ITEM-OF-DESCRIPTOR     PIC  X(16).
        10  RELOP-FOR-DATA-ITEM         PIC  X(02).
        10  VALUE-FOR-DATA-ITEM         PIC  X(6).
        10  NUM-VALUE-FOR-DATA-ITEM REDEFINES
            VALUE-FOR-DATA-ITEM          PIC  S9(9)  COMP.
01  SALES-DETAIL                        PIC  X(16).
01  SEARCH-ITEM-NAME                    PIC  X(16).
01  SEARCH-ITEM-VALUE                   PIC  X(6).
01  SALES-BUFFER.
```

```
    05  ACCOUNT-NUMBER                 PIC  S9(9)  COMP.
    05  STOCK-NUMBER                   PIC  X(8).
    05  QUANTITY                       PIC  S9(4)  COMP.
    05  PRICE                          PIC  S9(9)  COMP.
    05  TAX                            PIC  S9(9)  COMP.
    05  TOTAL                          PIC  S9(9)  COMP.
    05  PURCH-DATE                     PIC  X(6).
    05  DELIV-DATE                     PIC  X(6).
01  SALES-BUFFER-OUT.
    05  ACCOUNT-NUMBER-OUT             PIC  Z(9)9.
    05  STOCK-NUMBER-OUT               PIC  B(7)X(8).
    05  QUANTITY-OUT                   PIC  Z(5)9.
    05  PRICE-OUT                      PIC  Z(6)9.
    05  TAX-OUT                        PIC  Z(4)9.
    05  TOTAL-OUT                      PIC  Z(6)9.
    05  PURCH-DATE-OUT                 PIC  B(6)X(6).
    05  DELIV-DATE-OUT                 PIC  B(6)X(6).
01  SALES-BUFFER-HEADER.
    05  ACCOUNT-NUMBER-HEAD            PIC  X(13)
            VALUE "Acct-Number  ".
    05  STOCK-NUMBER-HEAD              PIC  X(15)
            VALUE "Stock-Number   ".
    05  QUANTITY-HEAD                  PIC  X(05)
            VALUE "QTY  ".
    05  PRICE-HEAD                     PIC  X(07)
            VALUE "Price  ".
    05  TAX-HEAD                       PIC  X(06)
            VALUE "Tax   ".
    05  TOTAL-HEAD                     PIC  X(07)
            VALUE "Total  ".
    05  PURCH-DATE-HEAD                PIC  X(13)
            VALUE "Purch-Date   ".
    05  DELIV-DATE-HEAD                PIC  X(14)
            VALUE "Delive-Date   ".
01  LINE-HEADER.
    05                                  PIC  X(40)
            VALUE "----------------------------------------".
    05                                  PIC  X(38)
            VALUE "--------------------------------------".
01  NOT-USED-PARM-16                   PIC  S9(4)  COMP.
01  NOT-USED-PARM-32                   PIC  S9(9)  COMP.
01  FOUND-VALUE                        PIC  S9(4)  COMP.
    88  NOT-FOUND    VALUE 0.
    88  FOUND        VALUE 1.
01  CUSTOMER-MASTER                    PIC  X(16).
01  CUSTOMER-BUFFER.
    05  ACCOUNT-NUMBER                 PIC  S9(9)  COMP.
    05  LAST-NAME                      PIC  X(16).
    05  FIRST-NAME                     PIC  X(10).
    05  INITIAL1                       PIC  X(02).
    05  STREET-ADDRESS                 PIC  X(26).
    05  CITY                           PIC  X(12).
    05  STATE                          PIC  X(02).
    05  ZIP                            PIC  X(06).
    05  CREDIT-RATING                  PIC  X(08).
01  CUSTOMER-BUFFER-OUT.
    05  ACCOUNT-NUMBER-CUST-OUT        PIC  9(6).
    05  FIRST-NAME-CUST-OUT            PIC  X(15) JUST RIGHT.
```

```
    05                                  PIC  X.
    05  INITIAL1-CUST-OUT               PIC  X(02).
    05  LAST-NAME-CUST-OUT              PIC  X(16) JUST RIGHT.
01  KEY-ITEM-VALUE-PRODUCT              PIC  X(08).
01  KEY-ITEM-VALUE                      PIC  S9(9)  COMP.
01  LIST-NO-ITEM                        PIC  S9(9)  COMP.
01  SAVED-RECORD-NUMBER                 PIC  S9(9)  COMP.
01  PRODUCT-MASTER                      PIC  X(16).
01  PRODUCT-BUFFER.
    05  STOCK-NUMBER                    PIC  X(08).
    05  DESCRIPTION                     PIC  X(20).

01  DONE-VALUE                          PIC  S9(4)  COMP.
    88  NOT-DONE     VALUE 0.
    88  DONE         VALUE 1.
01  TEXT1                               PIC  X(80).
01  TEXTLEN                             PIC  S9(9) COMP.
01  CUSTOMER-BUFFER-NEW.
    05  ACCOUNT-NUMBER                  PIC  S9(9)  COMP.
    05  LAST-NAME                       PIC  X(16).
    05  FIRST-NAME                      PIC  X(10).
    05  INITIAL1                        PIC  X(02).
    05  STREET-ADDRESS                  PIC  X(26).
    05  CITY                            PIC  X(12).
    05  STATE                           PIC  X(02).
    05  ZIP                             PIC  X(06).
    05  CREDIT-RATING                   PIC  X(08).
01  CUSTOMER-BUFFER-OLD.
    05  ACCOUNT-NUMBER                  PIC  S9(9)  COMP.
    05  LAST-NAME                       PIC  X(16).
    05  FIRST-NAME                      PIC  X(10).
    05  INITIAL1                        PIC  X(02).
    05  STREET-ADDRESS                  PIC  X(26).
    05  CITY                            PIC  X(12).
    05  STATE                           PIC  X(02).
    05  ZIP                             PIC  X(06).
    05  CREDIT-RATING                   PIC  X(08).
01  DATA-ITEM-NAME-IN                   PIC  X(16).
01  M-102-BUFFER.
    05  DATA-ITEM-NAME                  PIC  X(16).
    05  DATA-ITEM-TYPE                  PIC  X(02).
    05  DATA-ITEM-LENGTH                PIC  S9(4)  COMP.
    05  DATA-ITEM-COUNT                 PIC  S9(4)  COMP.
    05  NOT-USED-ITEM                   PIC  S9(4)  COMP.
01  MENU.
    05  MENU-LINE-1                     PIC  X(62)  VALUE
"------------------------------------------------------------".
    05  MENU-LINE-2                     PIC  X(62)  VALUE
"|                                                          |".
    05  MENU-LINE-3                     PIC  X(62)  VALUE
"|                      Entry Point                         |".
    05  MENU-LINE-4                     PIC  X(62)  VALUE
"|              O R D E R S   D A T A B A S E               |".
    05  MENU-LINE-5                     PIC  X(62)  VALUE
"|----------------------------------------------------------|".
    05  MENU-LINE-6                     PIC  X(62)  VALUE
"| 1)OPEN DATABASE             2)GET SALES RECORD FOR DATE  |".
    05  MENU-LINE-7                     PIC  X(62)  VALUE
```

```
"| 3)GET A CUSTOMER RECORD      4)GET A PRODUCT RECORD        |".
    05  MENU-LINE-8                   PIC  X(62)  VALUE
"| 5)LIST ALL CUSTOMERS         6)ADD A PRODUCT               |".
    05  MENU-LINE-9                   PIC  X(62)  VALUE
"| 7)UPDATE CUSTOMER RECORD     8)DELETE A PRODUCT            |".
    05  MENU-LINE-10                  PIC  X(62)  VALUE
"| 9)REWIND/RESET CUSTOMER SET 10)OBTAIN DATA ITEM INFORMATION|".
    05  MENU-LINE-11                  PIC  X(62)  VALUE
"| 11)GENERATE ERROR MESSAGES  12)CLOSE DATABASE              |".
```

Main Body of Program

```
PROCEDURE DIVISION.
10-MAIN-LINE.
    PERFORM WITH TEST AFTER UNTIL OPTION = 12
        PERFORM 20-DISPLAY-MENU
        PERFORM 30-DO-ACTION
    END-PERFORM
    STOP RUN.
20-DISPLAY-MENU.
    DISPLAY CLEAR SCREEN
    DISPLAY MENU-LINE-1
    DISPLAY MENU-LINE-2
    DISPLAY MENU-LINE-3
    DISPLAY MENU-LINE-4
    DISPLAY MENU-LINE-5
    DISPLAY MENU-LINE-2
    DISPLAY MENU-LINE-6
    DISPLAY MENU-LINE-7
    DISPLAY MENU-LINE-8
    DISPLAY MENU-LINE-9
    DISPLAY MENU-LINE-10
    DISPLAY MENU-LINE-11
    DISPLAY MENU-LINE-2
    DISPLAY MENU-LINE-1
    DISPLAY SPACE.
30-DO-ACTION.
    DISPLAY "          Enter your option : "
        WITH NO ADVANCING
    ACCEPT OPTION FREE
    EVALUATE OPTION
        WHEN  1 PERFORM 100-OPEN-THE-DATABASE
        WHEN  2 PERFORM 200-GET-SALES-FOR-DATE
        WHEN  3 PERFORM 300-GET-A-CUSTOMER-RECORD
        WHEN  4 PERFORM 400-GET-A-PRODUCT-RECORD
        WHEN  5 PERFORM 500-LIST-ALL-CUSTOMERS
        WHEN  6 PERFORM 600-ADD-A-PRODUCT
        WHEN  7 PERFORM 700-UPDATE-A-CUSTOMER
        WHEN  8 PERFORM 800-DELETE-A-PRODUCT
        WHEN  9 PERFORM 900-REWIND-CUSTOMER-SET
        WHEN 10 PERFORM 1000-GET-DATA-ITEM-INFO
        WHEN 11 PERFORM 1100-GET-ERROR-AND-EXPLAIN
        WHEN 12 PERFORM 1200-CLOSE-THE-DATABASE
        WHEN OTHER
               DISPLAY "-----------------------------------"
               DISPLAY "| Please enter an option between  |"
               DISPLAY "|          1 and 12.              |"
```

```
                DISPLAY "-----------------------------------"
                DISPLAY "Press Enter to Continue... "
                        NO ADVANCING
                ACCEPT OPTION FREE
        END-EVALUATE.
```

Opening the Database

This paragraph implements the OPEN-THE-DATABASE routine of the sample program in COBOL II. All required values, such as the password, are provided by the routine. Note that the password DO-ALL establishes user class number 18. The password DO-ALL is followed by a semicolon because it is less than eight characters long; a blank can be substituted for the semicolon. OPEN-THE-DATABASE uses open mode 1, which is the shared modify access mode. Error trapping is done by referring all non-zero conditions to paragraph 1100-GET-ERROR-AND-EXPLAIN.

```
      ****************************************************************
      * ACCESS   :  Mode 1 - Shared Modify Access (SMA) with locking required
      *
      * Called By:  30-DO-ACTION
      *
      * Calls    :  DBOPEN in mode 1 (SMA)
      *             1100-GET-ERROR-AND-EXPLAIN
       100-OPEN-THE-DATABASE.
           MOVE SPACES TO BASEID
           MOVE "ORDERS" TO BASENAME
           MOVE ";" TO TERMINATOR
           MOVE "DO-ALL;" TO PASSWORD
           MOVE 1 TO DB-MODE
           CALL "DBOPEN" USING DBNAME, PASSWORD, DB-MODE, STATUS1
           IF CONDITION NOT = 0 THEN
               PERFORM 1100-GET-ERROR-AND-EXPLAIN
           END-IF.
```

Retrieving All the Records on a Chain (with Item Level Locking)

This paragraph implements the GET-SALES-FOR-DATE routine of the sample program. Chain access is achieved using a call to DBFIND to determine the location of the first and last entries in the chain. The search item used for this call is PURCH-DATE. An item level lock is obtained on the value of the search item before the DBFIND call. After that, individual chain items are retrieved, until the end of the chain is encountered. This is done using multiple calls to the DBGET procedure.

The routine traps two exceptional conditions:

1. Status condition 17 from the DBFIND call, indicating that the chain head cannot be located.
2. Status 15 from the DBGET call, indicating the end of the chain.

The status interpretation routine permits you to either abort or continue with the execution after viewing all error messages.

```
*****************************************************************
* ACCESS   :  Mode 1 - Shared Modify Access
*
* Called By:  30-DO-ACTION
*
* Calls    :  DBLOCK in mode 5 (unconditional item level locking)
*             DBFIND in mode 1 (chained access)
*             DBGET  in mode 5 (forward chained read)
*             DBUNLOCK in mode 1 (unlock)
*             1100-GET-ERROR-AND-EXPLAIN
 200-GET-SALES-FOR-DATE.
     MOVE 1 TO NUM-OF-ELEMENTS
     MOVE 21 TO LENGTH-OF-DESCRIPTOR
     MOVE "SALES;" TO DATA-SET-OF-DESCRIPTOR
     MOVE "PURCH-DATE;" TO DATA-ITEM-OF-DESCRIPTOR
     MOVE " =" TO RELOP-FOR-DATA-ITEM
     DISPLAY CLEAR SCREEN
     DISPLAY "   Enter The Date of Purchase as (YYMMDD) >>> "
             NO ADVANCING
     ACCEPT SEARCH-ITEM-VALUE FREE
     MOVE 5 TO DB-MODE
     MOVE SEARCH-ITEM-VALUE TO VALUE-FOR-DATA-ITEM
     CALL "DBLOCK" USING DBNAME, LOCK-DESCRIPTOR-ARRAY, DB-MODE,
                         STATUS1
     IF CONDITION NOT = 0 THEN
         PERFORM 1100-GET-ERROR-AND-EXPLAIN
     END-IF
     MOVE "SALES;" TO SALES-DETAIL
     MOVE 1 TO DB-MODE
     MOVE "PURCH-DATE;" TO SEARCH-ITEM-NAME
     CALL "DBFIND" USING DBNAME, SALES-DETAIL, DB-MODE, STATUS1,
                         SEARCH-ITEM-NAME, SEARCH-ITEM-VALUE
     IF CONDITION = 0 THEN
         SET FOUND TO TRUE
     ELSE
         SET NOT-FOUND TO TRUE
         IF CONDITION = NO-CHAIN-HEAD THEN
             DISPLAY CLEAR SCREEN
             DISPLAY "****************************************"
             DISPLAY "* No Such Entry in the Sales Data Set. *"
             DISPLAY "* Please Try Again.                     *"
             DISPLAY "****************************************"
             DISPLAY "Press Enter to Continue -------------->"
                     NO ADVANCING
             ACCEPT OPTION FREE
         ELSE
             PERFORM 1100-GET-ERROR-AND-EXPLAIN
         END-IF
     END-IF
     IF FOUND THEN
         DISPLAY CLEAR SCREEN
         DISPLAY SALES-BUFFER-HEADER
         DISPLAY LINE-HEADER
         PERFORM WITH TEST BEFORE UNTIL CONDITION = END-OF-CHAIN
             MOVE 5 TO DB-MODE
             MOVE "@;" TO LIST
             CALL "DBGET" USING DBNAME, SALES-DETAIL, DB-MODE,
                                STATUS1, LIST, SALES-BUFFER,
```

```
                        NOT-USED-PARM-16
            IF CONDITION NOT = 0 THEN
                IF CONDITION = END-OF-CHAIN THEN
                   DISPLAY SPACE
                   DISPLAY SPACE
                   DISPLAY SPACE
                   DISPLAY "-----> End of Chain, " NO ADVANCING
                   DISPLAY "Hit Enter to Continue" NO ADVANCING
                   ACCEPT OPTION FREE
                ELSE
                    PERFORM 1100-GET-ERROR-AND-EXPLAIN
                END-IF
            END-IF
            MOVE ACCOUNT-NUMBER OF SALES-BUFFER
                     TO ACCOUNT-NUMBER-OUT
            MOVE STOCK-NUMBER OF SALES-BUFFER
                     TO STOCK-NUMBER-OUT
            MOVE QUANTITY OF SALES-BUFFER TO QUANTITY-OUT
            MOVE PRICE OF SALES-BUFFER TO PRICE-OUT
            MOVE TAX OF SALES-BUFFER TO TAX-OUT
            MOVE TOTAL OF SALES-BUFFER TO TOTAL-OUT
            MOVE PURCH-DATE OF SALES-BUFFER TO PURCH-DATE-OUT
            MOVE DELIV-DATE OF SALES-BUFFER TO DELIV-DATE-OUT
            DISPLAY SALES-BUFFER-OUT
        END-PERFORM
    END-IF
    MOVE 1 TO DB-MODE
    CALL "DBUNLOCK" USING DBNAME, SALES-DETAIL, DB-MODE, STATUS1
    IF CONDITION NOT = 0 THEN
        PERFORM 1100-GET-ERROR-AND-EXPLAIN
    END-IF.
```

Retrieving a Data Entry Using a Record Number

This paragraph implements the GET-A-CUSTOMER-RECORD routine of the sample program. The record number for the directed read is obtained by calling DBGET in mode 7. The saved record number is then used as the argument value for a call to DBGET in mode 4. Status 17 indicates two different conditions for DBGET in modes 4 and 7, as follows:

1. For mode 7, this value means that no entry exists with the specified search value.
2. For mode 4, this value means that the entry at the specified record number is empty.

Note that for increased performance, the calculated access call is made with a list parameter equal to zero.

```
*****************************************************************
* ACCESS   :  Mode 1 - Shared Modify Access
*
* Called By:  30-DO-ACTION
*
* Calls    :  DBGET in mode 7 (calculated read)
*             DBGET in mode 4 (directed read)
*             1100-GET-ERROR-AND-EXPLAIN
 300-GET-A-CUSTOMER-RECORD.
     SET NOT-FOUND TO TRUE
     DISPLAY CLEAR SCREEN
     DISPLAY "Enter the Account # For The Customer Master"
```

```
        NO ADVANCING
DISPLAY "------------> " NO ADVANCING
ACCEPT KEY-ITEM-VALUE FREE
MOVE 7 TO DB-MODE
MOVE ZERO TO LIST-NO-ITEM
MOVE "@;" TO LIST
MOVE "CUSTOMER;" TO CUSTOMER-MASTER
CALL "DBGET" USING DBNAME, CUSTOMER-MASTER, DB-MODE, STATUS1,
                   LIST-NO-ITEM, CUSTOMER-BUFFER,
                   KEY-ITEM-VALUE
IF CONDITION = 0 THEN
    SET FOUND TO TRUE
    MOVE RECORD-NUMBER TO SAVED-RECORD-NUMBER
ELSE
    IF CONDITION = NO-SUCH-ENTRY THEN
        DISPLAY CLEAR SCREEN
        DISPLAY "****************************************"
        DISPLAY "* No Such Entry in the Customer Data Set."
        DISPLAY "* Please Try Again.                     "
        DISPLAY "****************************************"
        DISPLAY "Press Enter to Continue ------------>"
                NO ADVANCING
        ACCEPT OPTION FREE
    ELSE
        PERFORM 1100-GET-ERROR-AND-EXPLAIN
    END-IF
END-IF
IF FOUND THEN
    MOVE 4 TO DB-MODE
    MOVE "@;" TO LIST
    MOVE "CUSTOMER;" TO CUSTOMER-MASTER
    CALL "DBGET" USING DBNAME, CUSTOMER-MASTER, DB-MODE,
                       STATUS1, LIST, CUSTOMER-BUFFER,
                       SAVED-RECORD-NUMBER
    IF CONDITION NOT = 0 THEN
        IF CONDITION = ENTRY-HAS-NO-DATA THEN
            DISPLAY CLEAR SCREEN
            DISPLAY "****************************************"
            DISPLAY "* Entry At The Specified Record Number "
            DISPLAY "* Has Been Deleted.                     "
            DISPLAY "****************************************"
            DISPLAY "Press Enter To Continue ------------>"
                    NO ADVANCING
            ACCEPT OPTION FREE
        ELSE
            PERFORM 1100-GET-ERROR-AND-EXPLAIN
        END-IF
    ELSE
        DISPLAY SPACE
        DISPLAY SPACE
        DISPLAY SPACE
        DISPLAY SPACE
        DISPLAY SPACE
        DISPLAY SPACE
        MOVE KEY-ITEM-VALUE TO ACCOUNT-NUMBER-CUST-OUT
        DISPLAY "Data On Account# = ",
                ACCOUNT-NUMBER-CUST-OUT
        DISPLAY "****************************************"
```

```
            DISPLAY "*                                        *"
            MOVE ACCOUNT-NUMBER OF CUSTOMER-BUFFER TO
                 ACCOUNT-NUMBER-CUST-OUT
            DISPLAY "* Account #  = ", ACCOUNT-NUMBER-CUST-OUT
            DISPLAY "* Last Name  = ", LAST-NAME
                                       OF CUSTOMER-BUFFER
            DISPLAY "* First Name = ", FIRST-NAME
                                       OF CUSTOMER-BUFFER
            DISPLAY "* Initial    = ", INITIAL1
                                       OF CUSTOMER-BUFFER
            DISPLAY "* Address    = ", STREET-ADDRESS
                                       OF CUSTOMER-BUFFER
            DISPLAY "* City       = ", CITY OF CUSTOMER-BUFFER
            DISPLAY "* State      = ", STATE OF CUSTOMER-BUFFER
            DISPLAY "* Zip        = ", ZIP OF CUSTOMER-BUFFER
            DISPLAY "*                                        *"
            DISPLAY "****************************************"
            DISPLAY SPACE
            DISPLAY SPACE
            DISPLAY "Press Enter to Continue ----------------->"
                    NO ADVANCING
            ACCEPT OPTION FREE
        END-IF
    END-IF.
```

Retrieving Master Data Using a Key Value

This paragraph implements the GET-PRODUCT-RECORD routine of the sample program. The calculated access is achieved by a call to DBGET in mode 7. The exceptional condition in this routine is indicated by status 17 for search values which do not have any corresponding entries. Error trapping calls 1100-GET-ERROR-AND-EXPLAIN upon detection of a non-exceptional condition.

```
*****************************************************************
* ACCESS   :  Mode 1 - Shared Modify Access
*
* Called By:  30-DO-ACTION
*
* Calls    :  DBGET in mode 7 (calculated read)
*             1100-GET-ERROR-AND-EXPLAIN
 400-GET-PRODUCT-RECORD.
     SET NOT-FOUND TO TRUE
     DISPLAY CLEAR SCREEN
     DISPLAY "Enter the Stock # in the Product Master ----->"
         NO ADVANCING
     ACCEPT KEY-ITEM-VALUE-PRODUCT FREE
     MOVE 7 TO DB-MODE
     MOVE "@;" TO LIST
     MOVE "PRODUCT;" TO PRODUCT-MASTER
     CALL "DBGET" USING DBNAME, PRODUCT-MASTER, DB-MODE, STATUS1,
                        LIST, PRODUCT-BUFFER,
                        KEY-ITEM-VALUE-PRODUCT
     IF CONDITION = 0 THEN
         SET FOUND TO TRUE
     ELSE
         SET NOT-FOUND TO TRUE
         IF CONDITION = NO-CHAIN-HEAD THEN
```

```
                DISPLAY CLEAR SCREEN
                DISPLAY "*************************************"
                DISPLAY "* No Such Entry in the Product Data Set."
                DISPLAY "* Please Try Again.                    "
                DISPLAY "*************************************"
                DISPLAY "Press Enter To Continue ------------>"
                         NO ADVANCING
                ACCEPT OPTION FREE
            ELSE
                PERFORM 1100-GET-ERROR-AND-EXPLAIN
            END-IF
        END-IF
        IF FOUND THEN
            DISPLAY SPACE
            DISPLAY "Data On Stock # = ", KEY-ITEM-VALUE-PRODUCT
            DISPLAY "*********************************************"
            DISPLAY "*                                           *"
            DISPLAY "* Stock # = ", STOCK-NUMBER OF PRODUCT-BUFFER
            DISPLAY "* Product = ", DESCRIPTION OF PRODUCT-BUFFER
            DISPLAY "*                                           *"
            DISPLAY "*********************************************"
            DISPLAY SPACE
            DISPLAY SPACE
            DISPLAY "Press Enter To Continue ------------>"
                    NO ADVANCING
            ACCEPT OPTION FREE
        END-IF.
```

Retrieving Data Serially (with Set Level Locking)

This paragraph implements the LIST-ALL-CUSTOMERS routine of the sample program. Serial read of the Customer data set is achieved using multiple calls to the DBGET procedure in mode 2. Using the list parameter the routine requests only the ACCOUNT, FIRST-NAME, LAST-NAME, and INITIAL data items.

This procedure locks the Customer data set exclusively using a call to the DBLOCK procedure in mode 3. The subsequent DBUNLOCK releases this lock. This is done when the exceptional condition, end of data set, is encountered. Locking in a shared modify access environment guarantees that no other user is modifying the data that you are reading. Error trapping calls 1100-GET-ERROR-AND-EXPLAIN for any non-exceptional condition codes.

```
*****************************************************************
* ACCESS   :  Mode 1 - Shared Modify Access
*
* Called By:  30-DO-ACTION
*
* Calls    :  DBLOCK in mode 3 (unconditional)
*             DBGET in mode 2 (forward read)
*             DBUNLOCK in mode 1 (unlock)
*             1100-GET-ERROR-AND-EXPLAIN
 500-LIST-ALL-CUSTOMERS.
    DISPLAY CLEAR SCREEN
    DISPLAY SPACE
    DISPLAY "Acct-Number        N A M E"
    DISPLAY "-----------------------------------------"
```

```
SET NOT-DONE TO TRUE
MOVE 3 TO DB-MODE
MOVE "@;" TO LIST
MOVE "CUSTOMER;" TO CUSTOMER-MASTER
CALL "DBLOCK" USING DBNAME, CUSTOMER-MASTER, DB-MODE, STATUS1
IF CONDITION NOT = 0 THEN
    PERFORM 1100-GET-ERROR-AND-EXPLAIN
END-IF
MOVE 2 TO DB-MODE
MOVE "ACCOUNT,LAST-NAME,FIRST-NAME,INITIAL;" TO LIST
PERFORM WITH TEST AFTER UNTIL DONE
    CALL "DBGET" USING DBNAME, CUSTOMER-MASTER, DB-MODE,
                       STATUS1, LIST, CUSTOMER-BUFFER,
                       NOT-USED-PARM-32
    IF CONDITION NOT = 0 THEN
        IF CONDITION = END-OF-DATA-SET THEN
            SET DONE TO TRUE
            DISPLAY SPACE
            DISPLAY "*End of Data Set"
            DISPLAY "* Press Enter to Continue ------->"
                    NO ADVANCING
            ACCEPT OPTION FREE
        ELSE
            PERFORM 1100-GET-ERROR-AND-EXPLAIN
        END-IF
    ELSE
        MOVE ACCOUNT-NUMBER OF CUSTOMER-BUFFER TO
             ACCOUNT-NUMBER-CUST-OUT
        MOVE FIRST-NAME OF CUSTOMER-BUFFER TO
             FIRST-NAME-CUST-OUT
        MOVE INITIAL1 OF CUSTOMER-BUFFER TO
             INITIAL1-CUST-OUT
        MOVE LAST-NAME OF CUSTOMER-BUFFER TO
             LAST-NAME-CUST-OUT
        DISPLAY CUSTOMER-BUFFER-OUT
    END-IF
END-PERFORM
MOVE 1 TO DB-MODE
CALL "DBUNLOCK" USING DBNAME, CUSTOMER-MASTER, DB-MODE,
                      STATUS1
IF CONDITION NOT = 0 THEN
    PERFORM 1100-GET-ERROR-AND-EXPLAIN
END-IF.
```

Adding an Entry

This paragraph implements the ADD-A-PRODUCT routine of the sample program to add a data entry to the Product manual master data set. The add entry operation is achieved using a call to DBPUT. Before this call, the paragraph initiates a TurboIMAGE/XL transaction and locks the product master data set. The beginning and end of the transaction are indicated by calls to DBBEGIN and DBEND. Locking is done exclusively at the set level. For error trapping, 1100-GET-ERROR-AND-EXPLAIN is called when any status code is not equal to zero.

Note that the list contains an at-sign (@) which requests TurboIMAGE/XL to return all fields of the data set in the order defined in the schema. Other valid lists are the null list

('0;') which returns no data, and same list ('*;') which returns the same fields listed in the previous call.

```
*****************************************************************
* ACCESS   :  Mode 1 - Shared Modify Access
*
* Called By:  30-DO-ACTION
*
* Calls    :  DBLOCK in mode 3 (unconditional)
*             DBBEGIN in mode 1 (transaction begin)
*             DBPUT in mode 1 (put)
*             DBEND in mode 1 (transaction end)
*             DBUNLOCK in mode 1 (unlock)
*             1100-GET-ERROR-AND-EXPLAIN
 600-ADD-A-PRODUCT.
     MOVE 0 TO ANSWER
     PERFORM WITH TEST BEFORE UNTIL ANSWER = 1
         DISPLAY CLEAR SCREEN
         DISPLAY "   Please Provide the Following Values  "
         DISPLAY "*************************************"
         DISPLAY "*                                   *"
         DISPLAY "* Stock # = "  NO ADVANCING
         ACCEPT STOCK-NUMBER OF PRODUCT-BUFFER FREE
         DISPLAY "* Product = "  NO ADVANCING
         ACCEPT DESCRIPTION OF PRODUCT-BUFFER FREE
         DISPLAY "*                                   *"
         DISPLAY "*************************************"
         DISPLAY SPACE
         DISPLAY SPACE
         DISPLAY "Enter, <1> to Continue,  <2> to Retry >"
                 NO ADVANCING
         ACCEPT ANSWER FREE
     END-PERFORM
     MOVE "PRODUCT;" TO PRODUCT-MASTER
     MOVE 3 TO DB-MODE
     CALL "DBLOCK" USING DBNAME, PRODUCT-MASTER, DB-MODE, STATUS1
     IF CONDITION NOT = 0 THEN
         PERFORM 1100-GET-ERROR-AND-EXPLAIN
     END-IF
     MOVE "Add Entry to Product Set Begin" TO TEXT1
     MOVE 16 TO TEXTLEN
     MOVE 1 TO DB-MODE
     CALL "DBBEGIN" USING DBNAME, TEXT1, DB-MODE, STATUS1, TEXTLEN
     IF CONDITION NOT = 0 THEN
         PERFORM 1100-GET-ERROR-AND-EXPLAIN
     END-IF
     MOVE "@;" TO LIST
     CALL "DBPUT" USING DBNAME, PRODUCT-MASTER, DB-MODE, STATUS1,
                        LIST, PRODUCT-BUFFER
     IF CONDITION = 0 THEN
         DISPLAY SPACE
         DISPLAY SPACE
         DISPLAY SPACE
         DISPLAY "***********************************"
         DISPLAY "Stock ", STOCK-NUMBER OF PRODUCT-BUFFER,
                 "Was Successfully Added to the Product Set"
         DISPLAY "***********************************"
         DISPLAY "Enter to Continue ..............<>"
```

```
                NO ADVANCING
            ACCEPT OPTION FREE
        ELSE
            PERFORM 1100-GET-ERROR-AND-EXPLAIN
        END-IF
        MOVE "Add Entry to Product Set End" TO TEXT1
        MOVE 14 TO TEXTLEN
        CALL "DBEND" USING DBNAME, TEXT1, DB-MODE, STATUS1, TEXTLEN
        IF CONDITION NOT = 0 THEN
            PERFORM 1100-GET-ERROR-AND-EXPLAIN
        END-IF
        CALL "DBUNLOCK" USING DBNAME, PRODUCT-MASTER, DB-MODE, STATUS1
        IF CONDITION NOT = 0 THEN
            PERFORM 1100-GET-ERROR-AND-EXPLAIN
        END-IF.
```

Updating an Entry

This paragraph implements the UPDATE-A-CUSTOMER routine of the sample program. The update process takes place in two phases.

In the first phase, the requested entry is located and retrieved. This is achieved by a call to DBGET in mode 7. Then, the user provides the new values.

In the second phase, the recently modified values replace the existing entry. This is implemented using a call to DBUPDATE. Before this call, the paragraph starts a TurboIMAGE/XL transaction bracketed by calls for locking the volatile item. To retrieve the entry, DBGET is called in mode 1. This call retrieves the entry located in the previous stage.

The paragraph must confirm that values retrieved in the first stage are still residing in the same entry. This should be done before the actual update. If the contents of the buffers are the same, the paragraph can continue with the operation. Otherwise, it should end the transaction and release the locks.

The exceptional condition for this paragraph is status 17. This indicates that the requested entry does not exist or is empty.

```
*****************************************************************
* ACCESS   :  Mode 1 - Shared Modify Access
*
* Called By:  30-DO-ACTION
*
* Calls    :  DBGET in mode 7 (calculated)
*             DBLOCK in mode 5 (unconditional)
*             DBBEGIN in mode 1 (transaction begin)
*             DBGET in mode 1 (re-read)
*             DBUPDATE in mode 1 (update)
*             DBEND in mode 1 (transaction end)
*             DBUNLOCK in mode 1 (unlock)
*             1100-GET-ERROR-AND-EXPLAIN
 700-UPDATE-A-CUSTOMER.
     SET NOT-FOUND TO TRUE
     DISPLAY CLEAR SCREEN
     DISPLAY "Enter the Account # for The Customer Master"
             NO ADVANCING
     DISPLAY "---------> " NO ADVANCING
```

```
ACCEPT KEY-ITEM-VALUE FREE
MOVE 7 TO DB-MODE
MOVE 0 TO LIST-NO-ITEM
MOVE "@;" TO LIST
MOVE "CUSTOMER;" TO CUSTOMER-MASTER
CALL "DBGET" USING DBNAME, CUSTOMER-MASTER, DB-MODE, STATUS1,
                   LIST, CUSTOMER-BUFFER, KEY-ITEM-VALUE
IF CONDITION = 0 THEN
    SET FOUND TO TRUE
    DISPLAY SPACE
    DISPLAY SPACE
    DISPLAY SPACE
    DISPLAY "Data On Account # = ", KEY-ITEM-VALUE
    DISPLAY "****************************************"
    DISPLAY "*                                      *"
    DISPLAY "* Account #  = ", ACCOUNT-NUMBER
                               OF CUSTOMER-BUFFER
    DISPLAY "* Last Name  = ", LAST-NAME
                               OF CUSTOMER-BUFFER
    DISPLAY "* First Name = ", FIRST-NAME
                               OF CUSTOMER-BUFFER
    DISPLAY "* Initial    = ", INITIAL1
                               OF CUSTOMER-BUFFER
    DISPLAY "* Address    = ", STREET-ADDRESS
                               OF CUSTOMER-BUFFER
    DISPLAY "* City       = ", CITY OF CUSTOMER-BUFFER
    DISPLAY "* State      = ", STATE OF CUSTOMER-BUFFER
    DISPLAY "* Zip        = ", ZIP OF CUSTOMER-BUFFER
    DISPLAY "*                                      *"
    DISPLAY "****************************************"
    DISPLAY SPACE
    DISPLAY SPACE
    DISPLAY "Press Enter to Continue ----------------->"
            NO ADVANCING
    ACCEPT OPTION FREE
ELSE
    IF CONDITION = NO-SUCH-ENTRY THEN
        DISPLAY CLEAR SCREEN
        DISPLAY "****************************************"
        DISPLAY "* No Such Entry in the Customer Data Set."
        DISPLAY "* Please Try Again.                     "
        DISPLAY "****************************************"
        DISPLAY "Enter to Continue --------->" NO ADVANCING
        ACCEPT ANSWER FREE
    ELSE
        PERFORM 1100-GET-ERROR-AND-EXPLAIN
    END-IF
END-IF
IF FOUND THEN
    MOVE CORRESPONDING CUSTOMER-BUFFER TO CUSTOMER-BUFFER-OLD
    MOVE 0 TO ANSWER
    PERFORM WITH TEST BEFORE UNTIL ANSWER = 1
        DISPLAY SPACE
        DISPLAY SPACE
        DISPLAY SPACE
        DISPLAY "Provide New Values For the Following"
        DISPLAY "****************************************"
        DISPLAY "*                                      *"
```

```
    DISPLAY "* Account #   = " NO ADVANCING
    ACCEPT ACCOUNT-NUMBER OF CUSTOMER-BUFFER-NEW FREE
    DISPLAY "* Last Name   = " NO ADVANCING
    ACCEPT LAST-NAME OF CUSTOMER-BUFFER-NEW FREE
    DISPLAY "* First Name  = " NO ADVANCING
    ACCEPT FIRST-NAME OF CUSTOMER-BUFFER-NEW FREE
    DISPLAY "* Initial     = " NO ADVANCING
    ACCEPT INITIAL1 OF CUSTOMER-BUFFER-NEW FREE
    DISPLAY "* Address     = " NO ADVANCING
    ACCEPT STREET-ADDRESS OF CUSTOMER-BUFFER-NEW FREE
    DISPLAY "* City        = " NO ADVANCING
    ACCEPT CITY OF CUSTOMER-BUFFER-NEW FREE
    DISPLAY "* State       = " NO ADVANCING
    ACCEPT STATE OF CUSTOMER-BUFFER-NEW FREE
    DISPLAY "* Zip         = " NO ADVANCING
    ACCEPT ZIP OF CUSTOMER-BUFFER-NEW FREE
    DISPLAY "*                                      *"
    DISPLAY "****************************************"
    DISPLAY SPACE
    DISPLAY SPACE
    DISPLAY "Enter <1> to Continue,  <2> to Retry >"
            NO ADVANCING
    ACCEPT ANSWER FREE
    IF ANSWER = 2 THEN
        DISPLAY CLEAR SCREEN
    END-IF
END-PERFORM
MOVE 1 TO NUM-OF-ELEMENTS
MOVE 22 TO LENGTH-OF-DESCRIPTOR
MOVE "CUSTOMER;" TO DATA-SET-OF-DESCRIPTOR
MOVE "ACCOUNT;" TO DATA-ITEM-OF-DESCRIPTOR
MOVE " =" TO RELOP-FOR-DATA-ITEM
MOVE KEY-ITEM-VALUE TO NUM-VALUE-FOR-DATA-ITEM
MOVE 5 TO DB-MODE
CALL "DBLOCK" USING DBNAME, LOCK-DESCRIPTOR-ARRAY, DB-MODE,
                    STATUS1
IF CONDITION NOT = 0 THEN
    PERFORM 1100-GET-ERROR-AND-EXPLAIN
END-IF
MOVE "Update Entry In Customer Set Begin" TO TEXT1
MOVE 17 TO TEXTLEN
MOVE 1 TO DB-MODE
CALL "DBBEGIN" USING DBNAME, TEXT1, DB-MODE, STATUS1,
                     TEXTLEN
IF CONDITION NOT = 0 THEN
    PERFORM 1100-GET-ERROR-AND-EXPLAIN
END-IF
CALL "DBGET" USING DBNAME, CUSTOMER-MASTER, DB-MODE,
                   STATUS1, LIST, CUSTOMER-BUFFER,
                   NOT-USED-PARM-32
IF CONDITION NOT = 0 THEN
    PERFORM 1100-GET-ERROR-AND-EXPLAIN
ELSE
    IF CUSTOMER-BUFFER = CUSTOMER-BUFFER-OLD THEN
        CALL "DBUPDATE" USING DBNAME, CUSTOMER-MASTER,
                              DB-MODE, STATUS1, LIST,
                              CUSTOMER-BUFFER-NEW
        IF CONDITION NOT = 0 THEN
```

```
                    PERFORM 1100-GET-ERROR-AND-EXPLAIN
                END-IF
            ELSE
                DISPLAY CLEAR SCREEN
                DISPLAY SPACE
                DISPLAY SPACE
                DISPLAY SPACE
                DISPLAY SPACE
                DISPLAY "*************************************"
                DISPLAY "**    During Terminal Interaction     *"
                DISPLAY "** Data On Account Number ",
                        KEY-ITEM-VALUE
                DISPLAY "**         Has Been Modified."
                DISPLAY "**                                     *"
                DISPLAY "**       Please Try Again.             *"
                DISPLAY "Press Enter to Continue  ---------->"
                        NO ADVANCING
                ACCEPT OPTION FREE
            END-IF
        END-IF
        MOVE "Update Entry On Customer Set End" TO TEXT1
        MOVE 16 TO TEXTLEN
        CALL "DBEND" USING DBNAME, TEXT1, DB-MODE, STATUS1, TEXTLEN
        IF CONDITION NOT = 0 THEN
            PERFORM 1100-GET-ERROR-AND-EXPLAIN
        END-IF
        CALL "DBUNLOCK" USING DBNAME, CUSTOMER-MASTER, DB-MODE,
                              STATUS1
        IF CONDITION NOT = 0 THEN
            PERFORM 1100-GET-ERROR-AND-EXPLAIN
        END-IF
    END-IF.
```

Deleting an Entry

This paragraph implements the DELETE-A-PRODUCT routine of the sample program. The delete operation is achieved by a call to DBDELETE. This call is preceded by a call to DBGET in mode 7, which locates the entry for the delete operation. These calls are bracketed by calls to DBBEGIN and DBEND, which designate the beginning and the end of a TurboIMAGE/XL transaction.

Using calls to DBLOCK and DBUNLOCK in mode 3, the required resources are locked before the start of the transaction and released after its end.

Exceptional condition code 17 is trapped after the DBGET call. This indicates that the requested entry does not exist in the Product data set.

```
*****************************************************************
* ACCESS   :  Mode 1 - Shared Modify Access
*
* Called By:  30-DO-ACTION
*
* Calls    :  DBLOCK in mode 3 (unconditional)
*             DBBEGIN in mode 1 (transaction begin)
*             DBGET in mode 7 (calculated read)
*             DBDELETE in mode 1 (delete)
*             DBEND in mode 1 (transaction end)
```

```
*             DBUNLOCK in mode 1 (unlock)
*             1100-GET-ERROR-AND-EXPLAIN
 800-DELETE-A-PRODUCT.
     DISPLAY CLEAR SCREEN
     DISPLAY "Enter the stock # in the Product Master ----> "
             NO ADVANCING
     ACCEPT KEY-ITEM-VALUE-PRODUCT FREE
     MOVE 3 TO DB-MODE
     MOVE "@;" TO LIST
     MOVE "PRODUCT;" TO PRODUCT-MASTER
     CALL "DBLOCK" USING DBNAME, PRODUCT-MASTER, DB-MODE, STATUS1
     IF CONDITION NOT = 0 THEN
         PERFORM 1100-GET-ERROR-AND-EXPLAIN
     END-IF
     MOVE 1 TO DB-MODE
     MOVE "Delete Entry From The Product Set Begin " TO TEXT1
     MOVE 18 TO TEXTLEN
     CALL "DBBEGIN" USING DBNAME, PRODUCT-MASTER, DB-MODE, STATUS1,
                          TEXTLEN
     IF CONDITION NOT = 0 THEN
         PERFORM 1100-GET-ERROR-AND-EXPLAIN
     END-IF
     MOVE 7 TO DB-MODE
     CALL "DBGET" USING DBNAME, PRODUCT-MASTER, DB-MODE, STATUS1,
                        LIST, PRODUCT-BUFFER,
                        KEY-ITEM-VALUE-PRODUCT
     IF CONDITION NOT = 0 THEN
         IF CONDITION = NO-CHAIN-HEAD THEN
             DISPLAY CLEAR SCREEN
             DISPLAY "****************************************"
             DISPLAY "* No Such Entry in the Product Data Set. *"
             DISPLAY "* Please Try Again.                      *"
             DISPLAY "****************************************"
         ELSE
             PERFORM 1100-GET-ERROR-AND-EXPLAIN
         END-IF
     ELSE
         MOVE 1 TO DB-MODE
         CALL "DBDELETE" USING DBNAME, PRODUCT-MASTER, DB-MODE,
                               STATUS1
         IF CONDITION NOT = 0 THEN
             PERFORM 1100-GET-ERROR-AND-EXPLAIN
         ELSE
             DISPLAY SPACE
             DISPLAY SPACE
             DISPLAY SPACE
             DISPLAY SPACE
             DISPLAY "****************************************"
             DISPLAY "Product Record ", KEY-ITEM-VALUE-PRODUCT
                     NO ADVANCING
             DISPLAY "Was Successfully Deleted."
             DISPLAY "****************************************"
         END-IF
     END-IF
     MOVE 1 TO DB-MODE
     MOVE "Delete Entry From the Product Set End" TO TEXT1
     MOVE 18 TO TEXTLEN
     CALL "DBEND" USING DBNAME, PRODUCT-MASTER, DB-MODE, STATUS1,
```

```
                        TEXTLEN
    IF CONDITION NOT = 0 THEN
        PERFORM 1100-GET-ERROR-AND-EXPLAIN
    END-IF
    MOVE 1 TO DB-MODE
    CALL "DBUNLOCK" USING DBNAME, PRODUCT-MASTER, DB-MODE, STATUS1
    IF CONDITION NOT = 0 THEN
        PERFORM 1100-GET-ERROR-AND-EXPLAIN
    END-IF
    DISPLAY "Press Enter to Continue ----------> " NO ADVANCING
    ACCEPT OPTION FREE.
```

Rewinding a Data Set

This paragraph implements the REWIND-CUSTOMER-SET routine of the sample program. Resetting the data set pointer is achieved by a call to DBCLOSE in mode 2. No special condition is trapped.

```
*****************************************************************
* ACCESS   :  Mode 1 - Shared Modify Access
*
* Called By:  30-DO-ACTION
*
* Calls    :  DBCLOSE in mode 2 (rewind)
*             1100-GET-ERROR-AND-EXPLAIN
 900-REWIND-CUSTOMER-SET.
    MOVE "CUSTOMER;" TO CUSTOMER-MASTER
    MOVE 2 TO DB-MODE
    CALL "DBCLOSE" USING DBNAME, CUSTOMER-MASTER, DB-MODE, STATUS1
    IF CONDITION NOT = 0 THEN
        PERFORM 1100-GET-ERROR-AND-EXPLAIN
    END-IF.
```

Obtaining Database Information

This paragraph implements the GET-DATA-ITEM-INFO routine of the sample program. This information is obtained using a call to DBINFO in mode 102. The data item name passed through the DBINFO buffer identifies the data item under inquiry.

```
*****************************************************************
* ACCESS   :  Mode 1 - Shared Modify Access
*
* Called By:  30-DO-ACTION
*
* Calls    :  DBINFO in mode 102 (item access)
*             1100-GET-ERROR-AND-EXPLAIN
 1000-GET-DATA-ITEM-INFO.
    DISPLAY CLEAR SCREEN
    DISPLAY "Enter your data item name------> " NO ADVANCING
    ACCEPT DATA-ITEM-NAME-IN FREE
    MOVE 102 TO DB-MODE
    CALL "DBINFO" USING DBNAME, DATA-ITEM-NAME-IN, DB-MODE, STATUS1,
                        M-102-BUFFER
    IF CONDITION NOT = 0 THEN
        PERFORM 1100-GET-ERROR-AND-EXPLAIN
    END-IF
    DISPLAY SPACE
```

```
    DISPLAY SPACE
    DISPLAY SPACE
    DISPLAY DATA-ITEM-NAME-IN, " Data Item"
    DISPLAY "---------------------------------------"
    DISPLAY "Data Item Name = ", DATA-ITEM-NAME
    DISPLAY "Data Item Type = ", DATA-ITEM-TYPE
    DISPLAY "Data Item Length = ", DATA-ITEM-LENGTH
    DISPLAY "Data Item Count = ", DATA-ITEM-COUNT
    DISPLAY "---------------------------------------"
    DISPLAY "Press Enter to Continue... " NO ADVANCING
    ACCEPT OPTION FREE.
```

Obtaining Error Messages and Explanations

The following paragraph implements the GET-ERROR-AND-EXPLAIN routine of the sample program. This paragraph calls DBEXPLAIN and DBERROR. DBEXPLAIN interprets the contents of the Status parameter and prints a message on $STDLIST. DBERROR returns a message in the ERROR-BUFFER, explaining the condition code returned by TurboIMAGE/XL. At the end the paragraph, users can choose to abort or continue the execution of this program.

```
*****************************************************************
* Access   :     Mode 1 - Shared Modified Access
*
* Called by:     100-OPEN-THE-DATABASE
*                200-GET-SALES-FOR-DATE
*                300-GET-A-CUSTOMER-RECORD
*                400-GET-PRODUCT-RECORD
*                500-LIST-ALL-CUSTOMERS
*                600-ADD-A-PROUDCT
*                700-UPDATE-A-CUSTOMER
*                800-DELETE-A-PRODUCT
*                900-REWIND-CUSTOMER-SET
*                1000-GET-DATA-ITEM-INFO
*                1200-CLOSE-THE-DATABASE
*
* Calls    :     DBERROR
*                DBEXPLAIN
 1100-GET-ERROR-AND-EXPLAIN.
     MOVE SPACES TO ERROR-BUFFER
     CALL "DBERROR" USING STATUS1, ERROR-BUFFER, ERROR-LENGTH
     DISPLAY "-------------------------------------------------"
     DISPLAY ERROR-BUFFER
     DISPLAY "-------------------------------------------------"
     DISPLAY SPACE
     CALL "DBEXPLAIN" USING STATUS1
     MOVE ZERO TO ANSWER
     DISPLAY "---Enter, >1< to Abort...,  <2> to Continue > "
             NO ADVANCING
     ACCEPT ANSWER FREE

     IF ANSWER NOT = 1 THEN
         DISPLAY "Continuing....."
     ELSE
         STOP RUN
     END-IF.
```

Closing the Database

This paragraph implements the CLOSE-THE-DATABASE routine of the sample program. Closing the database is achieved by a call to DBCLOSE in mode 1. Error handling is done by referring all non-zero returned conditions to the 1100-GET-ERROR-AND-EXPLAIN paragraph.

```
*****************************************************************
* ACCESS   :  Mode 1 - Shared Modify Access
*
* Called By:  30-DO-ACTION
*
* Calls    :  DBCLOSE in mode 1 (close)
*             1100-GET-ERROR-AND-EXPLAIN
 1200-CLOSE-THE-DATABASE.
     MOVE 1 TO DB-MODE
     CALL "DBCLOSE" USING DBNAME, PASSWORD, DB-MODE, STATUS1
     IF CONDITION NOT = 0 THEN
         PERFORM 1100-GET-ERROR-AND-EXPLAIN
     END-IF.
```

FORTRAN 77

Portions of the model program presented at the beginning of this chapter are now shown here in FORTRAN 77. The examples perform specific tasks to illustrate the use of TurboIMAGE/XL intrinsics.

Data items are defined at the beginning of the sample program. Explicit declaration of intrinsics is not required. Other global variables in this program are placed in a COMMON file.

NOTE Because the Schema Processor, DBSCHEMA, upshifts alphabetic characters, programs must specify data set and data item names in all uppercase characters. Take note of this if FORTRAN 77 does not require that you use uppercase characters.

For information on TurboIMAGE/XL data item lengths and type designators, refer to chapter 3. Tables 3-2 and 3-3 show the TurboIMAGE/XL type designators, sub-item lengths, and data types typically used to process them in Pascal.

NOTE All parameters must be on halfword boundaries.

Because FORTRAN 77 requires that the parameters be on halfword boundaries, they must be integer arrays equivalent to character strings if necessary.

Defining Data Types, Variables, and Intrinsics

The following declarations are placed in a FORTRAN 77 COMMON file. This file enables different subroutines to import all necessary declarations. In this program, the COMMON file is called comon1 and is included with the directive `$Include 'comon1'`.

```
C**** TurboIMAGE/XL's Global Declaration

C**** Set up for the Database name parameter.

      Integer*2 DBname(10)
      Character BaseName*16
      Equivalence(DBname(1),BaseName)
      Common /Database_Name_Type / DBname

C**** Set up for the Password parameter.
      Character Pass_Word*10
      Integer*2 Password(5)
      Equivalence (Password(1),Pass_Word)
      Common /Database_password_type/ password

C**** Set up for the Mode parameter.

      Integer In,Out,Not_Used_Parm
      Integer*2 Mode
      Integer*2 Mode1_SMA, Mode5_Unconditional, Mode1_Chained_Read
```

```
      Integer*2 Mode5_Forward, Mode1_Unlock

C**** Set up for the Status parameter.

      Integer*2 Status(10)
      Integer*2 Condition
      Integer*2 Length
      Integer*4 Record_Number
      Integer*4 Chain_Count
      Integer*4 Back_Pointer
      Integer*4 Forward_Pointer
      Equivalence(Status(1),Condition),(Status(2),Length)
      Equivalence(Status(3),Record_Number),(Status(5),Chain_Count)
      Equivalence(Status(7),Back_Pointer),(Status(9),Forward_Pointer)
      Common /Database_Status_Type/ Status

C**** Set up for the Lock_Descriptor_Array of the Sales data set.

      Integer*2  Lock_Descriptor_Array(22)
      Integer*2  Length_Of_Descriptor, Num_Of_Elements
      Character  Data_Set_Of_Descriptor*16
      Character  Data_Item_Of_Descriptor*16
      Character  Relative_Operator*2
      Character  Value_For_Data_Item*6
      Equivalence (Lock_Descriptor_Array(1), Num_Of_Elements)
      Equivalence (Lock_Descriptor_Array(2), Length_Of_Descriptor)
      Equivalence (Lock_Descriptor_Array(3), Data_Set_Of_Descriptor)
      Equivalence (Lock_Descriptor_Array(11),Data_Item_Of_Descriptor)
      Equivalence (Lock_Descriptor_Array(19),Relative_Operator)
      Equivalence (Lock_Descriptor_Array(20),Value_For_Data_Item)

C**** Set up for the Sales_Buffer of the Sales data set.

      Integer*2  Sales_Buffer(19)
      Integer*4  Account_Number
      Character  Stock_Number*8
      Integer*2  Quantity
      Integer*4  Price
      Integer*4  Tax
      Integer*4  Total
      Character  Purch_Date*6
      Character  Deliv_Date*6

      Equivalence (Sales_Buffer(1), Account_Number)
      Equivalence (Sales_Buffer(3), Stock_Number)
      Equivalence (Sales_Buffer(7), Quantity)
      Equivalence (Sales_Buffer(8), Price)
      Equivalence (Sales_Buffer(10),Tax)
      Equivalence (Sales_Buffer(12),Total)
      Equivalence (Sales_Buffer(14),Purch_Date)
      Equivalence (Sales_Buffer(17),Deliv_Date)
```

Main Body of Program

In the following portion of the program, the $hp3000_16$ compiler directive allows the FORTRAN 77 compiler to change the data alignment from a four-byte limit to a two-byte limit. For example, the non-alignment caused by the Quantity field in the Sales data set can be resolved using this directive.

```
$hp3000_16$
      Program Fortran_For_TurboIMAGEXL
C
C     This area will contain the main line for the
C     FORTRAN 77 example.
C
```

Obtaining Error Messages and Explanations

The following procedure implements the Get_Error_And_Explain routine of the sample program. In this procedure, DBEXPLAIN and DBERROR are called using FORTRAN 77. DBEXPLAIN interprets the contents of the status parameter and prints a message on $STDLIST. DBERROR returns a message in ERROR_Buffer, explaining the condition code returned by TurboIMAGE/XL. At the end of the procedure, users can choose to abort or continue the execution of this program. Note that aborting a process from within a transaction would result in an incomplete transaction. It is good programming practice to end your transaction, release your locks, and close any open database(s) before aborting your process.

```
C**********************************************************************
      Subroutine Get_Error_And_Explain
C     Access     :  Mode 1 - Shared Modified Access
C
C
C     Called By  :  Open_The_Database
C                   Get_Sales_For_Date
C                   Get_A_Customer_Record
C                   Get_A_Product_Record
C                   List_All_Customers
C                   Add_A_Product
C                   Update_A_Customer
C                   Delete_A_Product
C                   Rewind_Customer_Set
C                   Get_Data_Item_Info
C                   Close_The_Database

C
C
C     Calls      :  DBERROR
C                   DBEXPLAIN
C
$list Off
$Include 'comon1'
$list On
C     Prepare the error buffer for calls to DBERROR
C
      Character Error_Buffer_Text*80
```

```
      Integer*2 Error_Buffer(40)
      Equivalence (Error_Buffer(1),Error_Buffer_Text)


      Integer*4 Error_Length
      Integer*2 Answer
      Parameter (In=5,Out=6)

      Call DBERROR (Status,Error_Buffer,Error_Length)

      Write(Out,*)'-------------------------------------'
      Write(Out,10)Error_Buffer_Text
10    Format(A60)
      Write(Out,*)'-------------------------------------------'

      Call DBEXPLAIN (Status)

      Answer=0
      Write(Out,*)'---Enter, <1> to ABORT..., <2> to Continue >'
      Read (In,20) Answer
20    Format(I2)

      If (Answer.NE.1) Then
            Write(Out,*)'  Continuing......'
         Else
            Stop
      Endif
      Return
      End
```

Opening the Database

This procedure implements the Open_The_Database procedure of the sample program. All required values, such as, the password, are provided by the routine. Note that the password DO-ALL is followed by a semicolon because it is less that eight characters long; a blank can be substituted for the semicolon. Open_The_Database uses open mode 1, which is the shared modify access mode. Error trapping is done by referring all non-zero conditions to the Get_Error_And_Explain procedure.

```
C*************************************************************

      Subroutine Open_The_Database
C
C     ACCESS     : Mode 1 - Shared Modify Access (SMA) with locking required
C
C     CALLED BY  : Main Line
C
C     CALLS      : DBOPEN in mode 1 (SMA)
C                  Get_Error_And_Explain
C
$List Off
$Include 'comon1'
$List On
C**** Prepare the Base parameter of the DBOPEN.
C
      Mode1_SMA  = 1
      BaseName='  ORDERS;  '
```

```
      Pass_Word='DO-ALL;'

      Call DBOPEN (DBname,Password,Mode1_SMA,Status)
                  If (Condition.NE.0) Then
                      Call Get_Error_And_Explain
                  EndIf
      Return
      End
```

Retrieving All the Records on a Chain (with Item Level Locking)

This procedure implements the Get_Sales_For_Date routine of the sample program. Chained access is achieved using a call to DBFIND. The search item used for this call is Purch-Date. An item level lock is obtained on the value of the search item before the DBFIND call. After that, individual chain items are retrieved, until the end of chain is encountered. This is done using multiple calls to the DBGET procedure.

This routine traps two exceptional conditions:

1. Status condition from the DBFIND call, indicating that the chain head cannot be located.
2. Status 15 from DBGET, indicating the end of the chain.

The status interpretation routine permits you to either abort or continue with the execution after viewing all error messages.

```
C*************************************************************
      Subroutine Get_Sales_For_Date
C     ACCESS     : Mode 1 - Shared Modify Access
C
C     CALLED BY  : Main Line
C
C     CALLS      : DBLOCK in mode 5 (unconditional item level locking)
C                  DBFIND in mode 1 (chained access)
C                  DBGET in mode 5 (forward chain read)
C                  DBUNLOCK in mode 1 (unlock)
C*****             Get_Error_And_Explain (chained access)

$list off
$include 'comon1'
$list on

C**   The Input/Output indicator values
      Parameter    (In=5,Out=6)
      Integer*4    End_Of_Chain,No_Chain_Head
C**   Set up for the data set parameter.
      Character    Data_Set_Name_Is*16
      Integer*2    Sales_Detail(8)
      Equivalence  (Sales_Detail(1),Data_Set_Name_Is)

C**   Set up for the search item parameter.
      Character    Search_Item_Name_Is*16
      Integer*2    Search_Item_Name(8)
      Equivalence  (Search_Item_Name(1),Search_Item_Name_Is)

C**   Set up for the search value/argument parameter.
      Character    Search_Item_Value_Is*6
      Integer*2    Search_Item_Value(3)
```

```
      Equivalence  (Search_Item_Value(1),Search_Item_Value_Is)
      Parameter    (End_Of_Chain=15,No_Chain_Head=17)

C**   Set up for the predicate buffer used in item level locking.
      Num_Of_Elements        = 1
      Length_Of_Descriptor   = 21
      Data_Set_Of_Descriptor ='SALES;'
      Data_Item_Of_Descriptor='PURCH-DATE;'
      Relative_Operator      =' ='

C**   Accept the search value.
      Print*,'  Enter The Date of Purchase as (YYMMDD) >>> '
      Read (5,10) Search_Item_Value_Is
10    Format(A6)

C**   Request item level locks on all items identified by the search
C**   value.  A mode value of 5 indicates an item level lock request.
      Mode5_Unconditional =5
      Value_For_Data_Item = Search_Item_Value_Is
      Call DBLOCK (DBname,Lock_Descriptor_Array,Mode5_Unconditional,
     &             Status)
           If (Condition.NE.0) then
               Call Get_Error_And_Explain
           EndIf
C**   Locate all entries identified by the search value.
      Data_Set_Name_Is    = 'SALES;'
      Mode1_Chained_Read  = 1
      Search_Item_Name_Is = 'PURCH-DATE;'

      Call DBFIND (DBname,Sales_Detail,Mode1_Chained_Read,Status,
     &           Search_Item_Name,Search_Item_Value)

      If (Condition.NE.0) Then
                If (Condition.EQ.No_Chain_Head) Then
                 Print*,'_____________________________________________'
                 Print*,'|                                           |'
                 Print*,'|   No Such Entry In the Sales Data Set     |'
                 Print*,'|                                           |'
                 Print*,'|___________________________________________|'
                 Print*,'Hit Enter to Continue ..................... '
                        Read(5,*)
                 Else
                    Call Get_Error_And_Explain
                 EndIf
      Else

            Write(6,20)
      Write(6,30)
20    Format (' Acct-Number   Stock-Number  Qty  Price  Tax  Total ',
     &'Purch-Date   Deliv-Date ')
30    Format (' ------------------------------------------------- ',
     &'----------------------- ')

                 Mode5_Forward  = 5
                 List           = '@;'
               Do While (Condition.NE.End_Of_Chain)

                 Call DBGET (DBname,Sales_Detail,Mode5_Forward, Status,
```

```
&                     List, Sales_Buffer, Not_Used_Parm)
           If (Condition.NE.0) Then
               If (Condition.EQ.End_Of_Chain) Then
                   Print *,'-->End Of Chain, Hit Enter to Continue'
                   Read (5,*)
               Else
                   Call Get_Error_And_Explain
               EndIf
           Else
             Print*
             Print*,Account_Number,'          ',
&                   Stock_Number,'    ',
&                   Quantity,'  ',Price,'  ',Tax,'   ',Total,'  ',
&                   Purch_Date,'   ',Deliv_Date
           EndIf
        End Do
 EndIf
  Mode1_Unlock   =1
  Call DBUNLOCK (DBname,Sales_Detail,Mode1_Unlock,Status)
  If (Condition.NE.0) Then
      Call Get_Error_And_Explain
  EndIf
 Return
 End
```

Pascal

Portions of the model program presented at the beginning of this chapter are now shown here in Pascal. The examples perform specific tasks to illustrate the use of TurboIMAGE/XL intrinsics.

Data items are defined at the beginning of the sample program. TurboIMAGE/XL intrinsics must be declared for Pascal as external procedures. The procedure name is followed by the word "Intrinsic."

Type declarations declare names for data structure forms that will be used in allocating variables. Variable declarations allocate the variables of the program. Variables are defined with precise types or forms. Pascal string literals are delimited with single quotes (' '). Field and record names are separated with a dot (.), when referenced (for example, "base_name.baseid").

NOTE Because the Schema Processor, DBSCHEMA, upshifts alphabetic characters, programs must specify data set and data item names in all uppercase characters. Take note of this because Pascal does not require that you use uppercase characters.

For information on TurboIMAGE/XL data item lengths and type designators, refer to chapter 3. Tables 3-2 and 3-3 show the TurboIMAGE/XL type designators, sub-item lengths, and data types typically used to process them in Pascal.

NOTE All parameters must be on halfword boundaries and **cannot be odd-byte aligned**.

Defining Data Types, Variables, and Intrinsics

The following is part of the Pascal example program; it defines type declarations, variable declarations, and TurboIMAGE/XL intrinsics.

```
$Standard_Level 'HP_MODCAL'$
$hp3000_16$
Program Pascal_For_TurboIMAGEXL  (Input,Output);
Label 100;

     (* Define all your TurboIMAGE/XL constants. *)
Const
     End_Of_Chain           =15;       (* For DBGET Mode 5 *)
     End_Of_Data_Set        =11;       (* For DBGET Mode 2 *)
     No_Chain_Head          =17;       (* For DBFIND       *)
     No_Such_Entry          =17;       (* For DBGET Mode 7 *)
     Entry_Has_No_Data      =17;       (* For DBGET Mode 4 *)

     (* Define all your TurboIMAGE/XL record structures. *)
Type
                              (* for the base parameter *)
     Database_Name_Type     = Packed Record
```

```
                              BaseId        : Packed Array [1..2] of Char;
                              BaseName      : Packed Array [1..16] of Char;
                              End;
                              (* for the password parameter *)
      Database_Password_Type = Packed Array [1..10] of Char;

                              (* for the status parameter *)
      Database_Status_Type   = Packed Record
                              Condition          : ShortInt;
                              Length             : ShortInt;
                              Record_Number      : Integer;
                              Chain_Count        : Integer;
                              Back_Pointer       : Integer;
                              Forward_Pointer    : Integer;
                              End;
                              (* for the data set name parameter *)
      Data_Set_Name_Type     = Packed Array [1..16] of Char;

                              (* for data item names *)
      Data_Item_Name_Type    = Packed Array [1..16] of Char;

                              (* for the list parameter *)
      Data_Item_List_Type    = Packed Array [1..80] of Char;

                              (* for key items in manual masters *)
      Key_Item_Type          = Packed Array [1..40] of Char;

                              (* for the Sales data set of Orders DB *)
      Sales_Data_Set_Type    = Packed Record
                              Account_Number: Integer;
                              Stock_Number  : Packed Array [1..8] of Char;
                              Quantity      : ShortInt;
                              Price         : Integer;
                              Tax           : Integer;
                              Total         : Integer;
                              Purch_Date    : Packed Array [1..6]of Char;
                              Deliv_Date    : Packed Array [1..6]of Char;
                              End;
                              (* for item level locks in the Sales set *)
      Lock_Descriptor_Sales_Type     = Packed Record
            Length_Of_Descriptor       : ShortInt;
            Data_Set_Of_Descriptor     : Data_Set_Name_Type;
            Data_Item_Of_Descriptor    : Data_Item_Name_Type;
            Relative_Operator          : Packed Array [1..2]Of Char;
            Value_For_Data_Item        : Packed Array [1..6]Of Char;
            End;

                              (* for the lock buffer for the Sales set *)
      Lock_Descriptor_Sales_Array_Type   = Packed Record
            Num_Of_Elements              : ShortInt;
            Lock_Descriptor_Sales        : Lock_Descriptor_Sales_Type;
            End;

Var
            (*   Define all your global variables.          *)

            DBname                 :   Database_Name_Type;
            Password               :   Database_Password_Type;
```

```
Status                  :   Database_Status_Type;
Option                  :   ShortInt;
Mode                    :   ShortInt;
List                    :   Data_Item_List_Type;

(*   Define all TurboIMAGE/XL procedure calls that *)
(*   will be used in your application program.     *)

Procedure DBBEGIN       ;   Intrinsic;
Procedure DBEND         ;   Intrinsic;
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;
```

Obtaining Error Messages and Explanations

The following procedure implements the Get_Error_And_Explain routine of the sample program. In this procedure, DBEXPLAIN and DBERROR are called using Pascal. DBEXPLAIN interprets the contents of the Status parameter and prints a message on $STDLIST. DBERROR returns a message in Error_Buffer, explaining the condition code returned by TurboIMAGE/XL. At the end of the procedure, users can choose to abort or continue the execution of this program. Note that aborting a process from within a transaction would result in an incomplete transaction. It is good programming practice to end your transaction, release your locks, and close any open database(s) before aborting your process.

```
$Page$
Procedure Get_Error_And_Explain;
(*
      Access   :     Mode 1 - Shared Modified Access
                              The Orders database was opened in mode 1

      Called by:     Open_The_Database
                     Get_Sales_For_Date
                     Get_A_Customer_Record
                     Get_A_Product_Record
                     List_All_Customers
                     Add_A_Product
                     Update_A_Customer
                     Delete_A_Product
                     Rewind_Customer_Set
                     Get_Data_Item_Info
                     Close_The_Database
      Calls    :     DBERROR
                     DBEXPLAIN
*)
```

```
Var
             Error_Buffer          :    Packed Array [1..80] of Char;
             Error_Length          :    Integer;
             Answer                :    ShortInt;
Begin
             DBERROR (Status,Error_Buffer,Error_Length);

             Writeln('-----------------------------------------');
             Writeln(Error_Buffer);
             Writeln('-----------------------------------------');
             Writeln;

             DBEXPLAIN (Status);

             Answer:=0;
             Prompt( '---Enter, <1> to ABORT..., <2> to Continue >');
             Readln(Answer);
             If Answer <> 1 Then Writeln(' Continuing .........')
                Else Halt;
End;
```

Opening the Database

This procedure implements the Open_The_Database procedure of the sample program. All required values, such as the password, are provided by the routine. Note that the password DO-ALL is followed by a semicolon because it is less than eight characters long; a blank can be substituted for the semicolon. Open_The_Database uses open mode 1, which is the shared modify access mode. Error trapping is done by referring all non-zero conditions to the Get_Error_And_Explain procedure.

```
$Page$
Procedure Open_The_Database;

(*       Access   :  Mode 1 - Shared Modify Access (SMA) with locking required
       Called By:  Main Line
       Calls    :  DBOPEN in mode 1 (SMA)
                   Get_Error_And_Explain   *)

Begin
             Mode1_SMA             : Integer;

             DBname.BaseID         :='  ';
             DBname.BaseName       :='ORDERS; ';
             Password              :='DO-ALL;';
             Mode1_SMA             :=1;

             DBOPEN (DBname,Password,Mode1_SMA,Status);
             If Status.Condition <> 0 Then
                Get_Error_And_Explain;
End;
```

Retrieving All the Records on a Chain (with Item Level Locking)

This procedure implements the Get_Sales_For_Date routine of the sample program. Chained access is achieved using a call to DBFIND. The search item used for this call is Purch-Date. An item level lock is obtained on the value of the search item before the DBFIND call. After that, individual chain items are retrieved, until the end of chain is encountered. This is done using multiple calls to the DBGET procedure.

The routine traps two exceptional conditions:

1. Status condition 17 from the DBFIND call, indicating that the chain head cannot be located.
2. Status 15 from the DBGET call, indicating the end of chain.

The status interpretation routine permits you to either abort or continue with the execution of the program after viewing all error messages.

```
$Page$
Procedure Get_Sales_For_Date;

(*     Access   :  Mode 1 - Shared Modify Access
                          The Orders database was opened in mode 1.
       Called By:  Main Line
       Calls    :  DBLOCK in mode 5 (unconditional item level locking)
                   DBFIND in mode 1 (chained access)
                   DBGET  in mode 5 (forward chained read)
                   DBUNLOCK in mode 1 (unlock)
                   Get_Error_And_Explain                          *)

Var
       Lock_Descriptor_Array              : Lock_Descriptor_Sales_Array_Type;
       Sales_Detail                       : Data_Set_Name_Type;
       Search_Item_Name                   : Data_Item_Name_Type;
       Search_Item_Value                  : Packed Array [1..6]of Char;
       Sales_Buffer                       : Sales_Data_Set_Type;
       Not_Used_Parm                      : Shortint;
       Mode1_Chained_Read                 : Shortint;
       Mode5_Unconditional                : Shortint;
       Mode5_Forward                      : Shortint;
       Mode1_Unlock                       : Shortint;

Begin
       (* Prepare the lock descriptor buffer for obtaining item level  *)
       (* locks on the Sales data set.                                 *)
       With Lock_Descriptor_Array Do
            Begin
            Num_Of_Elements                 := 1;
            With Lock_Descriptor_Sales Do
                 Begin
                 Length_Of_Descriptor       := 21;
                 Data_Set_Of_Descriptor     :='SALES;';
                 Data_Item_Of_Descriptor    :='PURCH-DATE;';
                 Relative_Operator          :=' =';
                 End;
            End;

       Prompt ('    Enter The Date of Purchase as (YYMMDD) >>>  ');
```

```
Readln (Search_Item_Value);
Mode5_Unconditional          :=5; (* Request item level locks. *)

(* Append the user's input to the lock descriptor buffer. *)
Lock_Descriptor_Array. Lock_Descriptor_Sales.Value_For_Data_Item
                                          :=Search_Item_Value;


(* Place item level locks on all entries identified by *)
(* the value in the Search_Item_Value.                  *)
DBLOCK (DBname,Lock_Descriptor_Array,Mode5_Unconditional,Status);
If Status.Condition <> 0 then
   Get_Error_And_Explain;

Sales_Detail                       :='SALES;';
Search_Item_Name                   :='PURCH-DATE;';
Mode1_Chained_Read                 :=1;

(* Locate the chain identified by the value in the    *)
(* Search_Item_Value.                                 *)
DBFIND (DBname,Sales_Detail,Mode1_Chained_Read,Status,
         Search_Item_Name,Search_Item_Value);
If Status.Condition <>0 Then
   Begin
   If Status.Condition = No_Chain_Head Then
      Begin
      Writeln('*************************************');
      Writeln('* No Such Entry in the Sales Dataset  *');
      Writeln('* Please Try Again.                   *');
      Writeln('*************************************');
      Prompt ('Hit Enter To Continue ---------------->');
      Readln;
      End
   Else Get_Error_And_Explain;
   End
Else
   Begin
   Write('Acct-Number');
   Write('Stock-Number':14);
   Write('Qty':6);
   Write('Price':7 );
   Write('Tax':5);
   Write('Total':8);
   Write('Purch-Date':12);
   Write('Delive-Date':14);
   Write('-------------------------------------------------');
   Write('--------------------------');
   Writeln;

   (* Start retrieving all records in the current chain.  *)
   Mode5_Forward              :=5;
   List                       :='@;';


   While Status.Condition <> End_Of_Chain Do
         Begin
         (* Retrieve the contents of the entry which is at the   *)
         (* current record pointer.                               *)
```

```
                    DBGET
(DBname,Sales_Detail,Mode5_Forward,Status,List,Sales_Buffer,
                         Not_Used_Parm);
                    If Status.Condition= 0 Then
                       Begin
                       With Sales_Buffer Do
                            Begin
                            Writeln;
                            Write(Account_Number:10);
                            Write(Stock_Number:15);
                            Write(Quantity:6);
                            Write(Price:7 );
                            Write(Tax:5 );
                            Write(Total:7);
                            Write(Purch_Date:12);
                            Write(Deliv_Date:12);
                            End;
                       End     (* Check the status buffer for any condition *)
                    Else       (* codes not equal to zero.                  *)
                       Begin
                       If Status.Condition= End_Of_Chain Then
                          Begin
                          Writeln;
                          Writeln;
                          Writeln;
                          Prompt ('----> End Of Chain, Hit Enter to Continue');
                          Readln;
                          End
                       Else Get_Error_And_Explain;
                       End;
                    End;
              End;

           (* Release all locks acquired at the beginning of the process. *)
           Mode1_Unlock     :=1;
           DBUNLOCK (DBname,Sales_Detail,Mode1_Unlock,Status);
           If Status.Condition<>0 Then
              Get_Error_And_Explain
   End;
   $Page$
```

RPG

RPG contains language constructs that make calls to TurboIMAGE/XL intrinsics, rather than having the user code do the intrinsic calls directly. For example, RPG opens all files at the beginning of program execution, thereby calling DBOPEN for any databases named on File Description specifications. Likewise, DBCLOSE is automatically called for databases at the end of program execution. Another example is the RPG CHAIN operation which calls DBFIND and/or DBGET, depending on its usage.

A small set of TurboIMAGE/XL intrinsics have no RPG language equivalent and so cannot be accessed at the present time. These include DBERROR, DBEXPLAIN, DBBEGIN, DBEND, and DBMEMO. Because the sample program on which this RPG program is based contains calls to DBERROR and DBEXPLAIN, the RPG version cannot exactly match the functionality of the sample program. Instead, it displays the status value returned by TurboIMAGE/XL when an error occurs.

NOTE Because the Schema Processor, DBSCHEMA, upshifts alphabetic characters, programs must specify data set and data item names in all uppercase characters. Take note of this as RPG does not require that you use uppercase characters.

For information of TurboIMAGE/XL data item lengths and type designators, refer to chapter 3. Tables 3-2 and 3-3 show the TurboIMAGE/XL type designators, sub-item lengths, and data types typically used to process them in RPG.

Defining Data Types, Variables, and Intrinsics

The first five F-specs implement the OPEN-THE-DATABASE routine of the sample program in RPG. RPG automatically opens all databases (by calling DBOPEN) at the beginning of program execution. F-specs are used to declare how each database/set is to be opened and accessed.

In this program, the database is opened with mode 1 (shared modify access) by entry of L in column 66 on the KIMAGE line. This entry also allows user-controlled locking of the database/set/item to occur.

The data set is opened for chained sequential read accesses by entry of C in column 67 of the KIMAGE line. If an error occurs during opening of the database, the program will abort.

Note that RPG also closes all databases (by calling DBCLOSE) at the end of program execution, so there is no need for the user to call a separate routine to do this.

```
$CONTROL NOINFO,MAP,NAME=RTURBO
H                   L                          1
*************************************************************
*   Program name:   RTURBO                                  *
*   Description :   Example of RPG access to TurboIMAGE/XL  *
*************************************************************

FSALES   IC  F      38R 6AI    27 DISC
```

```
F                                                KIMAGE ORDERSLC
F                                                KITEM  PURCH-DATE
F                                                KLEVEL DO-ALL
F                                                KSTATUSSTAT
FTERMIN  ID  F      79           $STDIN
FTERMOUT O   V      79           $STDLST
*****************************************************************
*                   TABLE/ARRAY DECLARATIONS                    *
*****************************************************************
E                    ESC     1   1  1              Escape = 27

*****************************************************************
*                     INPUT RECORD LAYOUTS                      *
*****************************************************************
ISALES   NS
I                                    B   1   40ACCT#
I                                        5  12 STOCK#
I                                    6  13  140QTY
I                                    7  15  180$PRICE
I                                    5  19  220$TAX
I                                    8  23  260$TOTAL
I                                       27  320PDATE
I                                       33  380DDATE
ITERMIN  NS
I                                        1   8 OPTION
```

Main Body of Program

```
*****************************************************************
*                    CALCULATIONS MAINLINE                      *
*****************************************************************
C*   Execute GETSAL subroutine, then end program.
C*
C                     EXSR GETSAL
C                     SETON                     LR
```

Retrieving All the Records on a Chain (with Item Level Locking)

This subroutine implements the GET-SALES-FOR-DATE routine of the sample program. Chained access is achieved using the CHAIN operation which performs a DBFIND call and a DBGET call on the first execution, and then DBGET calls on subsequent executions for the same search value. Thus a loop is done on the CHAIN operation in order to retrieve all the entries in the data item chain. The routine traps two exceptional conditions: failure to find a chain head, and reaching end-of-chain.

```
*****************************************************************
*            G E T S A L   S U B R O U T I N E                  *
*****************************************************************
C           GETSAL    BEGSR

C*---------------------------------------------------------------
C*   Display prompt for date and read user input from screen.
C*
C                     EXCPT          CLEAR
C                     EXCPT          GETDAT
C                     READ TERMIN                  H0
```

```
C                     MOVELOPTION    DATE    6
C*---------------------------------------------------------------
C*   Do unconditional data item lock on DATE.
C*
C           DATE      LOCK SALES                  21
C   21                EXSR GETERR
C   LR                GOTO END1
C*---------------------------------------------------------------
C*   Loop on CHAIN operation to get all entries in chain.  On
C*   first occurrence of CHAIN for a unique value of DATE, RPG
C*   calls DBFIND, followed by DBGET.  On subsequent uses of
C*   CHAIN with the same value for DATE, only DBGET is called.
C*
C                     SETOF                     202122
C           LOOP1     TAG
C           DATE      CHAINSALES                2122
C*---------------------------------------------------------------
C*   No chain head found.
C*
C   21                EXCPT          CLEAR
C   21                EXCPT          NOHEAD
C   21                READ TERMIN                  H0
C   21                GOTO SKIP
C*---------------------------------------------------------------
C*   End-of-chain found.
C*
C   22                EXCPT          EOC
C   22                READ TERMIN                  H0
C   22                GOTO SKIP
C*---------------------------------------------------------------
C*   Any other error.
C*
C           STAT,1    IFGT 0
C                     EXSR GETERR
C   LR                GOTO END1
C                     END
C*---------------------------------------------------------------
C*   Print headings and data record.
C*
C  N20                EXCPT          CLEAR
C  N20                EXCPT          SALHDR
C  N20                EXCPT          LINHDR
C  N20                SETON                     20
C                     EXCPT          SALREC

C*---------------------------------------------------------------
C*   Loop back to do another CHAIN operation (DBGET).
C*
C                     GOTO LOOP1
C           SKIP      TAG
C*---------------------------------------------------------------
C*   Unlock the data item.
C*
C           DATE      UNLCKSALES                   24
C  N24                EXSR GETERR
C*
C           END1      ENDSR
```

Obtaining Error Messages and Explanations

The following subroutine implements the GET-ERROR-AND-EXPLAIN routine of the sample program. Because RPG as yet does not have access to DBEXPLAIN and DBERROR, this subroutine simply displays the TurboIMAGE/XL error number from the status array and then allows the user to either abort the program or continue its execution. If the user elects to abort, the LR indicator is set ON and the code which called this subroutine must test for LR and exit immediately to its caller, which in turn must exit to its caller, and so on.

```
*************************************************************
*              G E T E R R   S U B R O U T I N E            *
*************************************************************
C           GETERR    BEGSR
C                     EXCPT          ERRBUF
C           GETOPT    TAG
C                     READ TERMIN                  H0
C           OPTION    IFEQ "1"
C                     SETON                    LR
C                     ELSE
C           OPTION    IFEQ "2"
C                     EXCPT          ERRCON
C                     ELSE
C                     EXCPT          BADOPT
C                     GOTO GETOPT
C                     END
C                     END
C*
C                     ENDSR
```

Defining Output

```
*************************************************************
*                    OUTPUT RECORD LAYOUTS                  *
*************************************************************
O*-----------------------------------------------------------
O*   Display message for entry of invalid option.
O*

OTERMOUT E  1             BADOPT
O                                  23 "Invalid option - please"
O                                  33 " re-enter."
O*-----------------------------------------------------------
O*   Send 'Home' (Escape h) and 'Clear' (Escape J) to screen.
O*
O        E  1             CLEAR
O                         ESC        1
O                                    2 "h"
O                         ESC        3
O                                    4 "J"
O*-----------------------------------------------------------
O*   Display message for IMAGE End-Of-Chain condition.
O*
O        E 31             EOC
O                                  21 "-----> End of Chain, "
O                                  42 "Hit Enter to Continue"
O*-----------------------------------------------------------
```

```
O*   Display error message.
O*
O        E  1             ERRBUF
O                                  23 "-----------------------"
O                                  30 "-------"
O        E  1             ERRBUF
O                                  11 "IMAGE ERROR"
O                          STAT,1  21 "         *"
O                                  35 " HAS OCCURRED."
O        E  2             ERRBUF
O                                  23 "-----------------------"
O                                  30 "-------"
O        E  1             ERRBUF
O                                  23 "---Enter, <1> to Abort."
O                                  43 "..,  <2> TO Continue"
O*-------------------------------------------------------------
O*   Display message for continuing execution after error.
O*
O        E  1             ERRCON
O                                  15 "Continuing....."
O*-------------------------------------------------------------
O*   Display prompt for input of DATE.
O*
O        E  1             GETDAT
O                                  23 "Enter The DATE of Purch"
O                                  38 "ase as (YYMMDD)"
O*-------------------------------------------------------------
O*   Display Line Header (dashes).
O*
O        E  1             LINHDR
O                                  23 "-----------------------"
O                          *PLACE  46
O                          *PLACE  69
O                                  76 "-------"
O*-------------------------------------------------------------
O*   Display message that no IMAGE chain head was found.
O*
O        E  1             NOHEAD
O                                  23 "***********************"
O                                  39 "****************"
O        E  1             NOHEAD
O                                  23 "* No Such Entry in the "
O                                  39 "Sales Dataset  *"
O        E  1             NOHEAD
O                                  23 "* Please Try Again.    "
O                                  39 "               *"
O        E  1             NOHEAD
O                                  23 "***********************"
O                                  39 "****************"
O        E  1             NOHEAD
O                                  23 "Press Enter To Continue"
O*-------------------------------------------------------------
O*   Display Header line for listing of Sales records.
O*
O        E  1             SALHDR
O                                  13 "Acct-Number  "
O                                  28 "Stock-Number   "
O                                  33 "QTY  "
```

```
   O                                   40 "Price  "
   O                                   46 "Tax   "
   O                                   53 "Total  "
   O                                   66 "Purch-Date   "
   O                                   79 "Deliv-Date   "
   O*-------------------------------------------------------------
   O*   Display Sales record line.
   O*
   O        E  1             SALREC
   O                         ACCT#     10 "        0 "
   O                         STOCK#    25
   O                         QTY       31 "    0 "
   O                         $PRICE    38 "     0 "
   O                         $TAX      43 "   0 "
   O                         $TOTAL    51 "      0 "
   O                         PDATE Y   62
   O                         DDATE Y   75
   ** Following record contains Escape character (ASCII 27) in column 1
<--- ASCII 27
```

7 Logging and Recovery

This chapter discusses how to maintain database consistency; that is, how to log logical transactions and recover a TurboIMAGE/XL database from a system failure or program abort. This chapter is divided into the following major sections:

- Recovery Options
- Logical Transactions
- Dynamic Roll-Back Recovery
- Intrinsic Level Recovery
- Logging Preparation
- Logging Status
- Logging Maintenance
- TurboSTORE/iX 7x24 True-Online Backup
- Roll-Forward Recovery
- Roll-Back Recovery
- DBRECOV Commands Used With Roll-Forward and Roll-Back Recovery
- Record Tables
- Post-Recovery Options
- The Mirror Database

A quick guide to recovery and logging options is found in appendix G, "Recovery and Logging Quick Reference."

Database Utilities Used in Logging and Recovery

The TurboIMAGE/XL database is maintained using various TurboIMAGE/XL database utilities which are fully described in chapter 8. Because the following database utilities are used in logging and recovery, they are also referred to in this chapter:

DBUTIL Creates and maintains the database.

DBUNLOAD Copies data to specially formatted tape volumes.

DBLOAD Loads data from backup volumes (DBUNLOAD tape) into the database.

DBSTORE Stores a database to tape. You may also use TurboSTORE/iX 7x24 True-Online Backup with ONLINE=START or ONLINE=END option.

DBRESTOR Restores a database from backup volumes (DBSTORE tape) to disk.

If you use TurboSTORE/iX 7x24 True-Online Backup with ONLINE=START or ONLINE=END option to store your database, you can use the MPE RESTORE command which invokes TurboSTORE/iX 7x24 True-Online Backup to restore the database.

DBRECOV Recovers a database from a log file, even when you used TurboSTORE/iX 7x24 True-Online Backup with ONLINE=START or ONLINE=END option to store the database. The DBRECOV utility allows you to set control commands and create individual user recovery files. The information from these files enables you to inform each user where to resume transactions within the database.

Database utilities can be run in either job or session mode. With the exception of the DBUTIL >>SHOW command, the utilities DBUTIL, DBSTORE, DBRESTOR, DBUNLOAD, and DBLOAD all require a logon in the group and account that contains the database root file. Consequently, these utilities cannot be used with a remote database unless you initiate a remote session and run the utility as part of that session. The DBUTIL, DBSTORE, and DBRESTOR utilities do not allow you to use the MPE/iX FILE command to equate a database or database-access file.

CAUTION DBUNLOAD and DBLOAD do allow MPE/iX FILE commands to equate a database and can redirect the database to a different file. Except in a controlled environment, you should not use the MPE/iX FILE command to redirect a database or database-access file to a different file because that file can be purged easily.

The DBRECOV utility is not included in the discussion above because it is an exception. With DBRECOV, MPE/iX FILE commands are permissible and do not require a logon to the same group and account as the log file. However, DBRECOV must be invoked on the system where the database resides.

You can operate the database utilities if you are the database creator or if you know the maintenance word. If no maintenance word is defined, only the database creator can execute the database utilities. The exception to this rule is that a user with system

manager (SM) capability can use the `DBUTIL >>SHOW` command on any database without having to supply the maintenance word.

Use this chapter together with chapter 8 which gives the syntax of the database utilities and commands.

Recovery Options

The logging and recovery system is used to bring databases back to the same state at the time of a system failure; this state does not include incomplete transactions. TurboIMAGE/XL provides several types of recovery options:

- Default recovery mode
- Dynamic roll-back recovery
- Intrinsic Level Recovery (ILR)
- Roll-forward recovery
- Roll-back recovery
- Mirror database

The recovery options are described briefly below. Dynamic roll-back recovery, ILR, roll-forward recovery, roll-back recovery, and the mirror database are discussed in detail later in this chapter. A brief description of these recovery and logging options is found in appendix G, "Recovery and Logging Quick Reference." Use the information in this chapter and in appendix G to determine which recovery and logging options to choose. The recovery option is based on available database backup, logging resources, and performance requirements.

Default Recovery Mode. In default recovery mode, TurboIMAGE/XL uses an MPE/iX file system service, Transaction Management (XM), to ensure the structural integrity of the database following a system failure. All modifications to the database (DBPUTs, DBDELETEs, and DBUPDATEs) are automatically logged to an MPE/iX XM log file. However, this XM log file is only written to disk when one of the following situations is true:

- A system-specified time has elapsed.
- A request is made by a process to flush the log file to disk, for example, a call to DBEND mode 2 or 4.
- The XM buffer is full.

XM ensures that the TurboIMAGE/XL intrinsics are applied to the log file in a serial manner. If a system failure occurs, those completed intrinsics that have not been written to disk are not recovered. Thus, one or more completed DBPUTs, DBDELETEs, and DBUPDATEs can be lost, but the internal structure of the database remains consistent. *Recovery with DBRECOV must be performed at system startup time before anyone modifies the database.*

Dynamic Roll-Back Recovery. Dynamic roll-back recovery is a method of recovery that uses XM. XM ensures the physical and logical integrity of the databases. Dynamic roll-back allows a more timely recovery of databases than is possible with DBRECOV. Dynamic roll-back eliminates the overhead incurred when a database is enabled for user logging and permits database access to continue, even when recovering a database. It provides the most thorough protection for logical database transactions. Dynamic transaction can span one or more databases. The dynamic transaction spanning multiple databases is termed **dynamic multiple database transaction**, or **DMDBX**. Dynamic roll-back handles both program aborts and system failures without downtime for recovery,

and your database will not be left with an incomplete transaction. Dynamic roll-back also allows programs to selectively abort any ongoing transaction.

TurboIMAGE/XL uses XM to dynamically roll back uncommitted dynamic transactions. Dynamic roll-back requires using the three intrinsics: DBXBEGIN, DBXEND, and DBXUNDO.

Dynamic roll-back recovery can be used with roll-forward recovery to handle disk media failures.

Intrinsic Level Recovery (ILR). Intrinsic Level Recovery (ILR) is a recovery method provided within TurboIMAGE/XL. ILR ensures that all completed DBPUTs and DBDELETEs are recovered. On TurboIMAGE/XL, ILR is equivalent to calling DBEND mode 2 after every intrinsic. It is recommended that ILR not be used. It is not required for the DBRECOV roll-back recovery. See "Intrinsic Level Recovery" in this chapter for more information.

Roll-Forward Recovery. Roll-forward recovery is another method of recovery provided within TurboIMAGE/XL. It is used to ensure the structural and logical integrity of the database. Roll-forward recovery is used after a hard system failure such as a disk head crash or after problems occurring while roll-back recovery is in process.

Roll-forward recovery requires user logging and a backup copy of the database. The recovery time needed is generally more than roll-back recovery. The database backup may have been done using DBSTORE, TurboSTORE/iX 7x24 (with ONLINE=START or ONLINE=END option), or other similar programs and must be synchronized with the current log file.

If DBSTORE was used for storing the database, DBRESTORE needs to be used to restore the database. The correct log file is the first log file and recovery commences from the beginning of the log file.

If TurboSTORE/iX 7x24 True-Online Backup was used (with ONLINE=START or ONLINE=END option) to back up the database, the RESTORE command of MPE is used to restore the database. The recovery may even commence from the middle of a log file in use when backup was initiated. To find out the logfile from which recovery will start, use the SHOW *databasename* ALL command of DBUTIL. Refer to the TurboSTORE/iX True-Online Backup discussion later in this chapter.

Roll-Back Recovery. Roll-back recovery is another method of recovery provided within TurboIMAGE/XL. It is used to ensure the structural and logical integrity of the database. Roll-back recovery is used after a "soft" system crash such as a system failure or loss of working memory.

Roll-back recovery requires user logging. A backup copy of the database is recommended (for protection in the event of a hard disk failure), but not required. The time taken to perform roll-back is generally much less than roll-forward recovery.

Mirror Database. In addition to the recovery methods mentioned previously, roll-forward recovery can be used to mirror a database for constant access or "high availability" while providing controlled maintenance using a DBRECOV feature called STOP-RESTART. Backups and down-time can be regulated with this maintenance method which consists of two identical databases on two separate computer systems. The mirror database resides on the secondary system and is maintained with user logging, DBRECOV, and periodic DBSTOREs.

Logging and Recovery Considerations

To ensure database integrity, the following responsibilities need to be assigned to specific individuals:

- Enabling and disabling the logging and recovery processes.
- Creating database backup copies and synchronizing with log files.
- Performing actual recoveries when required.

NOTE In the event of a system failure and subsequent recovery operation when using private volumes, logging will not resume until these private volumes have been mounted. Enter the MPE/iX VMOUNT command into the SYSSTART file to resume logging.

The overhead required by the logging process depends on the number and type of modifications that are logged and the database structure. Both overhead and recovery time also depend on the type of recovery being used. For roll-forward recovery, the time needed for recovery depends on the number of transactions that were written to the log file following the last backup of the database. For roll-back recovery, the time needed to roll back the last incomplete transaction is generally much less than roll-forward recovery.

Dynamic roll-back recovery can process program aborts. This is significant because the TurboIMAGE/XL logging and recovery system is not intended to be a solution for transactions that fail to complete in real time due to a program abort. Because subsequent transactions may be dependent on a transaction interrupted by a program abort, the recovery system does not suppress transactions that fail for this reason. Instead, TurboIMAGE/XL logs an abnormal DBEND to the log file so that the transaction can be recovered.

NOTE Transactions that fail due to program aborts can be suppressed with the NOABORTS control option in DBRECOV as long as all processes are stopped immediately after a program abort and the database is 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 recover transactions that logged up to a time preceding the program abort. See the discussion on DBRECOV in chapter 8.

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 database. The log file provides information about previous entries as well as the current state of the database. The DBMEMO logging intrinsic, containing user text, provides a method of accessing and interpreting the log files.

Logical Transactions

A Definition

TurboIMAGE/XL logging and recovery (via DBRECOV) provide the ability to restore the database to a consistent state after a system failure. To understand how this is done, it is important to understand the concept of a logical transaction. A logical transaction is a sequence of one or more procedure calls that are considered one logical unit of work. Table 7-1. describes the types of logical transactions: static, multiple database, dynamic, and dynamic multiple database.

Table 7-1. Types of Logical Transactions

Transaction	Definition
Static	A logical transaction that begins with a DBBEGIN call and ends with a DBEND call. A static transaction spans only one database and uses DBBEGIN mode 1 and DBEND mode 1 or 2.
Multiple database	A logical transaction that begins with a DBBEGIN mode 3 or 4 call and ends with a DBEND mode 3 or 4 call. A multiple database transaction (MDBX) spans more than one database and can be recovered with roll-back or roll-forward recovery. Programmers may be tempted to call DBBEGIN twice (once for each database), update both databases, 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 the first database and its suppression on the second. *To perform a transaction accessing multiple databases, use a multiple database transaction.*
Dynamic	A logical transaction that begins with a DBXBEGIN mode 1 call and ends with a DBXEND mode 1 or 2 call. A dynamic transaction spans only one database. A call to DBXUNDO or a program abort will cause a dynamic transaction to be rolled back dynamically. If a system failure occurs, the dynamic transaction will be rolled back at the first call to DBOPEN for the database after the system is restarted. However, in the event of a media failure, DBRECOV roll-forward recovery can be used on dynamic transactions as long as user logging has first been enabled.
Dynamic multiple database	A dynamic transaction that spans multiple databases and begins with a DBXBEGIN mode 3 call and ends with a DBXEND mode 3 call. The total number of databases then can be included in a dynamic multiple database transaction is 15. The *baseid* parameter of DBXBEGIN mode 3 includes the number of databases and their respective base-ids. A call to DBXUNDO or a program abort will cause a DMDBX to be rolled back dynamically. In essence, modifications to all databases involved in DMDBX will be rolled back. If a system failure occurs, the dynamic transaction will be rolled back at the first call to DBOPEN for the databases after the system is restarted. However, in the event of a media failure, DBRECOV roll-forward recovery can be used on dynamic transactions as long as user logging has first been enabled.

Table 7-1. Types of Logical Transactions

Transaction	Definition
	Unlike a dynamic transaction for one database, DMDBX requires that a DBCONTROL mode 7 be done once for every database you want to include in a DMDBX, after DBOPEN of that database and before using it in the DBXBEGIN intrinsic. DBCONTROL mode 7 enables the database for deadlock detection, which when encountered, returns an error 26 instead of triggering a process hang. If the calling process is logging, DBXBEGIN, DBXEND, and DBXUNDO cause a log record to be written to the log file to identify the beginning, end, and roll-back, respectively, of a dynamic transaction. For DMDBX, logging should be either disabled or enabled for all databases involved in the DMDBX. In addition, if logging is enabled, the same logid needs to be used for all databases in the DMDBX as well. In case of DMDBX when logging is enabled, multiple log records, one for each database, will be written to the log file.

If logging is specified and DBBEGIN/DBEND (static transactions) or DBXBEGIN/DBXEND (dynamic transactions) are not used, TurboIMAGE/XL considers each DBPUT, DBDELETE, and DBUPDATE to be a single logical transaction. While a transaction is executing, the database is considered to be in an inconsistent state. Thus, each transaction takes the database from one consistent state to another.

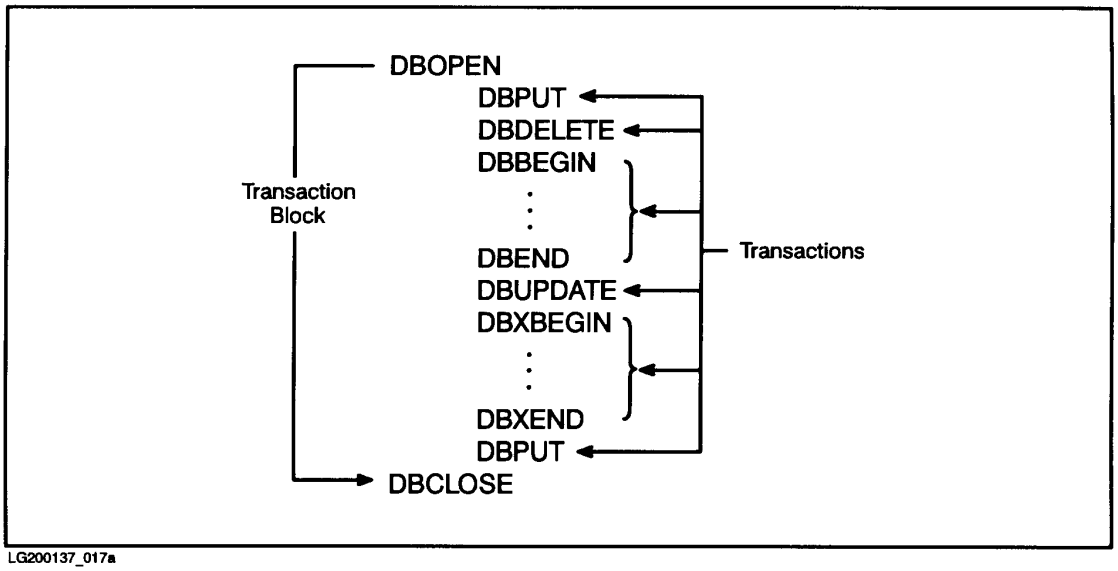
For example, consider the manual master data set CUSTOMER in the ORDERS database, with the addition of a new field, YTDSALES, indicating the total value of the year-to-date sales for each customer. A one-step transaction might involve updating a particular customer's address. Adding a new sales item is a two-step transaction: adding an entry to the SALES detail data set and updating the YTDSALES item in the CUSTOMER master set. The database 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 database is said to be inconsistent. After the second step, modifying the YTDSALES item in the CUSTOMER data set, the database is returned to a consistent state.

If the system fails while the database is being modified, database integrity could be affected. Logical inconsistency could result if the failure occurs between modifications of a multiple- step transaction, as illustrated by the example in the preceding paragraph. Secondly, if AUTODEFER is enabled, structural damage (such as, broken chains) can result if the failure occurs during the execution of a TurboIMAGE/XL intrinsic.

Because the recovery system is designed to restore the database 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 database modifications may be lost upon recovery, the resulting database will be consistent. To this end, each user application should indicate the beginning and end of each transaction by using a DBBEGIN and DBEND pair or a DBXBEGIN and DBXEND pair. (Refer to chapter 4 for more information on transactions.)

Figure 7-1. illustrates the concept of a logical transaction using a static and a dynamic transaction. Note that a **transaction block** is also illustrated. A transaction block consists of all transactions between a call to DBOPEN and a call to DBCLOSE. For further information about transaction blocks, refer to "FILE Command" later in this chapter.

Figure 7-1. Transactions and Transaction Blocks



Locking Requirements for Logical Transactions

DBRECOV requires that all multiple-intrinsic database transactions execute independently of all other transactions. Transaction independence within the database can be ensured in a user program by locking data before a transaction and then releasing locks after a DBEND or DBXEND is called, thus eliminating the possibility of another user modifying the same data at the same time. The following example may clarify the need for locking data to be modified.

Suppose transaction A consists of adding two records to the database that are later modified by transaction B. Transaction B is dependent upon transaction A, because the 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 database from one consistent state to another. A database may be in an inconsistent state during a transaction. Therefore, if transactions A and B are executing concurrently without locking, transaction B may be viewing the database in an inconsistent state and consequently could be generating invalid results. However, if transaction A locks the data and completes properly, this problem is avoided because transaction B cannot access the data until transaction A has released its locks.

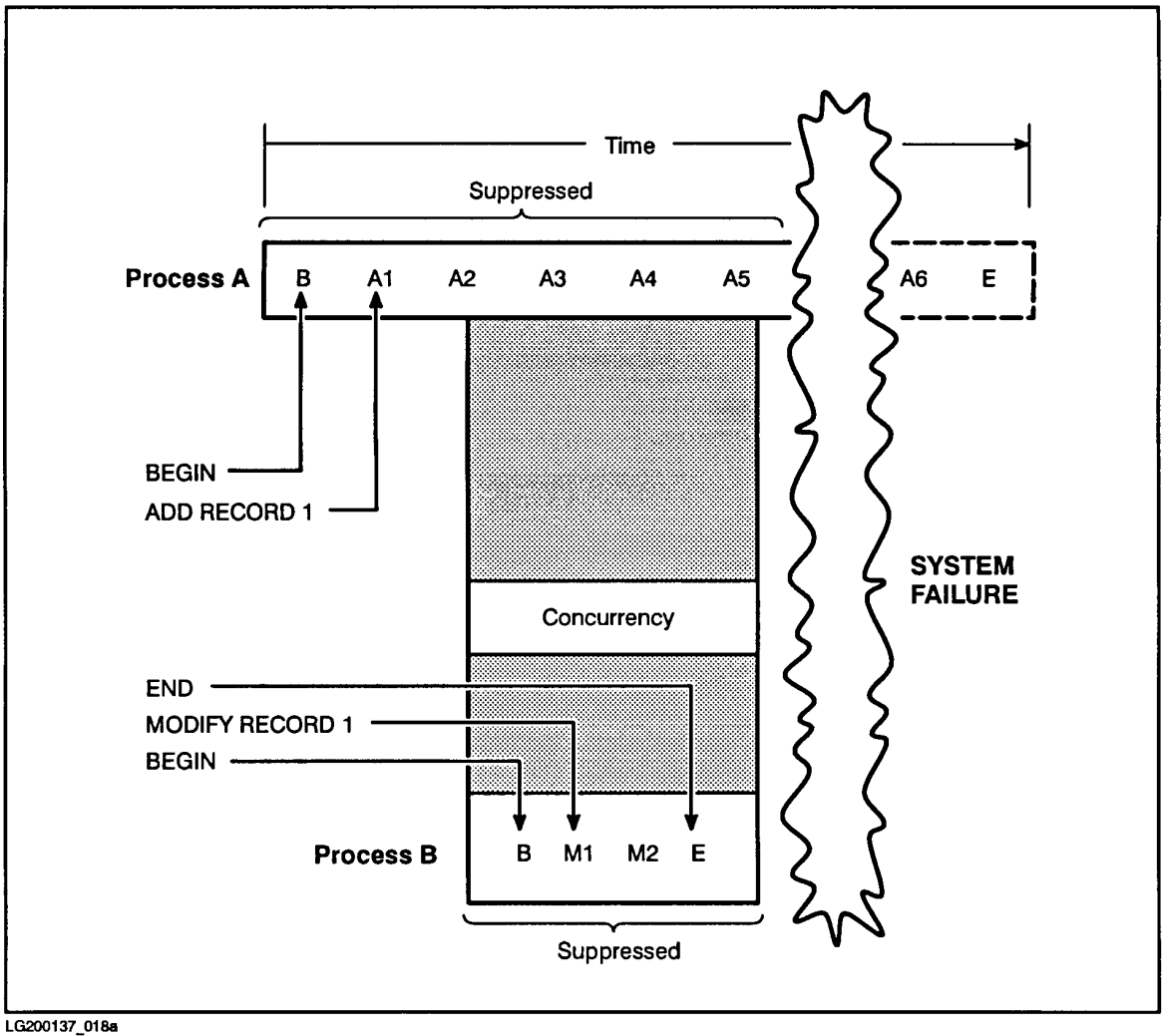
A second problem due to inadequate locking affects suppression of transactions by the recovery system (see Figure 7-2.) Suppose transaction A intends to add six records to the database, and after adding three records, transaction B is executed by another process. Transaction B concurrently modifies one of the records added by transaction A and then completes. Suppose that at this time, the system fails and recovery is executed. Because

transaction A failed to complete, all of its record additions are suppressed. Because 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 database employ locking correctly. Transactions attempting to access the same data concurrently are serialized by the locking mechanism.

Locking and Transaction Interdependence

To maximize the extent of recovery, locking should be used with logging to eliminate interdependence of concurrent static and multiple database transactions. Locking by logical transaction (that is, DBBEGIN and DBEND or DBXBEGIN and DBXEND) guarantees the logical consistency of the database. For roll-back recovery, locking by logical transaction ensures that all incomplete transactions are backed out of the database. For roll-forward recovery, transaction locking is recommended.

Figure 7-2. Suppression of Transactions Due to Inadequate Locking



Locking Examples

Examples of the two recommended locking schemes follow:

Single Lock Strategy

```
DBLOCK for account 2,18,34      Lock should precede DBBEGIN call.
DBBEGIN
DBGET 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 18
DBGET data account 18
DBDELETE data account 18
DBEND
DBUNLOCK for all accounts       DBUNLOCK must be last call.
```

CAUTION Use caution when employing a multiple lock strategy requiring Multiple RIN (Resource Identification Number) capability, also known as MR capability; refer to appendix D for information. 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. The recommendation is to call DBLOCK first, because there is no way of knowing how long DBLOCK will have to wait to acquire the lock after the transaction has begun. For additional locking information, refer to "Using the Locking Facility" in chapter 4.

Locking and Dynamic Transactions

Because dynamic transactions can be rolled back by calling DBXUNDO and are automatically rolled back in case of a system failure or program abort, TurboIMAGE/XL requires that a dynamic transaction be independent of all other transactions. When the database is opened in access mode 3 or 4, transaction independence is guaranteed because the program is the only modifier of the database.

When the database is opened in access mode 1, dynamic transactions require the programmer to use strong locking. A call to DBUNLOCK *must occur after* the call to DBXEND. Failure to follow this sequence after a call to DBPUT, DBDELETE, or DBUPDATE will cause an error. Intrinsic calls within the dynamic transaction *must* have covering locks. If an error

is returned indicating the failure of the intrinsic, the status condition must be checked before proceeding further or rolling back the entire transaction.

As with nondynamic transactions, the placement of DBLOCK calls either before or after DBXBEGIN is up to the programmer. A program with Multiple RIN (MR) capability can apply multiple locks with a dynamic transaction. MR capability is described in appendix D.

Dynamic transactions are not allowed when a database is opened in mode 2, because mode 2 does not enforce locking or guarantee transaction independence.

An example of strong locking follows:

```
DBLOCK for account 2,18,34      Lock should precede DBXBEGIN call.
DBXBEGIN
DBGET data for account 2
DBPUT data for account 34
DBGET data for account 18
DBDELETE data for account 18
DBXEND
DBUNLOCK for all accounts       DBUNLOCK must follow DBXEND
                                call.
```

Dynamic Roll-Back Recovery

Dynamic roll-back allows a more timely recovery of databases than is possible with DBRECOV. Dynamic roll-back eliminates the overhead incurred when a database is enabled for user logging and permits database access to continue, even while other users are accessing a database.

Using XM, uncommitted logical transactions can be rolled back dynamically (online) while other database activity is occurring. This is accomplished through the use of three intrinsics: DBXBEGIN, DBXEND, and DBXUNDO.

DBXBEGIN and DBXEND mark the beginning and end of the dynamic transaction which can be for one or multiple databases. The dynamic transaction can be rolled back in the following ways:

- Programmatically with a call to DBXUNDO.
- Automatically when the application aborts or a system failure occurs within the transaction.

In any case, those transactions begun after the call to DBXBEGIN that do not have a corresponding call to DBXEND will be rolled back. When you use DBXUNDO, your program logic must ensure that it does *not* call DBXEND subsequently; otherwise, you will get an error.

Use the following sequence of operations when modifying a database with dynamic transactions:

1. Call DBLOCK for each database to be included in the dynamic transaction to lock all data that must not be changed by other processes during the transaction. This includes data to be read and data to be modified.
2. If you wish, read data using DBFIND and DBGET to determine the necessary modifications.
3. If this is a multiple database transaction, ensure that DBCONTROL mode 7 is done at least once for each of the databases before including in the dynamic transaction.
4. Call DBXBEGIN to declare the beginning of modifications.
5. Make modifications using DBPUT, DBDELETE, or DBUPDATE.

   For every DBPUT, DBDELETE, or DBUPDATE, the status must be checked before proceeding further. If an error is returned indicating the failure of the intrinsic, the choices are:

   a. Call DBXEND. The successful modifications completed within this dynamic transaction will not be rolled back.

   b. Call DBXUNDO to roll back the entire transaction. Even successful modifications completed within one or more databases included in this dynamic transaction will be rolled back. That is, for DMDBX, DBXUNDO affects all databases included in the DBXBEGIN call.

   c. Continue with the remainder of the dynamic transaction even though this intrinsic

failed. As the application designer, you should be very cautious when taking this option as the modification to the database in this intrinsic did not take place.

6. Call DBXEND to declare the end of the modifications. If DBXUNDO was used in step 4 to roll back a transaction, your program logic should ensure that the subsequent call to DBXEND is not processed.
7. Call DBUNLOCK to release all of the locks.

Dynamic roll-back requires strong locking as discussed previously under "Locking and Dynamic Transactions." Calling DBUNLOCK after a call to DBPUT, DBDELETE, or DBUPDATE within a dynamic transaction will return an error because the call to DBUNLOCK must occur *after* the call to DBXEND. If necessary locks are not acquired before calling DBXBEGIN, covering locks must be used on the intrinsic calls within the dynamic transaction, otherwise an error will be returned.

It is essential that you check the status after each intrinsic. If a database intrinsic fails, you may end the transaction by calling DBXEND or roll back the entire transaction by calling DBXUNDO. You may be able to continue with the transaction despite the intrinsic failure, however, you must account for the intrinsic failure. The outcome varies based on your application design.

Do not use DBCLOSE mode 1 (close the database) inside an active dynamic transaction. This use of DBCLOSE will cause your dynamic transaction to be rolled back, the locks will be released, the database will be closed, and subsequent use of DBXEND or DBXUNDO will return an error.

If your dynamic transaction is very long causing XM to reach its limit of the log buffer space allowed per process, it will become a stalled transaction, and you cannot continue any further. This stalled transaction will be rolled back and the process is terminated.

If the system aborts or TurboIMAGE/XL encounters an internal error, the active memo records are saved in a file. All of the unrecovered memo records for a particular database are held in the *dbname*00 file. It resides in the same group and account as the database, and each database has only one *dbname*00 file. The first DBOPEN of an unrecovered database initiates the recovery using the *dbname*00 file.

User logging (discussed later in this chapter) is not required for dynamic roll-back recovery, but it is recommended to guard against a media failure.

Intrinsic Level Recovery

Intrinsic Level Recovery (ILR), one of the TurboIMAGE/XL recovery options, closely resembles default mode recovery. The distinction that ILR provides is an enforced flushing of the Transaction Management (XM) log file after every intrinsic. This enforced flushing emulates the immediate posting of intrinsics by TurboIMAGE/V and should only be used if immediate posting is required and DBEND mode 2 cannot be used to provide the immediate posting.

NOTE Intrinsic Level Recovery is not recommended. This feature is included for compatibility with MPE V programs only.

To enable a database for ILR, run DBUTIL and use the >>ENABLE command. For example:

```
:RUN DBUTIL.PUB.SYS
>>ENABLE database name FOR ILR
ILR is enabled
```

When the database is enabled for ILR, TurboIMAGE/XL sets a flag in the database root file. To determine if ILR has been enabled for a database, either use the DBUTIL >>SHOW command or programmatically call DBINFO with mode 402. Note that ILR is not required for roll-back recovery.

To discontinue using ILR on a database, use the DBUTIL >>DISABLE command. When ILR is disabled by the user, TurboIMAGE/XL clears the flag in the database root file.

NOTE If you have enabled the AUTODEFER option in DBUTIL, ILR cannot be used as the database recovery method. The following message is printed at the terminal if the user attempts to enable ILR when AUTODEFER is already enabled for the database:

```
AUTODEFER MUST BE DISABLED BEFORE ILR CAN BE ENABLED
```

The user should disable AUTODEFER and enable ILR using the DBUTIL >>ENABLE command. For more information on AUTODEFER, refer to chapter 8.

Logging Preparation

User logging is required for roll-forward recovery, roll-back recovery, and to maintain a mirror database. It is not required for dynamic roll-back recovery, but is recommended to protect against a hard disk failure. To prepare a database for user logging, you must set a log identifier (*logid*) into the database root file. The log identifier can be associated with an existing log file, in which case you can begin with step 6 below if you know the log identifier and password. Note that to recover a database using a log file, you must either be the creator of the *logid*, or supply the maintenance word and have system manager (SM) or operator (OP) capability.

Assuming you intend to create a new log identifier, you should take the following steps:

1. Check MPE/iX logging configuration.
2. Acquire logging capability.
3. Determine whether the log file will reside on tape or disk.
4. If logging to disk, build the log file.
5. Create the log identifier.
6. Set the log identifier into the database.
7. Set flags for the database backup copy.
8. Make a backup copy of the database. This step is required when using roll-forward recovery and recommended (for protection in the event of a hard disk failure), but not required, when using roll-back recovery.

This is a one-time procedure. The logging maintenance operations are performed on a regular basis, perhaps daily (refer to "Logging Maintenance" later in this chapter).

Step 1--Checking MPE/iX Logging Configuration

It is recommended that you check the logging configuration for adequate capacity before using the MPE/iX LOG command described later in this section. This precautionary step can prevent the downtime required to reconfigure the logging capacity. You need to check the number of user logging processes allowed on the system and the number of users that can access a single user logging process. For information on setting up these numbers at system startup time, refer to the *System Operation and Resource Management Reference Manual*.

Step 2--Acquiring Logging Capability

You must have MPE/iX logging (LG) or operator (OP) capability to use the following MPE/iX commands: `GETLOG`, `RELLOG`, `ALTLOG`, `CHANGELOG`, and `LISTLOG`. You must have LG or OP capability also if you intend to open a database with logging enabled. Logging capability is acquired through the MPE/iX system manager and account manager commands.

First, the system manager provides the account logging capability by using the MPE/iX

NEWACCT command for a new account, or the ALTACCT command for an existing account, as follows:

```
:NEWACCT acctname,mgrname;CAP=capability list   (include LG)
:ALTACCT acctname;CAP=capability list           (include LG)
```

Next, the account manager can provide logging capability to individual users by using the NEWUSER command for new users, or the ALTUSER command for an existing user, as follows:

```
: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
```

Refer to the *MPE/iX Commands Reference Manual* for information on other MPE/iX user logging commands, including these listed here:

RELLOG	Removes a log identifier.
ALTLOG	Alters an existing log identifier.
LISTLOG	Lists the current log identifiers.

Any messages that are followed by (ULOGERR#) or (ULOGMSG#) are MPE/iX errors or system messages.

Step 3--Logging to Tape or Disk

You must choose whether to log to tape or disk. The overhead required by the logging operation is comparable on disk or tape. However, other factors should be considered.

For roll-back recovery when logging to tape, the database must be in the system volume set. For roll-back recovery when logging to disk, the database and the log file must be in the same volume set. Logging to tape is the more secure option, because a log file residing on tape is less susceptible to damage from possible system failure than a disk log file. Refer to appendix G for more information on considerations when logging to disk and tape.

For allocating resources, logging to tape requires that the system be able to make a tape drive available as long as the database is accessible for modification. If the decision is made to log to disk, you must use the MPE/iX BUILD command to create a new file and allocate space on disk, as described in step 4. This allocation must be generous enough to avoid any possibility of filling the log file to capacity.

Step 4--Building a Log File for Logging to Disk

This step is required only when logging to disk. You must build the new file and allocate space for it on disk by using the MPE/iX BUILD command below.

Syntax

:BUILD *logfile*;CODE=LOG;DISC= [*numrec*] [, [*numextents*] [,*initial loc*]] [;DEV= [*device*]]

Parameters

logfile is the name of the log file being built, as specified in the MPE/iX GETLOG command. If you specify the AUTO option with the GETLOG command, the log file name must end with 001 to designate the first file in the log file set.

NOTE If you are using roll-back recovery, the log file and the database must be in the same volume set. To accomplish this, use the *device* parameter of the MPE/iX BUILD command and then create the log file in the appropriate group and account.

---

numrec is the maximum number of logical records. Maximum value allowed is 2,147,483,647. Default is 1023.

numextents is the maximum number of disk extents (from 1 to 32, inclusive). Default is 8.

initialloc is the number of extents to be initially allocated to the file at the time it is opened. Default is 1.

device is the class of the device on which the log file and database are to reside. This parameter puts the log file and database in the same volume set. *Required for roll-back recovery; is not used for roll-forward recovery.*

If the default NOAUTO option is used in the GETLOG command, disk log files must be of sufficient size to prevent the end-of-file from being reached, because MPE/iX causes the associated log process to terminate when the log file is filled to capacity. Therefore, subsequent calls to TurboIMAGE/XL intrinsics that require log records to be written to the log file will fail. If this occurs in the middle of a transaction, the database is left in an inconsistent state. It then becomes necessary to recover transactions with roll-forward or roll-back recovery. Because reaching the end of a disk log file is similar in effect to a system failure, disk log files should be built with a total capacity far exceeding their required size and consisting of as many extents of contiguous disk sectors as needed to meet the capacity requirements of the file, subject to the constraints of the MPE/iX file system; of these extents only enough to satisfy the expected capacity should be allocated initially.

NOTE When the NOAUTO option is used, the MPE/iX SHOWLOGSTATUS command can be used to determine the space usage of the existing log file and when to create a new log file using an MPE/iX CHANGELOG command.

Example

```
:BUILD ORDER001;CODE=LOG;DISC=200000,20,7
```

Step 5--Creating the Log Identifier

You create the log identifier on MPE/iX by using the GETLOG command shown in this section. A **log identifier** (*logid*) is a unique logical name that identifies a system logging process to which log records are passed. Before using the GETLOG command, use the MPE/iX LISTLOG command to check if the *logid* already exists on MPE/iX. The LISTLOG command lists all logging identifiers, including creator names, log file names, and whether or not the AUTO option is used.

If the *logid* exists and was created by someone else, you must specify a different *logid*. If the *logid* exists and was created by you, check the parameters to verify that they are the ones you want. If you find, for example, that you used the default NOAUTO option and you want to change it to AUTO, you can either change it with the MPE/iX ALTLOG command, or you can release the *logid* with the MPE/iX RELLOG command and then re-create it with the GETLOG command. You can release a *logid* only if you are the creator of that *logid*.

The *logid* has a maximum of eight characters. Other users can access the log file and records in the same log file by using the *logid* you acquire and its password. To access the logging system directly through MPE/iX, you must have logging (LG) or operator (OP) capability and supply the identifier and password on the OPENLOG intrinsic.

If you use logging and create a backup copy using TurboSTORE/iX 7x24 True-Online Backup (with ONLINE=START or ONLINE=END), you can use either the default NOAUTO option or use the AUTO option. At database backup time, using TurboSTORE/iX 7X24 True-Online Backup (with ONLINE=START or ONLINE=END), the necessary information (such as the DBSTORE flag, name of the log file in use, the log record number, date, and time stamps) is recorded in the database root file before backup. At roll-forward recovery time using DBRECOV, this information is used to determine the log file to be used and the position in the log file from where roll-forward recovery must begin. With the NOAUTO option, there is only one log file. When using the AUTO option, the log file next in sequence in a round-robin fashion is automatically created. Hence there can be more than one log file needed for roll-forward recovery. In this case, it is very important to maintain a process of synchronizing the log files with the backup copy. It is required that the AUTO/NOAUTO option be not changed after backup and while the log process is active. You can change it only when starting a new log cycle. Refer to step 8 later in this chapter.

Syntax

:GETLOG *logid*;LOG=*logfile*, {DISC/TAPE} [;PASS=*password*] [;{AUTO/NOAUTO}]

Parameters

logid is the logging identifier to be established on the system. It consists of a string of up to eight characters that is meaningful to the user application.

NOTE Do not give the *logid* and the log file the same name, because the *logid* is used to name the RESTART file (used for DBRECOV STOP/RESTART recovery explained later in this chapter).

logfile is an MPE/iX 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 (001-999). The first log file specified with the MPE/iX GETLOG command must end with the last three digits equal to 001 if the AUTO option is used. (A warning message is issued if the log file does not end in 001.) When the AUTO option is used, the next log file will be opened automatically when the current one is full. The new log file is numbered consecutively. When the AUTO option is not used, the next log file needs to be built manually using the CHANGELOG command, when the current log file is almost full.

DISC is the class of the device on which the log file is to reside. For roll-back recovery, the log file and the database must be in the same volume set. If the log file specified for the *logid* is a serial file, the AUTO/NOAUTO option is ignored.

TAPE is the class of the device on which the log file is to reside.

password 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.

AUTO performs an automatic CHANGELOG command when the disk log file becomes full. A new log file is automatically created with the same log file name incremented by one in the digit portion; for example, if the current log file is ORDER001, the next file will be ORDER002. This enables logging to continue uninterrupted, also creating a sequence of log files or a **log file set**.

If you use TurboSTORE/iX 7x24 True-Online Backup (with ONLINE=START or ONLINE=END) to back up your database and plan on using DBRECOV for roll-forward recovery, you must properly synchronize the log files with the backup copy because the log files are automatically created in a round-robin fashion. The SHOW *database* ALL command of DBUTIL for the restored backup copy gives you the name of the logfile to begin roll-forward recovery.

NOAUTO is the default. No CHANGELOG command is performed when the disk log file becomes full.

Example

In the following example, the *logid* is created with the default NOAUTO option and then changed to specify the AUTO option. The log file name follows the naming convention required by the CHANGELOG command. The LISTLOG command is used to check if the *logid* exists. The ALTLOG command is used to change the AUTO/NOAUTO option.

```
:LISTLOG
NO LOGGING IDENTIFIERS ASSIGNED (CIWARN 1231)
:GETLOG ORDERLOG;LOG=ORDER001,DISC;PASS=PASSLOG
:LISTLOG

LOGID                CREATOR             CHANGE    AUTO    CURRENT LOG FILE

ORDERLOG             BEA.MKTG            YES       NO      ORDER001.MKTG.SYS

:ALTLOG ORDERLOG;AUTO
:LISTLOG

LOGID                CREATOR             CHANGE    AUTO    CURRENT LOG FILE

ORDERLOG             BEA.MKTG            YES       YES     ORDER001.MKTG.SYS
```

Because NOAUTO is assumed by default in the GETLOG command, the disk log file is closed when it becomes full and logging is shut down unless you manually issue a CHANGELOG command. When the NOAUTO default is used, you need to verify the capacity of the log file on a regular basis and issue a CHANGELOG command when necessary. When the AUTO option is specified as in the ALTLOG command above, logging automatically initiates a CHANGELOG command when the current log file becomes full.

In the example above, the AUTO option has been specified. Here is what happens when log file ORDER001 becomes full. Logging initiates a CHANGELOG command causing the log records to be written to ORDER002, the next log file in the sequence. As each log file becomes full, logging initiates additional CHANGELOG commands creating log files automatically until log file ORDER999 becomes full. At this point, the log file name is reset to ORDER001 and logging continues automatically.

If you use TurboSTORE/iX 7x24 True-Online Backup (with ONLINE=START or ONLINE=END) to backup your database and DBRECOV to perform roll-forward recovery, you must take precautions to synchronize the proper log files with the stored database. Refer to step 8 for more information.

If you specified the AUTO option as in the preceding example, and you need to restart logging at *logfile*001, you can issue an ALTLOG command as shown in the following example:

```
:LISTLOG

LOGID                CREATOR             CHANGE    AUTO    CURRENT LOG FILE

ORDERLOG             BEA.MKTG            YES       YES     ORDER026.MKTG.SYS

:ALTLOG ORDERLOG;LOG=ORDER001,DISC
```

Step 6--Setting the Log Identifier

The two previous steps were executed using MPE/iX commands. At this point, you must notify the MPE/iX user logging system of the TurboIMAGE/XL logging intention by setting the log identifier and *logid* password into the database root file, using the DBUTIL >>SET command, as shown in the example below:

```
:RUN DBUTIL.PUB.SYS
>>SET ORDERS LOGID = ORDERLOG
  PASSWORD:? PASSLOG                    If no logid password was previously specified
                                        in the GETLOG command, you would press Return
                                        at the prompt.
```

DBUTIL checks the validity of the *logid* with MPE/iX, and reports a warning as follows if the log identifier is not valid or if its password is incorrect:

```
WARNING:  non-existent LOGID
```

Once the log identifier has been set into the database, the log identifier parameters cannot be altered for the logging and recovery system to function correctly.

Step 7--Setting Flags for the Database Backup Copy

In addition to setting the log identifier, certain flags need to be set for the database before creating the backup. Some flags such as LOGGING, ROLLBACK, and MUSTRECOVER need to be set only once and can remain enabled unless you change your recovery plan. However, the database access flag is disabled before the backup and enabled (access allowed) after the backup. On the contrary, the recovery flag is enabled before the backup and disabled after the database is recovered. This is done before someone starts accessing the database to facilitate the recovery of the database after it is restored from the backup copy. If you plan on using TurboSTORE/iX 7x24 True-Online Backup (with ONLINE=START or ONLINE=END) option to back up the database, when it is open for access, you will not be able to disable the database access flag and enable the recovery flag. If recovery becomes necessary, you can set these two flags immediately after restoring from the backup and before starting the recovery process. Use DBUTIL to set the following flags in the root file *before making the backup copy of the database*:

- **Enable Logging Flag.** This flag ensures that all database modifications are logged and available for later use by the recovery system, if necessary. When you enable the database for logging, DBUTIL checks whether a database backup copy has been stored with DBSTORE or TurboSTORE/iX 7x24 True-Online Backup (with ONLINE=START or ONLINE=END). If not, DBUTIL issues a warning message. Because the database is stored after logging has been first enabled, the DBUTIL warning message can be interpreted as a prompt to store the database. If you plan to use roll-forward recovery, use the following command to enable logging (implies roll-forward logging):

  ```
  >>ENABLE database name FOR LOGGING
  WARNING:  database modified and not DBSTORED
  ```

  If you plan to use roll-back recovery, enabling the roll-back flag will automatically enable the logging flag.

  Remember that, if logging to disk, the log identifier must be created before the preceding command can execute successfully (refer to "Step 5--Creating the Log Identifier").

- **Enable Rollback Flag.** If you plan to use roll-back recovery, use the following command to enable roll-back recovery:

```
>>ENABLE database name FOR ROLLBACK
```

  After the *logid* is set and the log file has been built, the >>ENABLE command for roll-back recovery shown above automatically enables logging for the database. However, the >>DISABLE *database name* FOR ROLLBACK command will not automatically disable logging.

- **Enable MUSTRECOVER Flag.** When the MUSTRECOVER flag is enabled, only readers can access the database after a system failure until the database is recovered. This prevents modifications to a potentially inconsistent database.

```
>>ENABLE database name FOR MUSTRECOVER
```

  After the *logid* is set and the log file has been built, logging is automatically enabled when the ENABLE command is used to enable MUSTRECOVER. However, the >>DISABLE *database name* FOR MUSTRECOVER command will not automatically disable logging.

- **Disable Access Flag.** By disabling the database for user access, you ensure that modifications cannot be made to the database after restoring it. Any attempt to open the database with a call to DBOPEN returns an error message. Access to the database should be disabled before storing the backup copy, so that in the event of a system failure the database is restored with access disabled. This prevents users from opening the database and making modifications before recovery is executed. Disabling access to the database is also useful as a general security measure to prevent database access at unauthorized times. (Read the note below.) The DBUTIL command for disabling access is shown below:

```
>>DISABLE database name FOR ACCESS
```

- **Enable Recovery Flag.** Enabling the database for recovery allows the TurboIMAGE/XL recovery system to access the database. The database is stored with recovery enabled so that when it is restored, it is ready for recovery. (Read the note below.) The DBUTIL command for enabling roll-forward recovery is shown below:

```
>>ENABLE database name FOR RECOVERY
```

NOTE If you plan on using TurboSTORE/iX 7x24 True-Online Backup (with ONLINE=START or ONLINE=END) when the database is open for access, you will not be able to set the DISABLE FOR ACCESS or ENABLE FOR RECOVERY flags for the backup copy. In the event recovery becomes necessary, set these flags immediately after restoring the database from backup.

The database can be stored with DBSTORE or TurboSTORE/iX 7x24 True-Online Backup

(with ONLINE=START or ONLINE=END)—after the preceding flags have been set in the database. Logging status can be checked by referring to the procedure in "Logging Status" later in this chapter.

Step 8--Making a Database Backup Copy

This step is required for roll-forward recovery. It is recommended (for protection in the event of a hard disk failure), but not required, for roll-back recovery. There are two ways to make a database backup copy:

- use DBSTORE
- use TurboSTORE/iX 7X24 True-Online Backup with ONLINE=START or ONLINE=END option

Using DBSTORE

This explains the backup process using DBSTORE. See the section after this for information on using TurboSTORE/iX 7X24 True-Online Backup.

Make a database backup copy using DBSTORE to store a copy of the database with flags (access disabled, recovery enabled, logging enabled) set as specified in step 7. Because the correspondence between log files and database backup copies is crucial, DBSTORE sets a DBSTORE flag in the database root file, along with a time stamp designating the date and time of the DBSTORE operation, before storing the database. DBSTORE can only store one database at a time and requires that the database be closed during backup. Also, it does not store all external files, such as third-party index files, along with the database.

NOTE The DBSTORE flag is cleared by the first modification to the database (DBPUT, DBDELETE, or DBUPDATE) indicating that the database no longer corresponds to the stored copy.

Before logging is enabled, DBUTIL checks the DBSTORE flag to ensure that the working database is the same as the database backup copy. For example, suppose a database is stored and some modifications to the database are made before logging is enabled. If you then try to enable logging, DBUTIL, determining that the DBSTORE flag has been cleared, prints a message indicating that the present state of the (modified) database 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 database. 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 DATABASE?  ORDERS
DATABASE STORED
END OF PROGRAM
```

NOTE If you plan to restore the TurboIMAGE/XL database on an MPE V system, use the TRANSPORT option of the DBSTORE or STORE command. Chapter 8 contains detailed information about the DBSTORE command. Refer to the *MPE/iX Commands Reference Manual* for more information about STORE options.

TurboSTORE/iX 7x24 True-Online Backup

This section describes the use of TurboSTORE/iX 7x24 True-Online Backup with the ONLINE=START or ONLINE=END option.

When multiple databases and related files are involved, you may use the MPE/iX STORE command (with or without the ONLINE=START or ONLINE=END option) to collectively copy them to tape or other storage media and, if necessary, collectively restore them by using the MPE/iX RESTORE command. You need to have either SM or OP capability to do this and should have an understanding of the standard rules and features associated with STORE and RESTORE. Note that when using the STORE command without the ONLINE=START or ONLINE=END option, neither the DBSTORE flag nor the time stamp (signifying the date and time the backup copy was made) is set in the database root file.

When you use MPE/iX STORE command (with the option ONLINE=START or ONLINE=END), you are using TurboSTORE/iX 7x24 True-Online Backup which does the following:

1. Backs up multiple databases along with their related files including the TC file, third-party index files, jumbo data set files, and B-Tree index files.
2. Sets DBSTORE flag in the root file(s)
3. Sets the time stamp designating the date and time of the backup operation before storing the database.
4. Quiesces the database if it is open for write access. The database is quiesced at a point when no active transaction is in progress.
5. Posts log records if enabled for logging, designating the beginning and end of quiesce. DBRECOV uses these records to determine the starting point of the roll-forward recovery.

The most significant advantage is that your database can even be open for access when you start the backup process and can continue to remain open for access. The backup becomes a True-Online Backup.

When performing a True-Online backup, there is a point in time when the backup occurred, especially when the database is open for modification. This is helpful when performing a roll-forward recovery to ensure that all files are in a logically consistent state at the same time for backup and later a subsequent restore for recovery. This point is called the *sync point*, where all data set files are synchronized. Also, it means this is the point when the TurboIMAGE/XL database is quiesced for a short duration, that is, it is in a logically consistent state and there is no active transaction in progress. All ongoing transactions, if any, are allowed to be completed before the *sync point*.

The ONLINE=START option allows the *sync point* at the beginning of a True-Online

backup, and the ONLINE=END option permits the *sync point* at the end of a True-Online backup. The ONLINE=START option has the following advantages over ONLINE=END:

- Allows the database to be restored on an earlier version of MPE/iX.
- Allows faster partial (selective, not @.@.@) restores, since sync point at end requires RESTORE to read the log files (used for backup) from the end of the last piece of media.
- Spreads log data throughout the backup media, and hence, is less vulnerable to media errors.

NOTE All True-Online backups created with the sync point at the beginning (ONLINE=START) can be restored on any MPE/iX system. However, backups created with the sync point at the end (ONLINE=END) can only be restored on MPE/iX Release 5.5 or later. If you know at the time of performing the backup that the database(s) must be restored onto an earlier system, create the backup with the sync point at the beginning. This is independent of using user logging and DBRECOV.

To perform roll-forward recovery of your database stored using True-Online backup, the following requirements must be adhered to:

- Your system must be on MPE/iX Release 5.5 or later.
- If you used the AUTO option with the GETLOG command, ensure that the log files following the backup are properly synchronized with the backup, especially in a situation when the last log file number, 999, switches to 001. It is recommended that you remove (STORE and PURGE) all log files preceding (but not including) this backup. The SHOW *database* ALL command gives the name of the log file from which the recovery will begin.
- LOG process must remain active when storing the database even when the database is not open. That is, LOG *logid*,STOP must not be issued before storing the database. This is because the logging information which is dynamic in nature, is incorporated in the root file when the database is stored. This dynamic logging information can only be obtained when the log process is active. It is used later when performing roll-forward recovery.
- One of the new options, ONLINE=START or ONLINE=END of TurboSTORE/iX, must be employed when performing the backup.

```
:FILE musicbk;dev=tape
:STORE music; musicbk;ONLINE=START
```

- After the backup is completed, purge all log files preceding the one in use when the backup was initiated. It is essential that you retain the one used when backup was initiated and the ones following that. These are the log files that will be needed later to perform roll-forward recovery.

  To find out the log file used by True-Online backup, employ the SHOW *database* ALL command of DBUTIL, which displays the time, date, and the log file name used for the

True-online backup. An example is given below:

```
Access is enabled.
.
.
.
Logging is enabled.
.
.
.
Database last stored using True-Online Backup and
   log file NLOG007 on THU, JAN 18, 1996,  6:06 PM.
```

If you used the AUTO option with GETLOG command, you may purge log files NLOG001 up to NLOG006. When you continue updating the database, log files up to NLOG999 will be automatically created when needed. After that, the log file NLOG001 is automatically created and used as is done today. If the log file, next in sequence to be automatically created, already exists, an error will be generated. As the log files are automatically created in a round-robin fashion as done today for the AUTO option, you will have to take extra measures to ensure that the log files, logically related to your database for recovery, are properly maintained. One way is to always start a new log cycle beginning with the first log file, NLOG001, of the log file set. If you did not use the AUTO option, you do not need to purge any files.

Example

Following is an example of how to store your database using this method:

```
:BUILD DBUSASF;DISC=5000
:STORE DBUSA;*DBUSASF;ONLINE=START

>>  TURBO-STORE/RESTORE  VERSION  C.55.07  B5152AA  <<
               (C) 1986 HEWLETT-PACKARD CO.
STORE  dbusa;*dbusasf;ONLINE=START

SUN, MAR 23, 1997,  4:05 PM

ONLINE BACKUP UTILIZED DISC SPACE FOR LOG ON THE FOLLOWING VOLSETS:

MPEXL_SYSTEM_VOLUME_SET :                          0 KB

DATABASE INFORMATION:

TURBOIMAGE DATABASE: DBUSA   .RECTOL   .QALANG
FILES STORED :                                     4

FILES STORED :                                     4


TOTAL MEDIA WRITTEN :                              1
```

Benefits of TurboSTORE/iX 7x24 True-Online Backup

The benefits of using True-Online backup to store TurboIMAGE/XL databases regardless of logging enabled are as follows:

- Stores the database when it is open for `read/write access`, or `closed`.
- Stores one or more TurboIMAGE/XL databases, along with their related files including the TC file, third-party index files, jumbo data sets, and B-Tree index files, if any.
- Stores the database after setting `DBSTORE` flag, time stamp, and other necessary information in the root file.
- Store without the need to stop the user log process before backup, if logging is enabled. Also, you do not need to start a new log cycle after the backup. Therefore, the roll-forward recovery, if enabled for roll-forward, need not commence from the beginning of the first log file.

NOTE If your database is opened exclusively, such as with mode 3 or 7, True-Online backup does not store your database, and reports an error:

```
DATABASE NOT STORED: UNABLE TO STORE SOME DATABASE FILES
BASED ON THE SELECTION CRITERIA, NO FILES WERE STORED. (S/R
1713)
```

Unlike `DBSTORE, STORE` with ONLINE=START or ONLINE=END option allows your database(s) to be open for access when you perform the backup.

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. If TurboSTORE/iX 7x24 True-Online Backup (with ONLINE=START or ONLINE=END option) is used to back up the database, it can also display the date, time, and the logfile used during backup. The following example illustrates roll-forward recovery and the commands used to set the *logid* and flags into the database, as presented in this chapter. If the steps discussed earlier in this chapter have been followed, the database can be stored. The password (denoted by *******) does not appear on the screen.

```
:RUN DBUTIL.PUB.SYS
     .
     .
     .
>>SET ORDERS LOGID=ORDERLOG
  PASSWORD *******

LOGID: ORDERLOG IS VALID
       PASSWORD IS CORRECT

>>DISABLE ORDERS FOR ACCESS
  Access is disabled

>>ENABLE ORDERS FOR RECOVERY, LOGGING, MUSTRECOVER
  WARNING: Database modified and not DBSTOREd
  Recovery is enabled
  Logging is enabled
  Mustrecover is enabled
>>SHOW ORDERS ALL
  For database 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.
  Mustrecover is enabled.
  Logging is enabled.
  Prefetch is disabled.
  Indexing is disabled.
  HWMPUT is disabled.
  Restart is disabled.
  Database last stored using True-Online Backup and
     Thu, JAN 18, 1996, 3:39 PM
  Database has been modified since last store date.
  Shadowing is disabled.
  Subsystem access is READ/WRITE.

  CIUPDATE is allowed.
  Dynamic capacity expansion is not used.
```

```
Database has at least one indexed dataset.
BTREEMODE1 is off, wildcard = "@".

Logid:   ORDERLOG is valid.
         password is correct.

XM log set      :  default XM user log set
                   for volume set MPEXL_SYSTEM_VOLUME_SET
XM log set type :  circular
XM log set size :  32 megabytes

The language is  0 - NATIVE-3000.

Buffer specifications:
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)

No other users are accessing the database.
```

NOTE The displayed buffer specifications are the default TurboIMAGE/V values. These specifications or any new ones that you set are displayed for MPE V compatibility, but they are not used by TurboIMAGE/XL which uses a large default value. See chapter 8 or appendix H for more information.

Logging Maintenance

You should determine a log maintenance cycle for the database. For example, suppose the database 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 the flags are set (see the following discussion). At the end of the day, the log process is stopped from the console and the flags are reset for storage of the database backup copy. If you use TurboSTORE/iX 7x24 True-Online Backup (with ONLINE=START or ONLINE=END option) to back up the database, the log process must remain active; you will not need to start and stop the log process every day.

The duration of this maintenance cycle depends on at least two considerations:

- The amount of time needed to store the database periodically, and
- The amount of time required to recover the database from the log file using DBRECOV if the system fails.

The more often the database backup copy is stored, the smaller the log file and the shorter recovery time will be. Regular backup of the database is recommended, though not required when using roll-back recovery. Refer to appendix G for a brief overview of the disadvantages and benefits of logging to disk and logging to tape. This appendix also includes sample job streams for the logging cycle.

Starting the Logging Process

After a database backup copy has been stored as described earlier in "Logging Preparation," a logging process must be allocated to the log identifier so that it can be activated. A **log process** is an MPE/iX 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 disk before writing them to the log file. This process is initiated from the console using the MPE/iX LOG command. To issue the LOG command, you must have logging (LG) or operator (OP) capability. The ALLOW command can be used to transfer permission to enter this console command.

Syntax

```
:LOG logid,{ START
           RESTART
           STOP    }
```

Parameters

logid	is the name of the *logid* to be activated; the *logid* must have been set previously into the database root file.
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
```

If the log process is stopped using the LOG command, but a database backup copy is not generated at that time, the RESTART option must be used to resume logging to the same log file.

To determine whether or not a log process is running, use the MPE/iX SHOWLOGSTATUS command to determine the log identifiers of active log processes. SHOWLOGSTATUS displays the percentage of records in the log file if the *logid* output is to disk. This information may be helpful in determining when to perform a CHANGELOG command (see "CHANGELOG Capability" in this section). The following sample listing was produced by a SHOWLOGSTATUS command:

```
:SHOWLOGSTATUS

LOGID      CHANGE    AUTO    USERS    STATE     CUR-RECS   MAX-REC   %USED    CUR-SET

MYLOG        NO       NO       4     INACTIVE     100       1000     10%        1
TAPELOG      YES               1     INACTIVE    5738                           1
ORDERLOG     YES      YES      2     INACTIVE     500       1000     50%        2
```

Re-enabling Logging

In the event that logging for roll-forward recovery is disabled and needs to be re-enabled, the database should be stored with DBSTORE or TurboSTORE/iX 7x24 True-Online Backup (with ONLINE=START or ONLINE=END option) before logging is re-enabled. This ensures that the DBSTORE flag and time stamp set when logging was first enabled are not reset when logging is re-enabled.

Setting Database Flags

After logging is initiated, you can allow users to modify the database by running DBUTIL and enabling the database for access. You should also disable recovery at this time. This provides a safeguard against unintended recovery if DBRECOV is executed from a stream file against several databases simultaneously. For example:

```
:RUN DBUTIL.PUB.SYS
     .
     .
     .
>>ENABLE ORDERS FOR ACCESS
  Access is Enabled
>>DISABLE ORDERS FOR RECOVERY
  Recovery is Disabled
```

CHANGELOG Capability

The MPE/iX CHANGELOG feature provides a continuous MPE/iX user logging process, with the ability to change tape or disk log files when they reach capacity without stopping the user logging process. User logging also keeps 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 the 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 allows 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 CHANGELOG is not the creator of the *logid*, either LG or OP capability is required. If the mirror database method (DBRECOV STOP/RESTART) is being used, CHANGELOG makes logging without interruption on the primary system possible.

NOTE If the database needs to be recovered when you are using the MPE/iX CHANGELOG feature, the DBRECOV recovery facility may start at the first log file in the logging cycle (for example, LOGF001), or at a different point (for example, LOGF013) if you used TurboSTORE/iX 7x24 True-Online Backup. Keep this in mind when determining the length of your logging cycle. See "Recovering the Database" later in this section for a sample scenario.

Syntax

```
:CHANGELOG logid [;DEV=device]
```

Parameters

logid is the name of the currently active logging process.

device is the class of the device of the new log file (DISC, TAPE). If the device class specified is DISC, the file is created in the *logid* creator's 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 log file is changed using ALTLOG, no linkage of the log file set is provided. A CHANGELOG command can only be used on a *logid* set up with the GETLOG command. The CHANGELOG command terminates if the logging process state is INACTIVE, INITIALIZING, or RECOVERING. The command will also terminate if a CHANGELOG command is already pending. The following message is 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 $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 on tape, the following message appears on the console requesting the mounting of a new log file (in this case the *logid* is ORDERLOG):

```
Mount new tape for logid ORDERLOG
```

If a LISTLOG command is executed while the logging process is performing a CHANGELOG command, the file name displayed is that of the current log file. The log file name is not 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 a LISTLOG display. A CHANGELOG is currently taking place on log file ORDER001; and, because the CHANGELOG to ORDER002 is not yet successfully completed, ORDER001 is displayed:

```
:LISTLOG

LOGID           CREATOR       CHANGE    AUTO    CURRENT LOG FILE

MYLOG           DATA.SYS        NO       NO     MY.PUB.SYS
TAPELOG         DATA.SYS        YES              TAPE001
ORDERLOG        TST.MKTG        YES      YES     ORDER001.MKTG.SYS
```

Recovering the Database

When a database needs to be recovered, DBRECOV starts with the first log file written to in a given logging cycle, or it may start with the log file in use when backup was done using TurboSTORE/iX 7x24 True-Online Backup (with ONLINE=START or ONLINE=END option). If you used True-Online Backup, the SHOW *database* ALL command of DBUTIL displays the name of the logfile for starting recovery.

To illustrate this concept, consider the following scenario when logging cycle began with a logfile LOGF001 and DBSTORE was used for backup.

The MPE/iX CHANGELOG command or the GETLOG AUTO feature is being used to log transactions to a log file called LOGF001. When LOGF001 fills up, user logging automatically appends LOGF001 with a CHANGELOG record indicating that TurboIMAGE/XL is now logging to file LOGF002. This process can continue until either LOGF999 is reached or another file is encountered that has the same name as the one being created by user logging.

Now, assume that the database needs to be recovered and the logging cycle is currently writing to LOGF020. Note that DBRECOV will begin recovery at LOGF001, even if you know or want to recover only the transactions from, for example, log file LOGF013.

To illustrate another example when TurboSTORE/iX 7x24 True-Online Backup (with ONLINE=START or ONLINE=END option) was used to back up the database, assume that the log file in use during backup was LOGF041. If a database needs to be recovered and the logging cycle is currently writing to LOGF065, DBRECOV will begin recovery with LOGF041 (following the backup) and will continue until LOGF065.

The resulting impact on recovery time is why serious consideration should be given to the logging cycle. The shorter the logging cycle is, the shorter the recovery time.

Ending the Logging Maintenance Cycle

At the end of the specified maintenance cycle (for example, the end of day) do the following:

1. If you plan on using TurboSTORE/iX 7x24 True-Online Backup (with ONLINE=START or ONLINE=END option), omit this step. Otherwise, stop the logging process at the console with the LOG command.
2. If you plan on using TurboSTORE/iX 7x24 True-Online Backup (with ONLINE=START or ONLINE=END option) when the database is *open for access*, you must omit this step. In other cases, including when using TurboSTORE/iX 7x24 True-Online Backup (with ONLINE=START or ONLINE=END option) when the *database is closed*, run DBUTIL and do the following tasks:
   - Disable access to the database.
   - Enable recovery of the database.
3. Store a database backup copy (required for roll-forward recovery) with DBSTORE or TurboSTORE/iX 7x24 True-Online Backup (with ONLINE=START or ONLINE=END option).

Example

Example when using DBSTORE:

```
: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
```

Example when using TurboSTORE/iX 7x24 True-Online Backup (with ONLINE=START or ONLINE=END option) and database is open for access:

```
:file orderssf;dev=disc
:store orders;*orderssf;online=start
 >>TURBO-STORE/RESTORE  VERSION  C.55.07  B5152AA <<
                   (C) 1986 HEWLETT-PACKARD CO.

STORE  orders;*orderssf;ONLINE=START

FRI, APR 18, 1997, 12:01 PM


ONLINE BACKUP UTILIZED DISC SPACE FOR LOG ON THE FOLLOWING VOLSETS:

MPEXL_SYSTEM_VOLUME_SET :                              0 KB


DATABASE INFORMATION:

TURBOIMAGE DATABASE: ORDERS  .RECTOL  .QALANG
FILES STORED :                                         7

FILES STORED :                                         7


TOTAL MEDIA WRITTEN :                                  1
```

Logging Results

All database modifications (DBPUTs, DBUPDATEs, and DBDELETEs) are logged; and in modes 1 through 4 calls to DBOPEN, DBCLOSE, DBBEGIN, DBXBEGIN, DBEND, DBXEND, and DBMEMO are logged to the log file. Each DBBEGIN, DBXBEGIN, DBEND, and DBXEND causes a **log record** to be written to the log file that includes such information as time, date, and user buffer. These log records are used by DBRECOV to identify logical transactions.

When using TurboSTORE/iX 7x24 True-Online Backup (with ONLINE=START or ONLINE=END option), log records designating the beginning and end of a quiesce point triggered by True-Online backup are also written to the current log file. DBRECOV uses these log records to determine the starting point of recovery of the database. Transactions following the quiesce point will be used to recover the database.

Log Records

DBOPEN log records contain a time stamp recorded in the database root file, indicating the date and the time of the last backup using DBSTORE or TurboSTORE/iX 7x24 True-Online Backup (with ONLINE=START or ONLINE=END option) (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, database, 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

The two log file time stamps are as follows:

- The DBSTORE time stamp set at the time the last database backup copy was made using DBSTORE or TurboSTORE/iX 7x24 True-Online Backup (with ONLINE=START or ONLINE=END option). This time stamp is used by roll-forward recovery. The DBSTORE time stamp is fixed and does not change once the database backup copy has been made.
- The roll-back time stamp created at the time the first DBOPEN is executed against the database. The roll-back time stamp is updated to the real time of the first DBOPEN following each close of the database, providing a roll-back termination point should a roll-back recovery be required.

Roll-Forward Recovery

Roll-forward recovery can be executed to bring databases back to a likeness of their state at the time of a hard system failure (for example, a disk head crash or a system failure while the database is enabled for AUTODEFER). Roll-forward recovery requires a synchronized backup copy of the database and the log file. (Refer to "Logging Preparation" earlier in this chapter for roll-forward logging information.)

When executing roll-forward recovery following a system failure, the TurboIMAGE/XL utility DBRECOV recovers the database physical and logical integrity by re-applying (to a backup copy of the database) all the completed transactions that were written to the log file. It does not re-apply incomplete transactions.

Recovery of the database requires restoring the backup copy and running the recovery system to re-execute the database modifications from the log file. In addition, the >FILE command of the DBRECOV utility can be used to create individual user recovery files and to return information regarding the successful recovery or suppression of transactions. The information from these files lets each user know where to resume transactions within the database following recovery. Refer to the discussion of the DBRECOV >FILE command in chapter 8.

Although the logging and recovery system is designed to successfully re-execute transactions that completed before the system failure, some transactions may 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.
- A transaction may depend upon some database modification that was suppressed. This condition indicates inadequate locking between processes.
- An incorrect version of the database was restored. Recovery will yield invalid and erroneous results 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 chapter.

CAUTION In the event of a system failure, do not restart logging before running DBRECOV. 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 database. The database should be disabled for user access until recovery has completed. To prevent access to the database after a system failure without recovery, enable the MUSTRECOVER feature when you enable 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 disk) as shown in steps 2 through 6 of "Logging Preparation" earlier in this chapter.
2. For each particular database, disable access and enable the logging and roll-forward features by entering the following DBUTIL commands. You can also enable the MUSTRECOVER option if you wish to use it.

   ```
   >>DISABLE database name FOR ACCESS
   >>ENABLE database name FOR LOGGING,RECOVERY
   >>ENABLE database name FOR MUSTRECOVER
   ```

   Enable any other appropriate flags as discussed in step 7 under "Logging Preparation."

   If you plan on using TurboSTORE/iX 7x24 True-Online Backup, your database must be closed so that you can set these flags to enable the roll-forward feature. For subsequent true-online backups, you have an option to set the ACCESS and RECOVERY flags. Remember that before you start the roll-forward recovery, the RECOVER flag must be enabled and ACCESS flag be disabled.
3. Make a backup copy of the database as discussed in step 8 under "Logging Preparation."
4. Start the logging process and enable user access to the appropriate databases as shown in "Logging Maintenance," earlier in this chapter.

Restoring the Database Backup Copy

After a hard system failure, and before roll-forward recovery can begin, you must restore the database to the state it was in when backup was done. This is done by running the DBRESTOR program (if DBSTORE was used for backup, as shown below) or by using the MPE/iX RESTORE facility after purging the damaged database. All databases and files must be restored to their original group and account, and you must have privileged mode (PM) capability. Ensure that recovery is enabled and access disabled to prevent user modifications before the recovery system executes.

If you restored a database backed up with TurboSTORE/iX 7x24 True-Online Backup (with ONLINE=START or ONLINE=END option), you may need to enable recovery and disable access. Refer to the DBUTIL >>SHOW command and the following example. If the flags were set as recommended prior to making the backup copy, no changes are needed.

```
:RUN DBUTIL.PUB.SYS

        .
        .
        .
>>PURGE ORDERS                                  Note: this is damaged database.
  Database ORDERS has been PURGED
>>EXIT
```

```
END OF PROGRAM

:RUN DBRESTOR.PUB.SYS
WHICH DATABASE?  ORDERS
DATABASE RESTORED
END OF PROGRAM

:RUN DBUTIL.PUB.SYS
>>SHOW ORDERS FLAGS
  For database ORDERS

  Access is disabled.
  Autodefer is disabled.
  Dumping is disabled.
  Rollback recovery is disabled.
  Recovery is enabled.
  ILR is disabled.
  Mustrecover is enabled.
  Logging is enabled.
  Prefetch is disabled.
  Indexing is disabled.
  HWMPUT is disabled.
  Restart is disabled.
>>EXIT
END OF PROGRAM
```

In the above example, recovery is enabled and access is disabled. You do not need to change flags.

If you used TurboSTORE/iX 7x24 True-Online Backup (with ONLINE=START or ONLINE=END option), you would use RESTORE command as follows:

```
:restore *orderssf;orders;show
   >> TURBO-STORE/RESTORE  VERSION  C.55.07  B5152AA  <<
                  (C) 1986 HEWLETT-PACKARD CO.

RESTORE  *orderssf;orders;SHOW


FRI, APR 18, 1997, 12:03 PM


WILL RESTORE        7 FILES          ; NUMBER OF FILES ON MEDIA        7

FILENAME GROUP    ACCOUNT  VOLUME RESTRICTIONS            SECTORS CODE   MEDIA
ORDERS  .RECTOL  .QALANG   DISC                        :C      16 PRIV       1
ORDERS01.RECTOL  .QALANG   DISC                        :C      96 PRIV       1
ORDERS02.RECTOL  .QALANG   DISC                        :C      96 PRIV       1
ORDERS03.RECTOL  .QALANG   DISC                        :C    1584 PRIV       1
ORDERS04.RECTOL  .QALANG   DISC                        :C      80 PRIV       1
ORDERS05.RECTOL  .QALANG   DISC                        :C     128 PRIV       1
ORDERS06.RECTOL  .QALANG   DISC                        :C     160 PRIV       1

DATABASE INFORMATION:

TURBOIMAGE DATABASE: ORDERS  .RECTOL  .QALANG
```

```
FILES RESTORED :                                    7

FILES RESTORED :                                7

:

:RUN DBUTIL.PUB.SYS
HP30391C.07.00 TurboIMAGE/XL:  DBUTIL (C) COPYRIGHT HEWLETT-PACKARD COMPANY 1987

>>show orders flags
  For database ORDERS

  Maintenance word is not present.

  Access is enabled.
  Autodefer is disabled.
  Dumping is disabled.
  Rollback recovery is disabled.
  Recovery is disabled.
  ILR is disabled.
  Mustrecover is enabled.
  Logging is enabled.
  Prefetch is disabled.
  Indexing is disabled.
  HWMPUT is disabled.
  Restart is disabled.

>>exit
```

In the above example, you will need to disable the access flag and enable the recovery flag.

Several databases can log to the same log file simultaneously because each call to DBOPEN specifies the fully qualified name of the database. If all databases that logged to the same log file are to be recovered simultaneously, then the appropriate backup copy of each database must be restored prior to running the recovery system. However, if the recovery system begins execution before a database has been restored, accidental recovery is prevented if recovery has been disabled on the working database, as specified earlier in "Logging Maintenance."

The TurboIMAGE/XL logging and recovery systems depend upon the exact correspondence between the stored database backup copy and the working database on disk at the time logging was initiated. The DBSTORE flag and time stamp, properly used, enforce this condition. Therefore, it is recommended that you use DBSTORE or TurboSTORE/iX 7x24 True-Online Backup (with ONLINE=START or ONLINE=END option) to generate backup copies.

For flexibility, in the event that you might use the STORE command without ONLINE=START or ONLINE=END option to store the backup, the capability exists to defeat the time stamp and DBSTORE flag mechanism, by using the NOSTAMP and NOSTORE options of the >CONTROL command of DBRECOV. In this case, you must assume responsibility for maintaining the correspondence between the backup copy and the log file. Note that a database recovered with the wrong log file causes DBRECOV to generate erroneous data in the database and that this condition cannot always be detected. Modifications to the database before the database is recovered and with logging disabled also cause the recovered database to be incorrect.

Performing Roll-Forward Recovery

To complete the transaction roll-forward process following a hard system failure, perform the following steps:

1. Following a **start recovery** operation to bring up the system, locate the applicable log file medium to be used for roll-forward recovery.

   If logging to tape, the correct tape needs to be mounted. If using the MPE/iX CHANGELOG feature and there are multiple log file tapes, this will be the first tape in the series if DBSTORE was used. If TurboSTORE/iX 7x24 True-Online Backup (with ONLINE=START or ONLINE=END option) was used to backup the database and AUTO option is used, this may not be the first file in the series. If logging to disk, TurboIMAGE/XL automatically locates the first log file in the given logging cycle after checking for the *logid* in the root file. When TurboSTORE/iX 7x24 True-Online Backup (with ONLINE=START or ONLINE=END option) is used, necessary information such as the log file name in use, log file record numbers, date, and time of backup is recorded in the root file before backup. When the database is restored from this backup, the SHOW *database* ALL command of DBUTIL displays this information.

   Note that the system boot-up process writes a crash record to the last volume of the user log set. If a start recovery operation is not performed, this will not occur, causing DBRECOV to issue a warning.

2. Check your backup listings to see when your database was last stored. If time permits, store your damaged database with DBSTORE. Having a current backup of your damaged database is an extra measure to protect against lost data due to a damaged log file.

3. Restore the backup of the database; this backup was created at the beginning of the current logging maintenance cycle. Use the SHOW *database* command of DBUTIL to find out which log file will be used to start recovery. If the TurboSTORE/iX 7x24 True-Online Backup (with ONLINE=START or ONLINE=END option) was used for backup, the log file name is displayed. Otherwise, it will be the first log file in this logging cycle.

4. Enter the following MPE/iX command:

   ```
   :RUN DBRECOV.PUB.SYS
   ```

5. If you used the MPE/iX STORE command without ONLINE=START or ONLINE=END option to store the database when you started your logging cycle, enter the >CONTROL NOSTORE command of DBRECOV.

6. Enter the DBRECOV >RECOVER command below (where *database name* is the name of the individual database to be recovered):

   ```
   >RECOVER database name [,database name2,...,database nameN]
   ```

7. Enter all other desired DBRECOV commands (>FILE, >CONTROL with other options except NOSTORE shown above, and >PRINT.) Refer to chapter 8 for more information.

8. If you want to recover the database(s), enter the DBRECOV >RUN command. Otherwise,

enter the >EXIT command; DBRECOV terminates, and no recovery takes place.

```
>RUN
```

After the >RUN command is given, DBRECOV recovers the specified databases, creates user recovery files, and terminates. After entering the >RUN command, DBRECOV asks you to mount the log tape (if the log file medium is tape). Continue the roll-forward process as directed by messages returned to both the console and the terminal screens.

9. If the CHANGELOG command or the GETLOG AUTO option was used during logging and the logging file medium is tape or the next disk file in the log set is missing, the following message appears on the terminal screen and the console:

   ```
   Reply CONtinue on console when logfile is ready
   ```

   When the required log file is available, enter the response CON to the console request from DBRECOV.

Note that the response is in a form similar to REPLY for a tape mount used when storing or restoring tapes; that is, you need to supply the Process Identification Number (PIN). For recovery to succeed, you must have access to the log file. This implies either knowing the logging identifier password and having system manager (SM) or operator (OP) capability, or being the creator of the log identifier with read access to the log file if it resides in a different logon group and account. If the log file is on tape, the operator must reply with the proper volume identifier.

NOTE If the operator is unsure of the volume identifier, it is displayed on the console when the tape is mounted.

If the database creator and the creator of the log identifier are not the same, and if the disk log file and the database are in two separate accounts, follow the steps listed below for recovery to proceed:

1. Assign a maintenance word to the database.
2. Log on as the creator of the log identifier.
3. Fully qualify the database name when issuing the >RECOVER command.
4. Specify the maintenance word.

If you want to use TurboSTORE/iX 7x24 True-Online Backup (with ONLINE=START or ONLINE=END option) for roll-forward recovery, follow the steps below. For this illustration, it is assumed that you are starting with the first log cycle with the AUTO option, and recovery is needed after two backups.

1. Create a logid with the AUTO option.
2. Build a logfile, LOGF001.
3. Use DBUTIL to set the logid and enable the logging flag. You may optionally set the MUSTRECOVER flag as well.

4. Start the log process:

   ```
   :LOG logid,START
   ```

5. Back up the database using TurboSTORE/iX 7x24 True-Online Backup (with ONLINE=START or ONLINE=END option) and logfile LOGF001. Let this be **backup number 1**.
6. Access the database for modifications. The current log file is LOGF005.
7. Back up the database, while open for access, using TurboSTORE/iX 7x24 True-Online Backup (with ONLINE=START or ONLINE=END option). Let this be **backup number 2**.

   Note that LOGF005 was in use. At this point, you have an option to store log files LOGF001 to LOGF004 to another storage media for added protection. You can then purge them from the system if you want to continue the log process beyond logfile LOGF999.

8. Continue accessing the database until there is an interruption (for example, a system failure). The current log file is LOG009.
9. Reboot the system using the `START RECOVERY` option.
10. Optionally store the damaged database. Purge database using `DBUTIL`.
11. Restore the database from **backup number 2**.
12. Use `DBUTIL` to disable access and enable recovery. You may use `SHOW database ALL` to get backup information.
13. Use `DBRECOV` for roll-forward recovery. The root file contains information about the starting point for recovery. The recovery will start with the logfile LOGF005, following the backup (`DBQUIESCE` log records).
14. You can back up the recovered database using TurboSTORE/iX 7x24 True-Online Backup (with ONLINE=START or ONLINE=END option) to keep recovery time short. Remember the access flag is disabled and recovery is enabled. When you restore from this backup for roll-forward recovery, you will not need to execute step 12 above.
15. At this point, you can either restart the logging process (continue with LOGF009) or start a new log process with LOGF001. If you want to start a new log process, you will need to stop the log process and remove (optionally store and purge) the log files used so far, LOGF005 through LOGF009 (or LOGF001 through LOGF009, if you did not purge in step 7).

Note that if you plan on using TurboSTORE/iX 7x24 True-Online Backup (with ONLINE=START or ONLINE=END option) for an existing database and logging is already enabled, you will not need to execute steps 1 through 4.

Recovery from a Stream File

A stream file may specify all of the databases logging to one log file for roll-forward recovery. If one of the databases has not been restored at the time the stream file is run, recovery for that database is prevented because recovery for that database is disabled if

the recommended procedures have been followed. Recovery can be completed for all of the other specified databases that have been restored from a backup copy and the recovery flag is enabled, as long as >CONTROL ERRORS is set appropriately (see "DBRECOV >CONTROL" in chapter 8). This means that ERRORS must be increased by one for each database disabled for recovery, because an error message occurs each time a database specified in the >RECOVER command is not enabled for recovery.

MPE/iX Cleanup Mode and Roll-Forward Recovery

In the event of a system failure and subsequent start recovery operation, MPE/iX attempts to clean up any user log files that were open at the time of the failure. The cleanup procedure involves writing any records left in the system log file disk buffer to the user log file. When using roll-forward recovery without this cleanup, records left in memory will still be lost. You have the option to cancel (from the console) this cleanup procedure if the log file is on tape.

The advantage of the cleanup procedure is that fewer user log records written just prior to the failure are lost. For tape files, the disadvantage 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.

The TurboIMAGE/XL recovery program DBRECOV does not require the cleanup to be performed. If it is not performed, however, DBRECOV most likely will report a sequence or checksum error when the discrepancy caused by the failure is encountered. This would cause DBRECOV to assume the end-of-file has been reached.

DBRECOV Abort Message

If DBRECOV aborts before recovery completes, the following information is printed:

```
Abort occurred on database: dbname  dbgroup  dbaccount
Total database open count:  #open             Current open count: #open
Process user is: dbuser dbgroup dbaccount    Running program: progname
Log file name: logfilename                   Logging ID: logid
Log file record number: nnnnnnnn             Transaction type: xx
Transaction date/time from log record: day, month, dd, year, time
Last successful transaction #: nn
First log record # of last successful transaction: nn
```

Record the information, set the log file, locate the database store, and contact your HP support representative.

Roll-Back Recovery

Roll-back recovery is another TurboIMAGE/XL recovery option. Roll-back recovery provides rapid recovery of database data integrity following a "soft" system crash (that is, system failure or loss of working memory). The roll-back feature is invoked through the DBRECOV utility and requires only the current database log files to restore data integrity. Note that ILR is not needed for roll-back recovery.

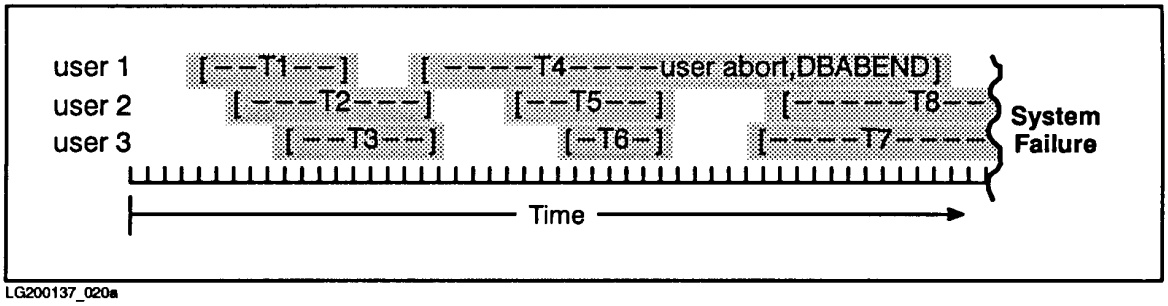
NOTE For roll-back recovery, the user log file and the database must be on the same volume set when logging to disk. When logging to tape, the database and the log file must be on the system volume set.

A database backup copy is not required for roll-back recovery. Regular backup of the database is recommended, however, and is always required for roll-forward recovery in the event of a more serious problem (for example, a disk head failure, or problems occurring while roll-back recovery is in progress).

With roll-back enabled prior to a 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. When invoked, the roll-back recovery feature will "roll back," or undo, any incomplete database transactions in the log file following a soft system crash.

NOTE A database must be enabled for roll-back before roll-back recovery can be performed.

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. Users two and three have each completed two transactions, and each has one incomplete transaction at the time of failure. Individual database transactions T1, T2, T3, T5, and T6 were completed and are properly reflected in the database 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 database. These incomplete transactions (T7 and T8) are then rolled back (undone) to their beginning, returning all affected data in the database to their original state before T7 and T8 began execution.

When transaction T4 is aborted, TurboIMAGE/XL 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 must be issued before issuing the DBRECOV >RUN command (refer to the discussion of DBRECOV in chapter 8):

```
>CONTROL NOABORTS
```

The above command string causes the aborted transaction to be treated as an incomplete transaction during roll-back recovery. *When >CONTROL NOABORTS is used, recovery must be performed at system startup time before anyone modifies the database.* Refer to "Record Numbers" under the DBRECOV >CONTROL discussion in chapter 8 for considerations when using the >CONTROL command.

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 disk) as shown in steps 2 through 6 of "Logging Preparation" earlier in this chapter.
2. For each particular database, disable access and enable the roll-back and recovery features by entering the following DBUTIL commands. You can also enable the MUSTRECOVER option if you wish to use it.

   ```
   >>DISABLE database name FOR ACCESS
   >>ENABLE database name FOR ROLLBACK,RECOVERY
   >>ENABLE database name FOR MUSTRECOVER
   ```

   Enable any other appropriate flags as discussed in step 7 under "Logging Preparation."

   If the *logid* was not set and/or the log file was not built before the >>ENABLE command is issued for roll-back recovery, a warning message that the *logid* or the log file is non-existent is displayed on the screen. Enable any other appropriate flags as discussed in step 7 under "Logging Preparation." Enabling for roll-back or MUSTRECOVER automatically enables the database for logging.
3. Make a backup copy of the database as discussed in step 8 under "Logging Preparation." You may use DBSTORE or TurboSTORE/iX 7x24 True-Online Backup (with ONLINE=START or ONLINE=END option) regardless of the database being open or closed with roll-back recovery.
4. Start the logging process and enable user access to the appropriate databases as shown in "Logging Maintenance," earlier in this chapter. After the *logid* has been set and the log file built, DBUTIL automatically enables logging when roll-back is enabled. Note that ILR is not needed for roll-back recovery; however, if ILR is enabled together with roll-back recovery, logging and ILR must be disabled separately.

When roll-back recovery is enabled, DBUTIL sets a roll-back flag in the root file to indicate that roll-back recovery is enabled for the database. 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 database is first opened. Roll-back recovery then uses the time stamp during recovery to verify the correct log file for each database.

Note that if the database and the log file are not on the same volume set, an error is issued.

CAUTION In the event of a system failure, do not restart user logging before running DBRECOV. Log records may have been lost due to the system failure. If user logging is resumed without a recovery, the resulting discontinuous log file would cause invalid results in the event of a subsequent recovery. To prevent access to the database after a system failure without recovery, enable the MUSTRECOVER feature when you enable the roll-back feature.

Disabling the Roll-Back Feature

To disable the roll-back feature, make sure that no users are accessing the database involved. Disable the roll-back feature by entering the following DBUTIL command string:

```
>>DISABLE database name FOR ROLLBACK
```

When roll-back is disabled, DBUTIL resets the roll-back flag and roll-back time stamp, but not the logging flag; therefore, logging is still enabled.

CAUTION DO NOT DISABLE ROLL-BACK IF ROLL-BACK RECOVERY MUST BE USED LATER. Disabling roll-back resets the logging time stamp. After the logging time stamp is reset, roll-back recovery cannot be performed with the current log file on the named database. The data in the database is considered correct and therefore cannot be rolled back.

When the >>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 continues to disable roll-back. If Y is not entered, the >>DISABLE command is not performed.

Performing Roll-Back Recovery

To complete the transaction roll-back process following a system failure, perform the following steps:

1. Following a **start recovery** operation to bring up the system, locate the applicable log file media to be used for transaction roll-back. If logging to tape, the correct tape needs to be mounted. If using the MPE/iX CHANGELOG feature and there are multiple log file tapes, this will be the first tape of the series. If logging to disk, TurboIMAGE/XL

automatically locates the first log file in the given logging cycle by checking the beginning of the root file for the *logid*.

Note that the system boot-up process writes a crash record to the last volume of the user log set. If a start recovery operation is not performed, this will not occur, causing DBRECOV to issue a warning.

2. Make a database backup copy in case a system failure occurs during the roll-back recovery process.
3. Enter the following MPE/iX command string:

```
:RUN DBRECOV.PUB.SYS
```

4. Enter the >CONTROL NOSTORE command of DBRECOV to allow recovery to proceed whether or not the DBSTORE flag is set.
5. Enter the following DBRECOV command (where *database name* is the name of individual databases to be rolled back):

```
>ROLLBACK database name [,database name2,....,database nameN]
```

6. Enter all other desired DBRECOV commands (>FILE, >CONTROL with other options except NOSTORE shown above, and >PRINT). Note that the >FILE command's optional parameter *rmode* is not used with the roll-back feature. Refer to chapter 8 for more information.
7. If you want to recover the databases, enter the DBRECOV >RUN command. Otherwise, enter the >EXIT command, and DBRECOV terminates, and no recovery takes place.

```
>RUN
```

After the >RUN command is given, DBRECOV asks you to mount the log tape (if the log file medium is tape). Continue the roll-back process as directed by messages returned to both the console and the terminal screen.

8. If the CHANGELOG command or GETLOG AUTO option was used during logging and the logging file medium is tape, the following message appears on the terminal screen and the console:

```
Reply CONtinue on console when logfile is ready
```

When the required log file is available, enter the response CON to the console request from DBRECOV.

Note that the response is in a form similar to REPLY for a tape mount used when storing or restoring tapes; that is, you need to enter the Process Identification Number (PIN). After the >RUN command is given, the DBRECOV program recovers the specified databases, 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, you must have access to the log file. This implies either knowing the maintenance word and having system manager (SM) or operator (OP) capability, or being the creator of the log identifier with read access to the log file if it resides in a different logon group and account. If the log file is on tape, you must know the volume identifier.

NOTE If the operator is unsure of the volume identifier, it is displayed on the console when the tape is mounted.

If the database creator and the creator of the log identifier are not the same, and if the disk log file and the database are in two separate accounts, follow the steps listed below for recovery to proceed:

1. Assign a maintenance word to the database.
2. Logon as the creator of the log identifier.
3. Fully qualify the database name when issuing the >ROLLBACK command.
4. Specify the maintenance word.

MPE/iX Cleanup Mode and Roll-Back Recovery

In the event of a system failure and subsequent start recovery operation, MPE/iX attempts to clean up any user log files that were open at the time of the failure. The cleanup procedure involves writing any records left in the system log file disk buffer to the user log file. Note that for roll-back recovery without this cleanup, transactions are not lost during a start recovery operation because they are not held in the memory buffer. You should not cancel (from the console) this cleanup procedure if the log file is on tape. Canceling the start recovery operation for tape log files will cause rollback recovery to fail.

The advantage of the cleanup procedure is that fewer user log records written just prior to the failure are lost. For tape files, the disadvantage 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.

The TurboIMAGE/XL recovery program DBRECOV does not require the cleanup to be performed. If it is not performed, however, DBRECOV most likely will report a sequence or checksum error when the discrepancy caused by the failure is encountered. This would cause DBRECOV to assume the end-of-file has been reached.

DBRECOV Abort Message

If DBRECOV aborts before recovery completes, the following information is printed:

```
Abort occurred on database: dbname  dbgroup  dbaccount
Total database open count:  #open          Current open count: #open
Process user is: dbuser dbgroup dbaccount     Running program: progname
Log file name: logfilename                    Logging ID: logid
Log file record number: nnnnnnnn            Transaction type: xx
Transaction date/time from log record: day, month, dd, year, time
Last successful transaction #: nn
First log record # of last successful transaction: nn
```

Record the information, set the log file, locate the database store, and contact your HP support representative.

DBRECOV Commands Used with Roll-Forward and Roll-Back Recovery

The following DBRECOV commands are used for either roll-forward or roll-back recovery:

CONTROL	Specifies the conditions for recovery.
FILE	Sorts the log file by individual users and/or user identifiers, and designates an MPE/iX file as the destination for each user's log records.
PRINT	Displays information before you actually initiate recovery with the >RUN command.
RECOVER	Designates the name of a database to be recovered using roll-forward recovery.
ROLLBACK	Designates the name of a database to be rolled back.
RUN	Initiates recovery of the specified database.

These commands are discussed below; refer to chapter 8 for more detailed information.

CONTROL Command

The >CONTROL command is used to specify the conditions for recovery. If the >CONTROL command is not issued, these four default conditions must be met for recovery to succeed:

- The database time stamp in the root file must correspond with the time stamp in each DBOPEN log file record.
- The DBSTORE flag must be set in the database root file.
- No errors are allowed in job (batch) execution.
- Transactions that 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, as follows:

Option	Default Option
NOMDBX	MDBX
NOSTAMP	STAMP
NOSTORE	STORE
NOABORTS	ABORTS
MODE4	MODEX
STATS	NOSTATS
ERRORS=nnnn	ERRORS=0 (job) or ERRORS=30,000 (session)
STOPTIME=dateX timeX	STOPTIME=dateY timeY
EOF=pppp	EOF=qqqq

The initial default condition for stop time and end-of-file is that none is imposed on recovery. When 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
```

Because STAMP was entered after NOSTAMP, STAMP negates NOSTAMP, so that recovery proceeds with the time-stamp check intact.

For the options and form of the >CONTROL command, refer to the discussion of the >CONTROL command of the DBRECOV utility in chapter 8. Note that the >CONTROL command does not specify a database. Therefore, all >CONTROL options (except NOSTORE which must be issued before recovery on a specified database is performed) apply to all databases 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/iX file as the destination for each user's log records.

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 a call to DBCLOSE (see Figure 7-1. earlier in this chapter). Within each transaction block, a transaction is defined as one of the following:

1. A single call to DBPUT, DBUPDATE, or DBDELETE if not preceded by a call to DBBEGIN (or DBXBEGIN if logging is enabled), or
2. A sequence of calls beginning with a call to DBBEGIN or DBXBEGIN, followed by any number of calls to DBPUT, DBUPDATE, or DBDELETE and ending with a call to DBEND or DBXEND respectively.

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 one of the following occurs:

1. All of the transactions within the block could not 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 database is open.

Consequently, an application can determine the outcome of recovery to some extent by examining the number of DBOPENs and DBCLOSEs or pairs of DBBEGIN and DBEND or DBXBEGIN and DBXEND 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, the possibility exists 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 intrinsics DBOPEN and DBCLOSE. *Rmode* refers to transactions that recovered successfully; *fmode* refers to transactions that failed to be recovered. Refer to the DBRECOV >FILE command 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 >PRINT command, the names of the databases specified for recovery by >RECOVER commands are returned. Note that no statistics are returned with the >PRINT DBTABLE command, because none exist before the log file is read. If you need these statistics without actually performing the recovery, use >CONTROL STATS to display this information. If FILETABLE is specified in the >PRINT command, file references, user references, *fmodes* and *rmodes* specified by >FILE commands are returned. These tables, along with statistics, are also printed when recovery is complete.

RECOVER Command

The >RECOVER command designates the name of a database to be recovered using roll-forward recovery. If more than one database has logged to the same log file, they can be recovered concurrently by entering the database names separated by a comma.

If the database copy was stored with a program other than DBSTORE or True-Online Backup (for example, MPE/iX STORE without ONLINE=START or ONLINE=END option), the DBSTORE flag will not have been set in the database root file.

If you are sure you have restored the correct, unmodified version of the database, and wish to use it for recovery, the >CONTROL NOSTORE option must be entered before the >RECOVER command can succeed (refer to the discussion of >CONTROL in chapter 8).

Other conditions necessary for the success of the >RECOVER command include:

- The database 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 "Step 6--Setting the Log Identifier," earlier in this chapter).
- The database must be enabled for recovery.
- All databases specified for recovery must contain the same log identifier.
- You must be the database creator, or know the database maintenance word.
- You must either be the creator of the log identifier, or have system manager (SM) or operator (OP) capability.
- No other users are accessing the database. The database may be concurrently accessed by users when the >CONTROL command is specified with the MODE4 option. Refer to the >CONTROL MODE4 command discussed with DBRECOV in the next chapter.

If the >RECOVER command succeeds, recovery can be initiated by typing the >RUN command.

ROLLBACK Command

The >ROLLBACK command is used for roll-back recovery. It designates the name of a database to be rolled back. Any incomplete transactions at the time of the system failure are rolled out. Multiple databases may be roll-back recovered by entering more than one database name after the command.

Conditions necessary for the success of the >ROLLBACK command include the following:

- The database must be accessible to you when running DBRECOV. If the database resides in a group or account different from the your logon, the MPE/iX file security must permit the user read and write access to the database files.
- The log identifier characteristics (name, password, log file name, and device type) must not have been altered since the log file was generated.
- The database must be enabled for recovery and roll-back.
- All databases specified for roll-back recovery must contain the same log identifier.
- You must be the database creator, or know the database maintenance word.
- You must either be the creator of the log identifier, or have system manager (SM) or operator (OP) capability.
- No other users can be accessing the database when >ROLLBACK is called. The database may be concurrently accessed by users when the >CONTROL command is specified with the 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.

RUN Command

After the >RUN command is given, the DBRECOV program recovers the specified databases, 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 database administrator) must have access to the log file. This implies either knowing the maintenance word and having system manager (SM) or operator (OP) capability or being the creator of the log identifier with read access to the log file if it resides on disk 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.

Recovery Tables

The first three of the following four tables are displayed, along with statistics, by every execution of the recovery system. The last table is displayed only if the user recovery file feature is used.

```
LOGFILE PROCESS STARTED ON MON, AUG 14, 1989,  5:48 PM
LOGFILE PROCESS TERMINATED ON MON, AUG 14, 1989,  5:49 PM

*************************************************************************
*  1                          PROCESS STATISTICS                        *
*                                                                       *
*LOG# TIME   NAME  ACCOUNT  PROGRAM  DATABASE  TRANS  PUTS  DELS UPS   *
*---- -----  ----  -------  -------  --------- -----  ----  ---- ---   *
* 1.1 15.45   TST     MKTG  INVENTRY   ORDERS     145   145     0 0    *
* 2.1 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 logging user process number assigned to each process by the OPENLOG intrinsic, the logon name and

account, program name, and transaction statistics. This table contains one entry for each process that logged transactions to the log file. An asterisk appears 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 DATABASE STATISTICS table, the total number of transactions is given for each database recovered. The columns "TRANS, PUTS, DELETES, UPDATES" indicate the number of transactions recovered in roll-forward recovery, or rolled back if using roll-back recovery.

NOTE If you need this table, along with statistics, without actually performing the recovery, use >CONTROL STATS to display this information. The >PRINT DBTABLE command also displays this table, but does not include the statistics; use this command if you only need to list the names of the databases specified with >RECOVER commands.

---

3 The LOGGING SYSTEM table should have only one entry, 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 MPE/iX 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 each log file contains header and trailer records.

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 MPE/iX FILE commands. Note that all of these tables can be returned without recovering a database by using the >CONTROL STATS option when running the recovery program. Roll-back recovery ignores the *rmode* parameter.

Post-Recovery Options

After a recovery has completed, there are three procedural options. The option chosen determines the recovery procedure in the event of a subsequent system failure. Together, the database administrator and system manager or console operator should agree upon the best post-recovery procedure to avoid confusion at recovery time. The options available after recovery include:

1. Making a new database backup copy and starting a new log file from the console with the `CHANGELOG` command.

   If you use TurboSTORE/iX 7x24 True-Online Backup (with `ONLINE=START` or `ONLINE=END` option), you can start a new log cycle, or restart with the same log file.

   In the event of a subsequent system failure, the new database backup copy is restored and recovered against the new log file. This option allows for a straightforward recovery procedure but delays users from accessing the database until the new backup copy has been generated.

2. Resuming transaction logging to the same log file using the `RESTART` option without creating a new backup copy.

   In the event of a subsequent system failure, the old database 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 database after the first system failure without waiting for it to be stored.

3. Initiating logging to a new log file without creating a new backup copy.

   In the event of a system failure, the old database copy is restored and two recoveries are executed: the first using the old log file and the second using the new log file. This procedure is not recommended if option 2 is available.

   Until a new database backup copy is generated, if you consistently start logging to a new log file after a system failure, a total recovery preceded by *n* failures requires *n* executions of the recovery system.

   The second and subsequent recoveries of a database against more than one log file are not permitted unless the `DBSTORE` flag is disabled. This is because the first modification executed again from the first log file clears the `DBSTORE` flag from the database root file. Subsequent calls to `DBRECOV` can only succeed by specifying the `>CONTROL NOSTORE` option. Ensure that the log files are recovered in the proper consecutive order.

NOTE For options 2 and 3, do not restart a log file before the database has been recovered after a system failure because of the following reasons:

- Some log records could have been lost in the system failure, and
- The log file may not be consistent with the true state of the database.

A recovery is necessary to bring the database and log file into agreement before restarting the log process.

The Mirror Database

Transaction logging and regular backups are good maintenance. However, if databases must be accessible at all times and cannot be down even for maintenance, then a different maintenance method is needed. A system can be set up for constant access or "high availability," and still have controlled maintenance.

The mirror database is the fundamental element in creating a high availability database system. This system consists of two identical databases on two separate computer systems. One database is housed on a primary system and is constantly accessible to users and application programs. The other "mirror" database resides on the secondary system and is used for maintenance.

To establish a mirror database, the following requirements are necessary:

- Two systems--one primary, one secondary.
- Two identical copies of the database(s) are 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.
- Periodically, the log file containing the transactions must be moved or copied to the secondary system, and used to update the database(s) on the secondary system. *It is recommended that the log file be kept on a private volume separate from the database or on magnetic tape.*

After the secondary system is established, it can be used to make backups of the database. The primary system never has to be brought down for maintenance.

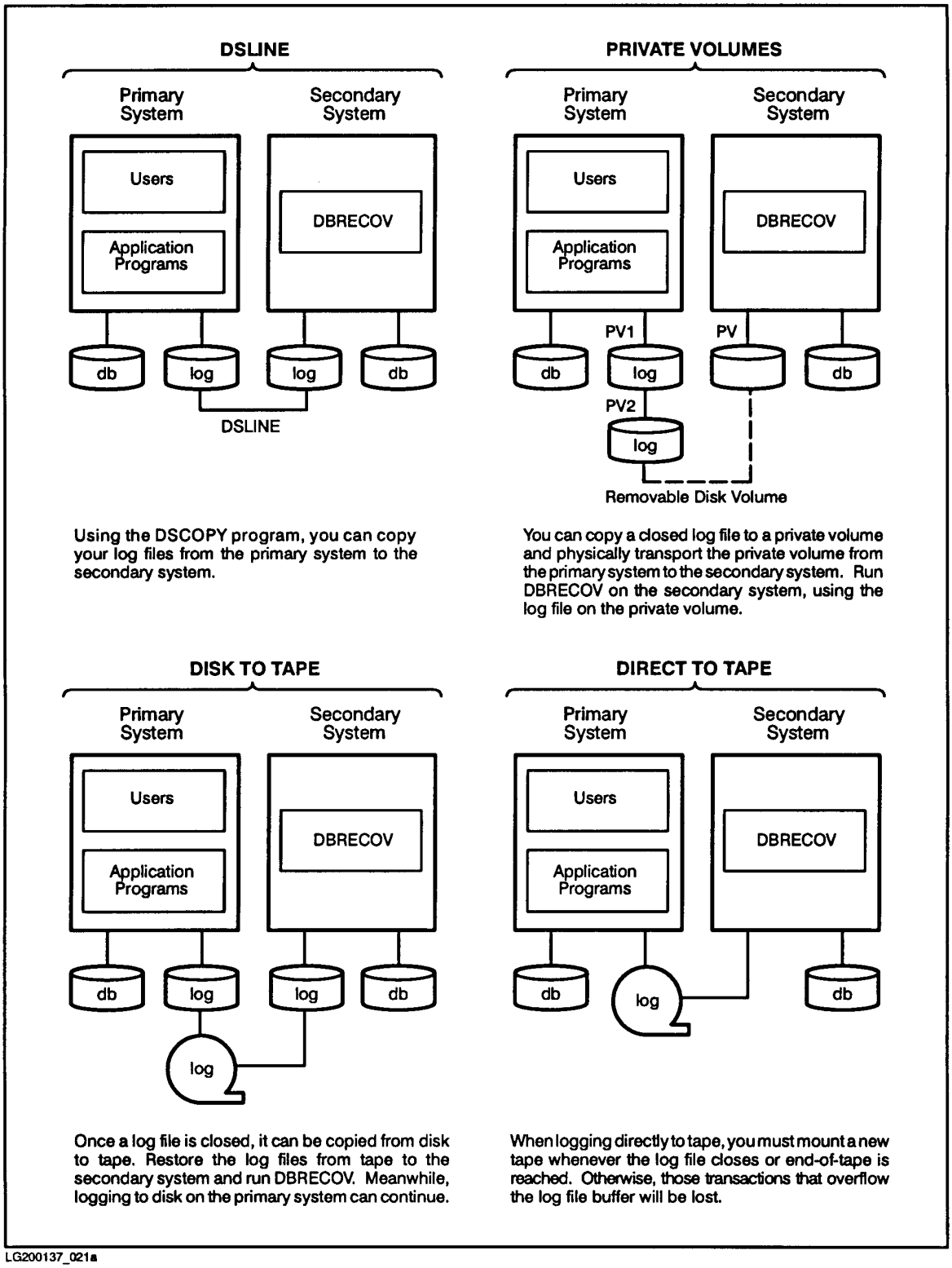
Transferring Log Files

The GETLOG command with the AUTO option and the CHANGELOG command provide the capability to schedule secondary system backups through various methods of logging. Listed below are four ways of copying log files from the primary to the secondary system. The method chosen should depend on the maintenance needs.

1. Copying files over a direct DSLINE from the primary to the secondary system.
2. Logging to a private volume, entering the CHANGELOG command to start a new log file, copying the closed log file to another private volume, and physically transporting that second private volume to the secondary system.
3. Logging to disk, entering the CHANGELOG command to start a new log file, copying the closed log file from disk to tape, and restoring the log file from tape to a disk on the secondary system.
4. Logging directly to tape and mounting the tape on the secondary system.

Figure 7-3. illustrates the four methods of copying log files as listed above.

Figure 7-3. Transferring Log Files to the Secondary System



Maintaining the Mirror Database

After the mirror database system is set up, the DBRECOV STOP-RESTART feature is used to maintain the secondary database. To start the initial DBRECOV procedure, the user must make sure logging is enabled on the primary system and that either the MPE/iX GETLOG AUTO option or CHANGELOG is being used. These MPE/iX options make logging without interruption on the primary system possible, thus increasing the availability of the databases. For more information on logging options refer to "Logging Preparation" and "Logging Maintenance" earlier in this chapter. Appendix G provides a brief outline of logging to disk and logging to tape.

After the log files are transferred to the secondary system (the mirror database system), they are applied to the mirror database using the DBRECOV roll-forward recovery process. The STOP-RESTART feature of DBRECOV is the key to making the mirror database system a workable maintenance method; for a detailed discussion of this feature, refer to the next section entitled "Performing DBRECOV STOP-RESTART." This feature adds the capability to CON[TINUE] or STOP the recovery process on the secondary system if DBRECOV cannot find the next log file in the log set. Whenever DBRECOV cannot find the next log file in a log set, the recovery process on the secondary system can be stopped, the databases can be backed up, and recovery can then 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 at the console to CON[TINUE] or STOP the recovery process.

If the reply is CON[TINUE], DBRECOV keeps searching for the next log file. When the next log file is found, DBRECOV resumes roll-forward recovery on the mirror database. The CON[TINUE] or STOP prompt appears as long as DBRECOV cannot find the next log file in the log set. DBRECOV is stopped if the STOP reply is entered and a RESTART file containing all the necessary information to restart recovery is created.

After the DBRECOV process is stopped, backup of the database in a consistent state can be done and limited database maintenance on the secondary system can be performed. Some DBUTIL functions cannot be performed while the DBRECOV process is stopped. If the database is in RESTART mode, the following DBUTIL processes cannot be performed:

- Access is not allowed in order to keep the database logically consistent.
- Resetting the maintenance word is not allowed. If the maintenance word were to be reset, RESTART would be impossible.
- Purging or erasing the database is not allowed. If either of these options were used in DBUTIL, the recovery process would be invalidated. (The user must run DBRECOV,ABORT or DBRECOV,PURGE before purging or erasing the database.)

DBRECOV,RESTART restarts the roll-forward recovery process from the point it was stopped. DBRECOV uses the information in the RESTART file to restart recovery. DBRECOV continues until, once again, it cannot find the next log file in the log set. The prompt to CON[TINUE] or STOP is displayed and backup of the database 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 database 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 database process again.

Performing DBRECOV STOP-RESTART

The processes involved in using the STOP-RESTART feature of DBRECOV are discussed here. They are broken down as follows:

- Stopping DBRECOV
- Storing the Databases
- Restarting DBRECOV

The following sections are included if problems are encountered while performing STOP-RESTART:

- Aborting DBRECOV
- Purging a RESTART file

Stopping DBRECOV

DBRECOV rolls 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 prints the following 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 CON[TINUE] on the console. DBRECOV will look for *filexxx* again. The roll-forward process continues 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, or the log file might have been renamed or be on a tape that was not mounted. This provides an opportunity to STOP recovery and perform maintenance on the database. Refer to "Storing the Databases" next in this chapter. To stop recovery, simply reply STOP at the console. A list of the databases 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; it also enables the RESTART flags and disables the access flags of the databases that are

recorded in the RESTART file.

```
DATABASE(S) WITH RECOVERY SUSPENDED:
     base1.group.acct
     base2.group.acct
            .
            .
            .
```

This is a list of the databases that are in the RESTART file. These database names are specified later when either the RESTART or ABORT options are used. The RESTART file name is the same as the *logid* name used in the GETLOG and the LOG,START commands when logging was enabled on the primary system.

DBRECOV then prints the name of the log file it needs to restart recovery, the record number beginning an internal structure, 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/iX, where the command DBUTIL >>SHOW *database name* FLAGS can be used to display the recovery state, that is, whether the database in recovery has been set for RESTART.

When running multiple recovery processes from the same log file, the user needs to equate the *logid*, that 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 Databases

The databases 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 database administrator should store the databases, 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 database backups, it is modified when recovery is restarted. Without the previous RESTART file or the log files, the database backup copies cannot be used to RESTART recovery in case the current RESTART fails.

The RESTART file and the databases have associated time stamps telling DBRECOV which RESTART file goes with which databases. When 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 succeed.

The method used to store the databases, RESTART file, and log files depends on the medium used for user logging, as follows:

- If logging to tape, the log files are already stored on a transportable medium and backing them up is not necessary. However, the log files must be grouped with the database and RESTART file backups. If the user does not keep track of which log files go

with which databases, the RESTART of recovery from a backup is not possible. To restart recovery from a backup, the user needs to restore the *tapes* containing the following:

- The databases.
- The RESTART file.
- All log files processed since the last successful RESTART. When running DBRECOV, the tapes containing the log files must be mounted in the correct sequence according to the tape volume labels.

- If logging to disk, remember to store the log files that were rolled forward since the last successful RESTART along with the RESTART file and the databases. Logging to disk makes it easier to keep the log files grouped with the databases and RESTART file because all the log files can be stored at the same time when recovery is stopped. Use an MPE/iX STORE command 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 database is ORDERS, name the *logid* ORDERRS (where RS represents RESTART), and the log file ORDER001. The user can store all the files with one MPE/iX command as follows:

```
:STORE ORDER@
```

NOTE To avoid incompatible time stamps, it is important to store the RESTART file at the same time that the databases are stored. If logging to disk, 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 database maintenance is completed, enter the following RUN command:

```
:RUN DBRECOV.PUB.SYS,RESTART
```

DBRECOV requests the name of one of the databases in the RESTART file:

```
WHICH DATABASE?
```

If the user types in the name of a nonexistent database, another prompt for the database recorded in the RESTART file appears. Once again enter the name of a database in the RESTART file. From the database name that is entered, DBRECOV determines the name of the RESTART file, tries to open it, and restarts 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 the user is returned to the MPE/iX 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 RESTART again. If the RESTART file cannot be located, go back to the previous tape, restore the databases 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 are reprocessed, and the roll-forward process continues with the current log file.

When the correct RESTART file is opened, DBRECOV looks 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 prints the following error message:

```
RESTART FILE NOT COMPATIBLE WITH THIS VERSION OF DBRECOV
```

and the user is returned to the MPE/iX prompt. This message means that another version of DBRECOV is running other than the version that created the current RESTART file. Install the correct version of TurboIMAGE/XL 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 using the same user and account names that were used when DBRECOV was originally suspended, and run DBRECOV again.

When the RESTART file is opened, DBRECOV tries to open all databases identified in the RESTART file. For each database that cannot be opened, *DBRECOV* displays the following message:

```
Can't re-open DATABASE basename
```

RESTART is terminated and the user is returned to the MPE/iX prompt. Make sure the correct databases are on the system. If the databases are the correct ones, but they still cannot be opened, use the DBRECOV,ABORT command (discussed in the next section) and RESTART recovery from the previous STOP point.

When all the databases have been opened, DBRECOV checks to make sure all the databases in the RESTART file are set for RESTART. When DBRECOV encounters a database not in RESTART mode, it displays the message:

```
DATABASE basename IS NOT IN RESTART MODE
RESTART TERMINATED
```

RESTART is terminated and the user is returned to the MPE/iX prompt. Make sure the correct databases are loaded on the system. If the databases are the correct ones and RESTART is still not accepted, use the DBRECOV,ABORT (discussed in the next section)

command and RESTART from the previous STOP point.

To start the recovery process again, find out why the databases are not in RESTART mode and try to correct the problem. If the problem cannot be corrected, take either of the following steps:

- Go to the previous STOP point and use the databases and RESTART file stored to restart roll-forward recovery, or
- ABORT the current RESTART process. Disable user access on the primary databases 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.

If all databases are found, and they are in RESTART mode, then the time stamps in the database root file are 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 DATABASE TIME STAMPS
```

This indicates incompatibility of the RESTART file and the databases. The user is returned to the MPE/iX prompt. Use the same steps given above to recover from a time stamp error. After all the compatibility checks have passed, DBRECOV prints a table of the databases to be recovered:

```
DATABASE(S) TO BE RESTARTED:
     base1.group.acct
     base2.group.acct
          .
          .
          .
```

The user is then prompted to confirm the restart:

```
CONTINUE WITH RECOVERY (N/Y)?
```

Respond Y or YES to continue, or type N or NO (or press carriage return) to return to the STOP point. If any of the databases cannot be opened during recovery, an MPE/iX file error is returned and DBRECOV RESTART is terminated. When this happens, go back to the previous STOP point and use the databases, log files, and the RESTART file to RESTART recovery. 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 database process again. When the recovery process terminates, the user is returned to the MPE/iX prompt.

Two ways of continuing to mirror the databases are listed here:

- Go to the previous STOP point and use the databases, 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.
- Disable user access on the primary databases 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.

CAUTION When recovery is aborted (refer to the following discussion), the current RESTART file is purged and RESTART must be done from the previous STOP point.

Aborting DBRECOV

To run the ABORT option:

```
:RUN DBRECOV.PUB.SYS,ABORT
```

Just like the RESTART option the prompt for a database in the RESTART file appears:

```
WHICH DATABASE?
```

If a nonexistent database name is entered, an error message is printed and the user is prompted once again to enter the name of a database in the RESTART file. Using the database name 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/iX 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 running DBRECOV,ABORT again.

When the correct RESTART file is opened, DBRECOV looks 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, DBRECOV prints the following error message:

```
RESTART FILE NOT COMPATIBLE WITH THIS VERSION OF DBRECOV
```

and the MPE/iX prompt is returned. This message means the version of DBRECOV is not the same as the version that created the current RESTART file. Install the correct version of TurboIMAGE/XL 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 using the same user and account names that were used when DBRECOV was originally suspended, and run DBRECOV again.

When the RESTART file is successfully opened, DBRECOV identifies all the databases in the RESTART file, and verifies that they are in RESTART mode. DBRECOV then checks the time stamps in the RESTART file and the databases to make sure they match. If the time stamps do not match, the following message is printed:

```
RESTART TIME STAMPS DON'T AGREE WITH DATABASE TIME STAMPS
```

This indicates incompatibility of the RESTART file with the data bases. The MPE/iX prompt is returned. Locate the correct RESTART file, and run DBRECOV,ABORT again.

After the RESTART file is opened, DBRECOV tries to open all databases identified in the RESTART file. For each database that cannot be opened, DBRECOV displays the following message:

```
Can't re-open DATABASE basename
CONTINUE WITH ABORT (N/Y)?
```

DBRECOV then allows the user to make sure that the ABORT is desired. If not all databases are in the RESTART file, it may mean that this a different set of databases. Respond Y or YES to continue the ABORT, and N, NO (or press carriage return) to stop the ABORT.

DBRECOV then checks to make sure all the databases in the RESTART file are set for RESTART. When DBRECOV encounters a database not in RESTART mode, it prompts:

```
DATABASE basename IS NOT IN RESTART MODE
CONTINUE (N/Y)?
```

Respond Y or YES to continue the ABORT, and N, NO (or press carriage return) to stop the ABORT.

When all compatibility checks have passed, DBRECOV displays all databases in the RESTART file:

```
DATABASE(S) WITH RECOVERY TO BE ABORTED:
     base1.group.acct
     base2.group.acct
       .
       .
```

If not all of the databases can be opened, DBRECOV prints an MPE/iX 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 press carriage return) to stop the ABORT.

After ABORT is successfully completed, the current RESTART file is purged, the MPE/iX prompt is returned, and the user can issue a DBUTIL >>SHOW command. The RESTART flag

is disabled, and the database access flag is reset to the state it was in 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 databases 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 databases and RESTART file of a previous STOP point.

CAUTION When using PURGE on a RESTART file, RESTART must be done from the previous STOP point.

```
:RUN DBRECOV.PUB.SYS,PURGE
```

DBRECOV prompts for the name of the RESTART file:

```
ENTER RESTART FILENAME?
```

Enter the filename displayed when DBRECOV was stopped. DBRECOV opens the file and verifies 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 displays the table of databases in the RESTART file:

```
RESTART FILE CONTAINS FOLLOWING DATABASE(S):
     base1.group.acct
     base2.group.acct
          .
          .
          .
```

All the databases will be opened, and DBRECOV checks if they are all enabled for RESTART. If they are all in RESTART mode, the following message is printed and DBRECOV is terminated:

```
DATABASE base1.group.acct IS IN RESTART MODE.
DATABASE base2.group.acct IS IN RESTART MODE.
RECOVERY SUSPENDED - USE DBRECOV,ABORT TO ABORT RECOVERY.
```

Run DBRECOV,ABORT to purge the RESTART file.

If none of the databases in the RESTART file are set for RESTART, the RESTART file is purged with no further confirmation.

If some of the databases are not found, DBRECOV prompts for confirmation to purge the RESTART file:

```
Can't re-open DATABASE basename.
CONTINUE WITH PURGE (N/Y)?
```

DBRECOV allows you to make sure that you wish to purge this recovery process. If not all databases are in the RESTART file, this could be a different set of databases. Respond Y or YES to purge the RESTART file, or N, NO, (or press 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. Listed below are the four basic steps used to reestablish the mirror database system after an abnormal termination of DBRECOV:

1. Disable user access on the primary system and store the databases from the primary system.
2. Purge the databases on the secondary system.
3. Restore the databases from the primary system onto the secondary system.
4. Start a new log set, enable user access on the primary system and start roll-forward recovery on the secondary system.

Controlling the Logging Process

Backups on the secondary system are made more efficient by controlling the logging processes on the primary system. Some important factors to consider before enabling logging on the primary system follow:

- When logging to tape and to allow roll-back recovery, the database must be on the system volume set. Logging to tape eliminates the step of storing log files with the databases once they are rolled forward on the secondary system. However, keep track of log file tapes that correspond with each database and RESTART file backup tapes. Logging to tape requires a dedicated tape drive.
- When logging to disk and to allow roll-back recovery, the database and the user log file must be on the same volume set. Logging to disk enables storing log files and the databases on a single tape using an MPE/iX STORE command rather than a DBSTORE command. When logging to disk, remember to backup all log files that were processed after the last DBRECOV,RESTART along with the databases and RESTART file.
- When naming data sets, follow proper naming conventions. This will make storing the databases, log files, and RESTART file much easier and eliminate the use of several different tapes for the log files. If naming conventions are followed, an MPE/iX STORE command using the "@" sign followed by the database name can be used to store the log files, database, and RESTART file.
- When changing log files, 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.

Log File Size

Determine the size of 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 database 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. It also means that the CHANGELOG maximum file limit of 999 will be exhausted quickly. In the event that this limit is exhausted, stop, backup the database, and reinitiate logging on the primary system.

One disadvantage of having several small log files is in the application of STOP-RESTART. DBRECOV prompts to CON[TINUE] or STOP recovery if it is between log files in a log set, and it cannot find the next log file. Therefore, the prompts to CON[TINUE] or STOP are more frequent when there are several small log files.

An alternate 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. When you are ready to copy the log file over to the secondary system, change to the next log file on the primary system, copy the current one to the secondary system, and start recovery. This method requires someone at the system console to monitor the logging and database maintenance processes. If you want to schedule backups on the secondary system around certain times of the day, for example, at the beginning and end of a work day, use this logging procedure on the mirror database. You can log a full shift's transactions and then manually issue a CHANGELOG command at the system console to create a new log file in the log set. Even if the GETLOG command AUTO option was specified when logging was enabled, a manual CHANGELOG command can also be issued at any time. The closed log file is transferred to the secondary system, and the DBRECOV roll-forward recovery process can be continued on the secondary databases.

After the log file has been processed, DBRECOV looks for the next log file in the log set on the secondary system. If the next log file on the primary system is in use, the user is prompted to CON[TINUE] or STOP. At this point, recovery can be stopped and the secondary database 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 databases.

8 Using the Database Utilities

The TurboIMAGE/XL utilities create and initialize the database files and perform various maintenance functions, such as restructuring a database. This chapter discusses these utilities and the syntax of each utility.

You must be the database creator to execute the >>CREATE command of the DBUTIL utility program or to change or remove the maintenance word with the DBUTIL >>SET command. The **database creator** is defined by the logon group and account that was used when the Schema Processor created the root file. To operate the other utilities or to enter other DBUTIL commands (with the exception of the >>SHOW command), you need not be the database creator provided you know the maintenance word. *If no maintenance word is defined, only the database creator can execute the other utilities and the DBUTIL commands that require a maintenance word*. The exception to this rule is that a user with system manager (SM) capability can execute the DBUTIL >>SHOW command on any database without having to supply the maintenance word.

Restructuring the Database with TurboIMAGE/XL Utilities

Using the utilities DBUNLOAD, DBUTIL, DBSCHEMA, and DBLOAD, certain changes to the structure of an existing database, such as capacity changes, adding additional items and sets, and repacking data sets, can be made without having to write special programs to transfer data from the old database to the new one. The general sequence of operations to do this is as follows:

1. Run DBUNLOAD on the old database, copying all the data entries to tape.
2. Purge the old database using the DBUTIL >>PURGE command.
3. Redefine the database using the same database name by modifying the schema file, and create a new root file with the Schema Processor, DBSCHEMA.
4. Use the DBUTIL >>CREATE command to create and initialize the data sets of the new database.
5. Run DBLOAD on the new database using the tape created in step 1 to put the data into the new database.

The above procedure allows only the supported structural changes to the schema listed below. DBLOAD does not prohibit other changes; however, the data is not guaranteed to be consistent. Supported schema changes yield structurally intact databases and always result in a good transformation. Commercial software packages are available that can perform other structural changes without doing a DBUNLOAD and DBLOAD operation.

Supported Structural Changes Using DBUNLOAD and DBLOAD

Any of the following schema changes, alone or combined, always result in a successfully transformed database:

- 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.
- Adding and deleting automatic master paths.

Unsupported Structural Changes Using DBUNLOAD and LOAD

The following structural changes are legitimate only in certain circumstances and can 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.
- Adding or deleting a data set at the end of a schema.
- Changing the native language definition for the database.

These are unsupported schema changes. DBLOAD does not prohibit these changes; however, the data is not guaranteed to be consistent. A change must be judged in light of the particular database and the functioning of DBUNLOAD and DBLOAD, described later in this chapter.

Basically, all entries from an old data set are put into the corresponding new 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 entry length of the new data set. DBUNLOAD and DBLOAD always handle full entries, without regard to item positions or lengths. If the new data set entry is defined with the items in a different order than the old data set, DBLOAD may not fail but the data set entries, nevertheless, will be invalid. For example, data of type real may now occupy the position of a character type item.

The number of the data set is determined by the order in which the data sets we re entered in the schema file. Therefore, because data sets are loaded by number, additions and deletions should be made at the end of the schema. Data set one would correspond with the first data set appearing in the schema. DBLOAD always returns 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 DBLOAD to continue after the warning if you are confident that the database will not be corrupted.

In some circumstances, the load completes, but data is lost. For example, data is lost if the capacity of a data set is reduced in the new database to less than the number of data set entries on the tape. If this is not desired, increase the capacity of the data base and restart the DBLOAD process (refer to "Restructuring the Database with TurboIMAGE/XL Utilities" earlier in this chapter.)

Summary of Utility Routines

Here is a brief summary of the utilities, their commands, and their functions.

Table 8-1. TurboIMAGE/XL Utilities

Program	Commands	Function
DBLOAD		Loads data entries, which were copied to tape by DBUNLOAD, back into the data sets.
DBRECOV		Performs database recovery from log files.
	CONTROL	Controls various options that affect the execution of DBRECOV. *Refer to the discussion of DBRECOV in this chapter for valid options when executing.*
	EXIT	Terminates DBRECOV without re-executing any transactions.
	FILE	Routes log records to individual user files and returns information about recovery.
	PRINT	Prints information about databases or user files specified for recovery.
	RECOVER	Designates name of database(s) to be roll-forward recovered.
	ROLLBACK	Defines name of database(s) to be roll-back recovered.
	RUN	Initiates recovery process.
DBRESTOR		Copies the database to disk from magnetic tape volumes created by DBSTORE, or by the MPE/iX STORE or SYSGEN command.
DBSTORE		Copies entire database including root file to magnetic tape volumes.
DBUNLOAD		Copies data entries to specially formatted magnetic tape volumes; arranges entries in each data set via the primary path if the chained entry point is used.
DBUTIL		Allows you to perform several database functions, such as setting database flags, changing database security, creating or purging a database, and determining current users and the status of locks.
	ACTIVATE	Prepares a database-access file which is used to access a remote database.
	ADDINDEX	Adds the associated B-Tree index.
	CREATE	Creates and initializes a database file for each data set.
	DEACTIVATE	Deactivates a database-access file which is used to access a remote database.

Table 8-1. TurboIMAGE/XL Utilities

Program	Commands	Function
	DETACH	Detaches the database from the attached DBEnvironment(s).
	DISABLE	Disables logging, roll-back recovery, ILR, AUTODEFER, HWMPUT, MUSTRECOVER, PREFETCH, access, third-party indexing, and dumping options.
	DROPINDEX	Drops the associated B-Tree index.
	ENABLE	Enables logging, roll-back recovery, ILR, AUTODEFER, HWMPUT, MUSTRECOVER, PREFETCH, access, third-party indexing, and dumping options.
	ERASE	Erases existing data entries from all data sets. Used before loading stored data entries back into the database. Also disables options, such as logging, ILR, roll-back recovery, and third-party indexing.
	EXIT	Terminates DBUTIL program execution.
	HELP	Lists all DBUTIL commands.
	MOVE	Moves TurboIMAGE/XL files across devices.
	PURGE	Purges entire database including root file, data sets, and any third-party indexes. Used before restoring a stored database and before creating a new, restructured database.
	REBUILDINDEX	Rebuilds the B-Tree index file for a specified dataset that should have an index file.
	REDO	Same as MPE/iX REDO command.
	RELEASE	Suspends MPE/iX security provisions for the root file, data sets, and any third-party indexes.
	SECURE	Restores MPE/iX security provisions suspended by RELEASE.
	SET	Changes or removes the maintenance word or password, stores log identifier and password into root file, specifies the setting for the critical item update (CIUPDATE) option, and changes the native language of the database. For databases that will be migrated to MPE V, specifies the number of buffers to be used.
	SHOW	Used to display information about a database, such as flags, users, the status of locks, DBEnvironments to which the database is attached, and if third-party indices are registered.
	VERIFY	Used to determine whether a database-access file is activated or deactivated.

Utility Program Operation

Database utilities can be run in either job or session mode. With the exception of DBUTIL's >>SHOW command, DBUTIL, DBSTORE, DBRESTOR, DBUNLOAD, and DBLOAD all require you to be logged on in the group and account that contains the database root file. Consequently, these utilities cannot be used with a remote database unless you initiate a remote session and run the utility as part of that session. The DBUTIL, DBSTORE and DBRESTOR utilities do not allow you to use the MPE/iX FILE command to equate a database or database-access file.

CAUTION DBUNLOAD and DBLOAD do allow MPE/iX FILE commands to equate a database and can redirect the database to a different file. Except in a controlled environment, you should not use the MPE/iX FILE command to redirect a database or database-access file to a different file, because that file can be purged easily.

The DBRECOV utility is not included in the discussion above because it is an exception. With DBRECOV, MPE/iX FILE commands are permissible and you need not be logged on to the same group and account as the log file. However, DBRECOV must be invoked on the system where the database resides.

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 (including account name) that was used when the Schema Processor created the root file; this verifies to TurboIMAGE/XL that you are the database creator. To operate the other database utilities or enter other DBUTIL commands, you need not be the database creator provided you know the maintenance word. *If no maintenance word is defined, only the database creator can execute the other utilities and the DBUTIL commands that require a maintenance word.* The exception to this rule is that a user with system manager (SM) capability can use the DBUTIL >>SHOW command on any database without having to supply the maintenance word.

NOTE To maintain compatibility with earlier versions of DBUTIL, the >>CREATE, >>ERASE, and >>PURGE commands can also be executed by specifying them as DBUTIL entry points.

Backup Files

The backup files created by DBSTORE and DBUNLOAD can be written only to magnetic tape volumes. In the discussion of the utilities that follows, the term **volume** refers to a magnetic reel.

Error Messages

Some of the error messages are described with the operating instructions for the utilities. Appendix A contains a complete listing of the error messages issued by these programs.

DBLOAD

Loads data entries from the backup volume(s) created by the DBUNLOAD utility into data sets of the database.

Operation

```
1          [:FILE DBLOAD[=filename] [;DEV=device] ]

2          :RUN DBLOAD.PUB.SYS
                .
                .
                .

3          WHICH DATABASE? database name [/maint word]
           WARNING: The LANGUAGE of the database is DIFFERENT from     the
           language found on the DBLOAD MEDIA.   Continue DBLOAD operation?
           (Y/N):

4          DATA SET m: x ENTRIES
                .
                .
                .
5          END OF VOLUME n, y READ ERRORS RECOVERED

6          DBLOAD OPERATION COMPLETED

              END OF PROGRAM
```

(Refer to "Operation Discussion" later in this section.)

The volume(s) must have been produced by the DBUNLOAD program, and the database name on the volume must be exactly the same as the database 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 database name or maintenance word specified is different from the DBUNLOAD 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 database, the DBUTIL ERASE command must be used prior to DBLOAD unless the database has been purged and re-created. Executing the >>ERASE command reinitializes the data sets to an empty state while keeping the root file and data sets as catalogued MPE/iX files on the disk.

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 database 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 an at-sign followed by a semicolon (@;). 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 key 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 database. If the database is already open to any other process, DBLOAD terminates and prints the message:

```
DATABASE IN USE
```

Parameters

filename is the name (up to 8 characters) that replaces DBLOAD in the mount request at the operator's console.

device is the device class name of the device from which the data entries are to be loaded. Tape is the only supported device class.

database name is the name of a TurboIMAGE/XL database root file created in the current session or job's account and logon group.

maint word is the maintenance word defined by the database creator. This word must be supplied by anyone other than the database creator.

Message Variables

m is the number of the last data set loaded from the backup volume.

x is the number of entries loaded into the specified data set. *x* 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.

n is the volume number.

y is the number of read errors from which DBLOAD recovered.

Operation Discussion

1 Is an optional file equation that 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 in the PUB group and SYS account.

3 In session mode, DBLOAD prompts for the database name and maintenance word. In job mode, the database name and maintenance word, if any, must be in the record immediately following the RUN command.

The language ID of the database is stored along with data when DBUNLOAD has been used to copy the database to tape. If the database native language (on disk) is not consistent with the system level native language (on tape), the following message will appear (refer to appendix A for more information):

```
WARNING: The LANGUAGE of the database is DIFFERENT from
the language found on the DBLOAD MEDIA.
Continue DBLOAD operation? (Y/N):
```

NOTE When using DBLOAD, the database language must match the language ID stored in the backup media. If they do not match, DBLOAD will give you a warning messages in a session, and if you reply **Y**, the DBLOAD will continue. However, in a job, DBLOAD does not load the database.

---

4 After each data set is copied, DBLOAD prints a message on the list file 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 *z* is the logical device number of the unit, *XXXX* is the database name, and *n* 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 XXXXn ON LOGICAL DEVICE z
```

If the operator mounts the wrong volume, DBLOAD informs the operator with the following message (where *z* is the logical device number):

```
WRONG VOLUME MOUNTED ON LOGICAL DEVICE z
```

DBLOAD then terminates and you must begin loading the database again. This requires executing the DBUTIL >>ERASE command 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 database name and DBLOAD opens the input file, a message is displayed on the system console. A tape must be mounted on the appropriate unit and identified through an operator reply. Refer to the *Volume Management Reference Manual*

for instructions about console interaction.

Using ControlY

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 m:x ENTRIES HAVE BEEN PROCESSED
```

Example

```
:RUN DBLOAD.PUB.SYS

WHICH DATABASE?  ORDERS/SELL
DATA SET 1:   AUTOMATIC MASTER
DATA SET 2:   19 ENTRIES
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
DATABASE LOADED

END OF PROGRAM
```

Initiate execution of DBLOAD. Supply the database name and maintenance word. DBLOAD indicates the number of entries copied. Data set 1 is an automatic master so 0 entries are copied; the entries are created as related detail entries are copied to the database.

One volume was copied with no read errors.

NOTE For optimum performance, DBLOAD uses deferred output when it adds entries to a database. With deferred output, data and structural information cannot be written back to disk each time DBPUT returns to the DBLOAD program. As a result, the database is not considered to be logically or structurally complete on disk until the DBLOAD is complete. During DBLOAD the database being loaded is considered inconsistent ("bad") and only at the completion of a DBLOAD run is the database considered consistent ("good") again.

During a load if an MPE/iX or hardware failure occurs, the database is definitely not structurally intact, and it returns its "bad" flag. After the system is brought back up, TurboIMAGE/XL does not allow the database to be opened for normal access. If you get a "bad database" error in such a situation, erase the database with DBUTIL and then perform the load again. (For more information on the error message "Bad Database" refer to appendix A). Alternatively, the database can be purged with DBUTIL and then restored from a backup copy.

DBRECOV

The DBRECOV program usually is executed after a backup database copy has been restored by running DBRESTOR in the event of a system failure. DBRECOV reads the log file containing records of all database modifications and re-executes the transactions against the restored database(s). The DBRECOV >FILE command enables individual users to be informed of the extent of recovery. For more information on roll-forward recovery, roll-back recovery, and the DBRECOV STOP-RESTART feature, refer to chapter 7.

DBRECOV also uses a mirror database on a secondary system as 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 step-by-step mirror database maintenance.

DBRECOV can perform rollforward recovery of TurboIMAGE/XL databases stored using the TurboSTORE/iX 7x24 True-Online Backup. No new option for DBRECOV is needed.

The commands associated with DBRECOV are >CONTROL, >EXIT, >FILE, >PRINT, >RECOVER, >ROLLBACK and >RUN. Each command is discussed separately.

Operation

```
:RUN DBRECOV.PUB.SYS [,option]
```

Options

RESTART — restarts 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 database 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 in the PUB group and SYS account. The recovery system prints a banner indicating the version, date, and time. It then prompts for a command input.

Example 1

Roll-forward recovery of database ORDERS.

```
:RUN DBRECOV.PUB.SYS
>RECOVER ORDERS
DATABASE ORDERS LAST DBSTORED THURS, SEP 21, 1989, 8:30 AM
>RUN
```

Example 2

Roll-forward recovery of multiple databases ORDERS and RETAIL. PART and SALES are *filenames*, ADMIN and MKTG are *accounts* in the FILE commands. The 0 is the *rmode* and the 3 is the *fmode*.

```
:RUN DBRECOV.PUB.SYS
>RECOVER ORDERS
DATABASE ORDERS LAST DBSTORED MON, SEP 25, 1989, 6:40 PM
>CONTROL NOSTORE
>RECOVER RETAIL
>FILE PART,JOHN.ADMIN
>FILE SALES,MARY.MKTG,0,3
>RUN
```

Example 3

Roll-back recovery of multiple databases ORDERS and RETAIL.

```
:RUN DBRECOV.PUB.SYS
>CONTROL NOABORTS
>ROLLBACK ORDERS,RETAIL
DATABASE ORDERS LAST USED THURS, SEP 21, 1989, 6:00 PM
DATABASE RETAIL LAST USED FRI, SEP 22, 1989, 8:00 AM
>RUN
```

Example 4

DBRECOV STOP-RESTART recovery on database ORDERS. The recovery process is done on a secondary system with the mirror database 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 in a log set, the user can stop the recovery process and back up the secondary system. In the example, note that the restart file ORDERLOG is named after the database *logid*.

```
UNABLE TO OPEN LOG FILE ORDER005
REPLY `CONTINUE' OR `STOP' ON CONSOLE.
STOP

DATABASE(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 L;DEV=TAPE
:STORE ORDERS@;*L
:RUN DBRECOV.PUB.SYS,RESTART
```

```
WHICH DATABASE?  ORDERS

DATABASE(S) TO BE RESTARTED:
  ORDERS.DATAMGT.ADMIN

CONTINUE WITH RECOVERY (N/Y)?  Y
```

Text Reference

Chapter 7

>CONTROL

Used to control various options that affect the execution of DBRECOV. The options are STAMP, NOSTAMP, STORE, NOSTORE, ABORTS, NOABORTS, UNEND, NOUNEND, STOPTIME, ERRORS, STATS, NOSTATS, MODEX, MODE 4, EOF, MDBX, and NOMDBX.

Syntax

```
>CONTROL parameters [,parameters...]
```

Discussion

The >CONTROL options are described in detail on the next page. If the >CONTROL command is not used, the following default conditions apply:

STAMP	is the database time stamp and must correspond with the one written to the log file.
STORE	is the DBSTORE flag set in the database 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 were aborted prior to a system failure.
ERRORS=	during job (batch) execution allows zero errors (DBRECOV terminates), and during interactive sessions allows 30,000 errors.
MODEX	DBRECOV proceeds with exclusive access to the database, using deferred output (see discussion under DBCONTROL in chapter 5).
NOSTATS	if the database is not recovered, no tabulated information will be printed.
STOPTIME=	DBRECOV will recover all log records, regardless of the time stamp.
EOF=	DBRECOV will recover all log records in the log file.
MDBX	DBRECOV treats multiple database transactions contained in the log file as separate transactions.

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 time stamp check intact.

If additional databases are specified for simultaneous recovery, they are all governed by the same >CONTROL options.

In the specifications below, default options are shown in brackets [ ]. The default conditions for STOPTIME, ERRORS, and EOF are included with their descriptions.

Parameters

[STAMP]	is the time stamp in the database root file. It is compared with the time stamp in each DBOPEN log record in the log file. If the time stamps do not match, DBRECOV returns an error message, and terminates recovery for the offending database.
NOSTAMP	disables the check of the database and log file time stamps. Allows recovery to proceed regardless of the database and log file time stamps.
[STORE]	is the DBSTORE flag in the database root file and is checked to ensure that the database has not been modified between restoration and recovery. If the flag has been cleared, the >RECOVER command fails. The DBSTORE flag is set only when the database is stored using DBSTORE. It is cleared when the database is accessed by DBDELETE, DBPUT, or DBUPDATE.
NOSTORE	disables the check of the DBSTORE flag. Allows recovery to proceed whether or not the DBSTORE flag is set. Useful when the database has been stored by the MPE/iX STORE command rather than DBSTORE. Storing the database using the STORE command does not set the DBSTORE flag, and is not recommended.
[ABORTS]	when transactions do not complete due to a program abort, TurboIMAGE/XL appends an abnormal DBEND (DBABEND) to the log file 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/XL a user or program abort is abnormal, or incomplete. NOABORTS should only be used if all database modifications were stopped immediately after the abort and recovery was initiated. Otherwise, recovery can fail due to record file overflow (see below). For more information on both ABORTS and NOABORTS refer to chapter 7.
[NOUNEND]	causes DBRECOV to suppress incomplete transactions. Recovery can fail due to a record file overflow (see "Record Numbers" later in this section).
UNEND	prevents DBRECOV from suppressing incomplete transactions.

STOPTIME= *mm/dd/yy hh:mm* causes DBRECOV to impose an artificial end-of-file when the specified log record time stamp (supplied by MPE/iX) is encountered. All log records with subsequent time stamps will not be recovered. This feature is useful in the event of a user program failure; the database can be recovered to a point in time before the suspect program began execution.

Default condition: Log record time stamps 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 for batch jobs and ERRORS=30,000 for interactive sessions. The number of errors allowed can be altered by entering a revised ERRORS parameter.

STATS is used to obtain information from the log file without actually recovering a database. Requires use of a file equation to specify the log file. For example:

```
:FILE LOGFILE=ORDER001;DEV=TAPE;LABEL=LOG001
:RUN DBRECOV.PUB.SYS
>CONTROL STATS
>RUN
```

This example shows the log file ORDER001 residing on tape and belonging to an expandable file set (refer to the GETLOG command with AUTO option in the *MPE/iX Commands Reference Manual*. The recovery system responds by printing tabulated information from log files, similar to tables printed after a database recovery. However, no databases are actually opened or recovered.

[NOSTATS] negates the STATS option; tabulated information is not printed unless a database is recovered.

[MODEX] causes recovery to execute in exclusive (deferred) mode. No other users can access the database concurrent with recovery.

MODE4 recovery proceeds in DBOPEN mode 4, allowing users in mode 6 to access (read) the database while recovery is in process.

EOF=*nnnn* causes DBRECOV to impose an artificial end-of-log file 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 database can be recovered up to a record number preceding the suspect records. While logging is in progress, the MPE/iX SHOWLOGSTATUS command can be used to determine the current number of records logged before initiating a questionable program.

Default condition: All log records are recovered by DBRECOV.

[MDBX] causes DBRECOV to treat multiple database transactions contained in the

log file as separate transactions. If all of the databases involved in the multiple database transaction are not specified in the >RECOVER or >ROLLBACK command, DBRECOV will abort. It does not allow partial recovery of multiple data base transactions. If a multiple database transaction is dependent on any transaction that was not recovered, the multiple database transaction will be rolled out.

NOMDBX causes DBRECOV to treat multiple database transactions contained in the log file as single transactions. Therefore, each database can be recovered separately. This option is useful if only part of a multiple database transaction is to be recovered.

Record Numbers

DBRECOV identifies detail records by their record number. Suppressing aborted or unended transactions during recovery with the NOUNEND or NOABORTS options can cause subsequent detail calls to DBPUT to use different record numbers. In order to change old record numbers into new ones, DBRECOV uses an internal record table. The record table provides a "before" and "after" location of the record numbers for DBPUT calls.

Text Reference

Chapter 7

>EXIT

Used to terminate DBRECOV without recovering any databases.

Syntax

```
>EXIT
```

Text Reference

Chapter 7

>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/iX file reference: *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 copy log records associated with transactions successfully recovered. *rmode* can take one of the following values:

0	No records associated with recovered transactions are copied to the user file. (Default value.)
1	Log records corresponding to the last successfully recovered call to DBEND of each transaction block are copied.
2	The sequence of log records associated with the last successfully recovered transaction of each transaction block are copied. In addition, all DBMEMO log records which immediately follow this transaction are copied.
3	All log records associated with successfully recovered transactions for each transaction block are copied.

fmode directs recovery system to copy log records associated with transactions that failed to recover. Used with both roll-forward and roll-back recovery.

CAUTION The (roll-forward) recovery system cannot guarantee that all records associated with unsuccessfully recovered transactions can be copied, because log records which reside in the log system's memory buffers are lost in the event of a system failure. When accessing the database for critical

transactions, use DBEND mode 2 for immediate posting of the log system's memory buffer.

fmode can take one of the following values:

0	No records associated with failed transactions are copied. (Default value.)
1	Log records corresponding to the first unsuccessfully recovered call to DBBEGIN of each transaction block are copied.
2	The sequence of log records associated with the first unsuccessfully recovered transaction of each transaction block are copied.
3	All log records that could not be recovered are copied.

Discussion

The >FILE command copies qualified DBOPEN and DBCLOSE log records to each user's recovery file. See "File Command" in chapter 7 for a full discussion qualifying the return of log records. The optional *rmode* and *fmode* parameters 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 set by DBRECOV in the record itself. MPE/iX WRITELOG records returned by DBRECOV are variable length, because 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 an MPE/iX 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.

Example

```
>FILE PART/MGR,MARY/RYAN.MKTG,0,3
```

PART is the *filename*. MGR is the *lockword*. MARY is the *username* and RYAN is the *identifier*. MKTG is the account. The 0 is the *rmode*, and the 3 is the *fmode*.

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

Chapter 7

>PRINT

Prints the names of databases specified for recovery (DBTABLE option) or recovery files specified (FILETABLE option). Can be used as a check before actually initiating recovery with the >RUN command.

NOTE The >PRINT DBTABLE command produces the DATABASE STATISTICS table, but does not include statistics. The table, along with statistics, is automatically displayed by every execution of the recovery system. If you need this table, along with statistics, without actually performing the recovery, use the >CONTROL STATS command instead.

Syntax

```
>PRINT { DBTABLE FILETABLE }
```

Parameters

DBTABLE	displays names of databases specified for recovery
FILETABLE	displays file references, user references, *rmodes* and *fmodes* specified in >FILE commands.

Example

```
>PRINT DBTABLE

*****************************************************************
*                       DATABASE STATISTICS                     *
*                                                               *
*  NAME   GROUP   ACCOUNT   OPENS   TRANS   PUTS   DELETES   UPDATES  *
*  ----   -----   -------   -----   -----   ----   -------   -------  *
*  ORDERS  TST   MKG           0       0      0         0         0  *
*****************************************************************
```

Text Reference

Chapter 7

>RECOVER

Used to designate the name of a database to be roll-forward recovered; opens database root file, validates *logid* and password with MPE/iX, and checks the DBSTORE flag. Multiple databases can be roll-forward recovered concurrently if they have all logged to the same log file by entering the >RECOVER command once for each database or as follows:

>RECOVER *database name, database name*

Syntax

```
>RECOVER database name[/maint word][.group[.account]]
```

Parameters

database name is the name of the TurboIMAGE/XL database to be recovered.

maint word is the maintenance word defined by the database creator. This word must be supplied by anyone other than the database creator.

group is the group where the database(s) resides.

account is the account where the database(s) resides.

Discussion

If the >RECOVER command is accepted, the following message is returned:

```
DATABASE database name LAST DBSTORED day, date, time
```

The following conditions must be satisfied before the >RECOVER command is accepted:

1. The database must be accessible to the user (database administrator) running DBRECOV. This user must either be the creator of the database or know the maintenance word. If the database resides in a group or account different from the user's logon, the MPE/iX file security must permit the user read and write access to the database files.
2. The database must be enabled for recovery.
3. The log identifier characteristics (name, password, log file name and device type) must not have been altered since the log file was generated. This restriction applies to MPE/iX log commands as well as those provided for TurboIMAGE/XL by DBUTIL. This is necessary because the MPE/iX log identifier is used by TurboIMAGE/XL to obtain the name and device type of the log file.

   The >RECOVER command will not be accepted if the *logid* is unknown to MPE/iX. However, if the *logid* is known to MPE/iX but specifies the wrong log file, this condition is not detected at this time and >RECOVER will be accepted. DBRECOV will generate

erroneous data in the database if the database is recovered with the wrong log file.

4. The DBSTORE flag must be set, indicating that the database 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 database when >RECOVER is called. Exception: When the MODE4 option of the >CONTROL command is specified, the database can be concurrently accessed in mode 6 (read only).

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 THURS, SEP 7, 1989, 6:30 PM
DATABASE RETAIL LAST DBSTORED MON, SEP 11, 1989, 10:00 PM
```

ORDERS and RETAIL are *database names*.

Text Reference

Chapter 7

>ROLLBACK

Rolls out any incomplete transactions following a system crash. Multiple databases can be roll-back recovered concurrently by entering the command as follows:

>ROLLBACK *dbname, dbname*

Syntax

```
>ROLLBACK dbname[/maint word][.group[.account]]
```

Parameters

database name is the name of individual database(s) to be rolled-back.

maint word is the maintenance word defined by the database creator. This word must be supplied by anyone other than the database creator.

group is the group where the database(s) resides.

account is the account where the database(s) resides.

Discussion

If the >ROLLBACK command is accepted, the following message is returned:

```
DATABASE database name LAST USED day, date, time
```

The following conditions must be satisfied before the >ROLLBACK command is accepted:

1. The database must be accessible to the user (database administrator) running DBRECOV. This user must either be the creator of the database or know the maintenance word. If the database resides in a group or account different from the user's on, the MPE/iX file security must permit the user read and write access to the database files.
2. The database must have been enabled for roll-back recovery.
3. The log identifier characteristics (name, password, log file name and device type) must not have been altered since the log file was generated. This restriction applies to MPE/iX log commands as well as those provided by TurboIMAGE/XL by DBUTIL. This is necessary because the MPE/iX log identifier is used by TurboIMAGE/XL to obtain the name and device of the log file.
4. When roll-back is enabled, DBUTIL sets a roll-back flag to indicate that roll-back is enabled for the database. The roll-back time stamp is updated when the database 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 database.
5. No other users can be accessing the database when >ROLLBACK is called. The database

can be concurrently accessed by users when the >CONTROL command is specified with the MODE4 option.

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 >ROLLBACK:

```
>FILE
>PRINT
>CONTROL
```

The >FILE command optional parameter *rmode* is not used with >ROLLBACK.

The following >CONTROL options are not applicable with >ROLLBACK:

```
STAMP, NOSTAMP, STORE, NOSTORE, STOPTIME
```

Example

```
>CONTROL NOSTATS
>ROLLBACK ORDERS
DATABASE ORDERS LAST USED THURS, SEP 7, 1989, 6:00 PM
>RUN
```

ORDERS is the *database name*.

Text Reference

Chapter 7

>RUN

Initiates recovery-process. The recovery system opens the log file and validates the log identifier before roll-forward recovery or roll-back recovery begins.

Syntax

```
>RUN
```

Discussion

For recovery to succeed, the log file must be accessible to the database administrator. This means that the database administrator must either be the creator of the log identifier used to create the log file, or know the maintenance word and have system manager (SM) or operator (OP) capability. If the database administrator does not have system manager capability, and if the log file resides on disk in a group and account different from logon, then the administrator must have read access to the log file according to MPE/iX file security. File equations are permitted. However, the fully qualified file name of the expected log file must be specified. If the log file resides on tape, the database administrator must know the volume identifier, so that the operator can respond to the log file 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, and 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 may not have been 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, and DBUPDATE log records that were "rolled-out." An asterisk (*) will appear next to these processes.

A table of database statistics includes the same information totaled for each database. A logging system table includes the log identifier, log file information, and recovery file information if this facility is used. Refer to "Recovery Tables" in chapter 7 for more information on Process, Database, Logging and Recovery Tables.

Example 1

Roll-forward recovery of database ORDERS.

```
:RUN DBRECOV.PUB.SYS
>RECOVER ORDERS
DATABASE ORDERS LAST DBSTORED FRI, SEP 22, 1989, 4:00 PM
>RUN
```

Example 2

Roll-back recovery of database ORDERS.

```
:RUN DBRECOV.PUB.SYS
>ROLLBACK ORDERS
DATABASE ORDERS LAST USED THURS, SEP 21, 1989, 6:00 PM
>RUN
```

DBRESTOR

Copies a database to disk from the backup volume(s) created by the DBSTORE program or by the MPE/iX STORE command.

Operation

```
1       [ :FILE DBRESTOR [=filename] [;DEV=device]
          [;REC=recsize] [; {BUFNOBUF}                    ] ]
2       :RUN DBRESTOR.PUB.SYS
              .
              .
              .
3          WHICH DATABASE? database name [/maint word]
4          DATABASE RESTORED
                 END OF PROGRAM
```

Parameters

filename is a name (up to 8 characters) that replaces DBRESTOR in the mount prompt at the operator's console.

device is the device class name of the device from which the database is to be recorded.

recsize is the record size of the record to be restored. *recsize* must be at least as large as the record written to the device to avoid losing data.

database name is the name of a TurboIMAGE/XL database root file to be restored.

maint word is the maintenance word defined by the database creator. This word must be supplied by anyone other than the database creator.

Operation Discussion

1 An optional file equation that specifies the device class name for the device from which the database 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 in the PUB group of the SYS account.

3 In session mode, DBRESTOR prompts for the database name and maintenance word. In job mode, the database 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 sets and restored the data to these files, it prints a confirmation message.

Console Messages

After you supply the database name and DBRESTOR opens the file specified by *filename*, a message is displayed on the system console. A tape must be mounted on the appropriate unit and identified through an operator reply. Refer to *Volume Management Reference Manual* for instructions about console interaction.

If the database is on more than one volume, another message is displayed on the system console. The operator must mount the next volume in the sequence. If the volume that is mounted is not the correct format, the operator is notified through a console message. If the correct volume is available, the incorrect one should be removed and the correct one mounted. The operator must enter a reply on the console.

Example

```
:JOB MRG.ACCOUNTA          Initiate job.
:RUN DBRESTOR.PUB.SYS      Initiate DBRESTOR.
ORDERS/SELL                Specify database name and maintenance word.
:EOJ                       Terminate job.
```

After creating the files and restoring the file contents, DBRESTOR prints the following message on $STDLIST:

```
DATABASE RESTORED
```

NOTE If there is already a copy of the database on disc, it must be purged using DBUTIL before running DBRESTOR.

DBSTORE

Stores the database root file and all data sets to a tape in a format compatible with backup files created by the MPE STORE and SYSGEN commands. DBSTORE differs from these commands in that it handles only TurboIMAGE/XL databases.

Operation

1 [:FILE DBSTORE [=*filename*] [;DEV=*device*] [;REC=*recsize*] [;{ BUF NOBUF}]]

2
```
:RUN DBSTORE.PUB.SYS [;INFO="MPE STORE options"]
     .
     .
     .
```

3 `WHICH DATABASE? database name [/maint word]`

4
```
DATABASE STORED
END OF PROGRAM
```

If you try to store a database that needs recovery, DBSTORE will do the recovery before storing the database.

Before copying the files, DBSTORE gains semi-exclusive access to the referenced database; that is, DBSTORE determines that the only other database activity consists of other users executing DBSTORE or application programs that open the database in mode 6 or 8. If DBSTORE cannot gain semiexclusive access, it terminates and prints the following message:

```
DATABASE IN USE
```

You must be the database creator or provide the maintenance word to use DBSTORE.

Parameters

filename	is the name (up to 8 characters) that replaces DBSTORE in the mount request at the operator's console.
device	is the device class name of the device on which the data entries are to be stored.
recsize	is the record size of the record to be written to the device; must be a multiple of 512 bytes and less than the configured record size for the device.
database name	is the name of a TurboIMAGE/XL database to be stored.
maint word	is the maintenance word defined by the database creator. This word must be supplied by anyone other than the database creator.
INFO=	is used for the parameters that can be passed to the MPE STORE/RESTORE process. For example, the TRANSPORT

parameter (INFO="TRANSPORT") allows you to migrate files from MPE/iX to MPE V with the MPE/iX STORE command.

Operation Discussion

1 The optional file equation that specifies the device class name for the device on which the database 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 in the PUB group of the SYS account. The *MPE STORE options* are parameters that can be passed to the STORE/RESTORE process. For example, the TRANSPORT option is used when moving a TurboIMAGE/XL database to MPE V.

Note that the database may be too large to move to MPE V because, with the expanded file size available on MPE/iX, data sets can exceed the MPE V file size limit. If the database contains a data set larger than the MPE V limit, an MPE error is displayed.

3 In session mode, DBSTORE prompts for the database name and maintenance word. In job mode, the database 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 sets, it prints a message to signal completion.

NOTE DBSTORE does not store jumbo data sets or B-Tree index files. Instead, use the STORE command with POSIX names or use TurboSTORE/iX 7x24 True-Online Backup with ONLINE=START or ONLINE=END option.

Logging

DBSTORE updates a time stamp and store flag in the database root file before storing the database. The time stamp designates the date and time of the DBSTORE operation, and is used by DBRECOV to help identify the correspondence between log files and backup databases.

The store flag is set by DBSTORE to indicate that the database has been stored; this flag is cleared (reset) when the first modification to the database 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 ensure that no one has modified the backup database prior to recovery. DBUTIL checks this flag whenever logging and recovery is enabled, because a valid database backup 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:  database modified and not DBSTORED
```

This warning does not necessarily indicate that a valid backup does not exist, because

either an MPE SYSGEN or STORE command could have been used instead of DBSTORE. Because neither SYSGEN or STORE update the database time stamp 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 chapter 7 for further discussion of logging and recovery.

If the mirror database maintenance method is being used, storing the database on the secondary system can be done differently than using the DBSTORE process. When using DBRECOV STOP-RESTART recovery on the database, storing the database, RESTART file, and the log files that were processed since the last successful RESTART can be done with an MPE STORE command. DBRECOV STOP-RESTART places a time stamp in the RESTART file and in the database to identify which RESTART file to apply to which database. If naming conventions have been followed, an MPE STORE @ command can be used to store all the necessary files and database(s). If DBSTORE is used, the user must remember to use an MPE STORE command to store the RESTART file and the log files. For more information on DBRECOV STOP-RESTART, refer to "The Mirror Database" in chapter 7.

Console Messages

After you supply the database name and DBSTORE opens the output file, a message is displayed on the system console. A tape must be mounted on the appropriate unit and identified through an operator reply. Refer to the *Volume Management Reference Manual* for instructions about console interaction.

If more than one volume is required to store the database, a request is displayed on the console for the next one. The next tape must be mounted and the unit readied. The volume that has been removed should be properly labeled with the database name and volume number.

Example

```
:JOB MGR.ACCOUNTA       Initiate job.
:RUN DBSTORE.PUB.SYS    Initiate DBSTORE program.
ORDERS/SELL             Supply database name and maintenance word.
:EOJ                    Terminate job.
```

After copying the ORDERS root file and all data sets, DBSTORE prints the following message on $STDLIST:

```
DATABASE STORED
```

CAUTION If you need to cancel a DBSTORE, reply zero to the tape request:

:REPLY pin#,0

Do not use **Break** and ABORT to abort the process when the tape mount is requested. When DBSTORE is aborted by using **Break** and ABORT, the date-time stamp and store flag in the root file are updated even though the database was not stored.

TurboSTORE/iX 7x24 True-Online Backup

You can also use TurboSTORE/iX 7x24 True-Online Backup to back up your database. The advantages of using the option ONLINE=START or ONLINE=END are:

- You can store the database even when it is open for access.
- The related files including TC file, third-party indexing files, jumbo data set files, and B-Tree index files are also stored along with it.
- The DBSTORE flag and time stamp are also set before storing the database.
- You can restore it using the MPE RESTORE command.
- If the database is enabled for roll-forward logging, DBRECOV can recover the database.
- You can store multiple databases with one command.

For more information, refer to the *STORE and TurboSTORE/iX Products Manual*.

DBUNLOAD

Copies the data entries from each data set to specially formatted tape volumes.

Operation

1
```
[:FILE DBUNLOAD[=filename] [;DEV=device] ]
```

2
```
:RUN DBUNLOAD.PUB.SYS [ ,CHAINED
                        ,SERIAL    ]
```
```
.
.
.
```

3
```
WHICH DATA BASE? database name [/maint word]
```
```
.
.
.
```

4
```
DATA SET m: x ENTRIES EXPECTED, x ENTRIES PROCESSED
```

5
```
END OF VOLUME n, y WRITE ERRORS RECOVERED
SAVE VOLUME ON LOGICAL DEVICE z AS n
```

6
```
DATABASE UNLOADED

END OF PROGRAM
```

(Refer to "Operation Discussion" later in this section.)

Parameters

filename 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 *filename*. This causes a simulated unloading of the database, preventing the need to mount a tape.

device is the device class name of the device to which the data entries are to be copied.

database name is the name of the TurboIMAGE/XL database to be unloaded.

maint word is the maintenance word defined by the database creator. This word must be supplied by anyone other than the database creator.

Message Variables

m is the number of data sets in the database.

x is the number of entries (expected) and the number of entries processed or copied from the specified data set.

n is the number of the volume.

y is the number of write errors from which DBUNLOAD has successfully recovered.

z is the logical device number of the unit.

DBUNLOAD is necessary if you want to modify the database structure to, for example, increase the capacity of a data set. To increase a capacity,

1. Unload the entries.
2. Purge the database.
3. Change the schema and create a new root file.
4. Execute the DBUTIL >>CREATE command.
5. 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 database and, to read the complete entry, uses a *list* parameter of an at-sign followed by a semicolon (@;). 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 database.

DBUNLOAD requires exclusive access to the database. If the database is already open by any other process, DBUNLOAD prints the message:

```
DATABASE IN USE
```

and prompts again for a database name.

DBUNLOAD operates in either serial or chained mode as explained below. The mode is determined by the entry point specified with the RUN command; for example:

```
:RUN DBUNLOAD.PUB.SYS,CHAINED
```

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. (shown at the end of this section on DBUNLOAD) illustrates the method for unloading a data set in chained mode. After the database 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 follow the chain backward to the break, then unload the remaining records of the chain. 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 (see statement 4 under "Operation Discussion" in this section).

Operation Discussion

1 An optional file equation that 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 in the PUB group of the SYS account.

3 In session mode, DBUNLOAD prompts for the database name and maintenance word. In job mode, the database 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 that 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 m:  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 four message lines are repeated for every broken chain in the data set, followed by the end-of-data-set summary that reports the number of lost entries, if any:

```
DATA SET m: 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 n, y WRITE ERRORS RECOVERED
```

where *n* is the number of the volume and *y* 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 *z* is the logical device number of the tape drive and *n* is the volume number):

```
SAVE VOLUME ON LOGICAL DEVICE z AS n
MOUNT NEXT VOLUME ON LOGICAL DEVICE z.
```

6 After the data sets have been successfully copied, DBUNLOAD issues a completion message.

```
DATABASE UNLOADED
END OF PROGRAM
```

Console Messages

After you supply the database name and DBUNLOAD opens the output file, a message is displayed on the system console. A tape must be mounted on the appropriate unit and identified through an operator reply. Refer to the *Volume Management Reference Manual* for instructions about console interaction.

Using ControlY

When executing DBUNLOAD in session mode, you can press **Control**Y to request the approximate number of entries in the current data set that have already been written. DBUNLOAD then prints the following message on $STDLIST:

```
<CONTROL Y>DATA SET m: 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 *z* is the logical device number of the unit):

```
WRITE PROBLEMS TRY ANOTHER VOLUME ON LOGICAL DEVICE z
```

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 DATABASE? 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

DATABASE UNLOADED

END OF PROGRAM
```

Example (Job Mode)

```
:JOB MGR.ACCOUNTA       Initiate job.
:RUN DBUNLOAD.PUB.SYS   Initiate execution of DBUNLOAD.
ORDERS                  Specify database name.
:EOJ                    Initiate end of job.
```

Because the user in this example is the database 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:

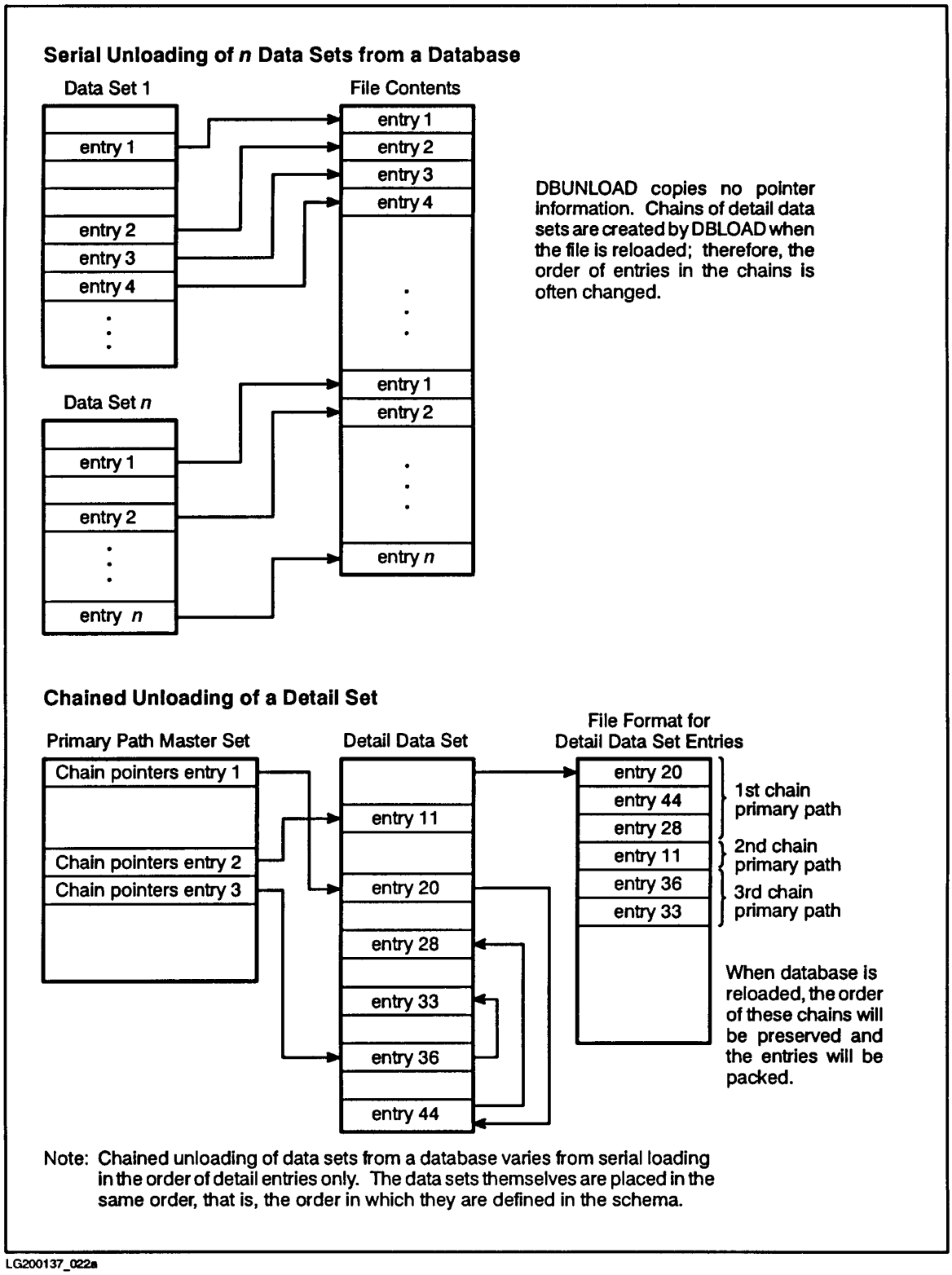
```
DATA SET 1:     50 ENTRIES EXPECTED, 50 ENTRIES PROCESSED.
DATA SET 2:      9 ENTRIES EXPECTED, 9 ENTRIES PROCESSED.
DATA SET 3:     24 ENTRIES EXPECTED, 24 ENTRIES PROCESSED.
DATA SET 4:     12 ENTRIES EXPECTED, 12 ENTRIES PROCESSED.
DATA SET 5:      5 ENTRIES EXPECTED, 5 ENTRIES PROCESSED.
DATA SET 6:      0 ENTRIES EXPECTED, 0 ENTRIES PROCESSED.

END OF VOLUME 1,0 WRITE ERRORS RECOVERED

DATABASE UNLOADED

END OF PROGRAM
```

Figure 8-1. DBUNLOAD File: Sequence of Entries



LG200137_022a

DBUTIL

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 in the PUB group of the SYS account.

2 Prompts for a DBUTIL >>*command*.

Enter one of the following:

HELP	VERIFY	ADDINDEX	EXIT
CREATE	SET	DROPINDEX	
ERASE	ENABLE	REBUILDINDEX	
MOVE	DISABLE	REDO	
PURGE	RELEASE	DO	
DEACTIVATE	SECURE	LISTREDO	
ACTIVATE	SHOW	DETACH	

DBUTIL commands can be abbreviated to the first three characters or less. For example, >>CREATE can be abbreviated to >>C or >>CRE. Enter the HELP command for the minimum abbreviation for each command.

When using the >>CREATE, >>PURGE, or >>ERASE command, 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/XL prompts you for the database name and, optionally, for the maintenance word, as follows:

```
Database name: database name [/maint word]
```

where:

database name is the name of a TurboIMAGE/XL database root file catalogued in the current session or job's account and logon group.

maint word is an optional ASCII string, one to eight characters long with no commas

or semicolons, that defines a password to be used by anyone other than the database creator to enable them to execute certain DBUTIL commands, and operate other utilities. (The database creator can also define or change the maintenance word by using the >>SET command).

In job mode, the database name and maintenance word, if any, must be in the record immediately following the RUN command. To perform any DBUTIL command except >>SHOW, >>HELP, or >>EXIT, you must have exclusive access to the database or database-access file.

>>ACTIVATE

Activates the database-access file for use with DBOPEN. Before using this command, read the description of remote database access in chapter 9.

This command should be used to prepare a database-access file before accessing a remote database residing on another HP 3000.

Syntax

```
>>A[CTIVATE] database-access file name
```

For example:

```
ACTIVATE ORDDBA
```

where ORDDBA is the *database-access file name*

Parameter

database-access file name is the name of the database-access file that you created with an editor.

The database-access file (created on the local system) can have any legal MPE/iX file name and is not dependent on the database name.

Unexpected Results

TurboIMAGE/XL checks that the following conditions are not violated:

- File code is 0.
- Record length does not exceed 128 characters.
- File is unnumbered.
- 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>
DSLINE 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) that would cause the file to be rejected.

Example

```
   :RUN DBUTIL.PUB.SYS Initiate DBUTIL execution.
            .
            .
            .
   >>ACT ORDDBA Enter abbreviated form of ACTIVATE command and
database-access file name.
   Verification follows
   FILE command:    Looks good
   DSLINE 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.

>>ADDINDEX

The ADDINDEX command updates the root file, and adds the associated B-Tree index file. When using the ALL option and there is no master dataset, a warning is generated, but the command is considered to be successful.

Syntax

```
>>ADDI[NDEX] database name[/maintword] FOR { ALL
                                             setnamelist
                                           setnumlist  }
```

For example:

```
>>ADDINDEX ORDERS FOR ALL
```

where ORDERS is the *database name*.

Parameters

database name	is the name of a TurboIMAGE database.
maintword	is the maintenance password.
setnamelist	is the list *setname*[,...]
setnumlist	is the list *setnum*[,...]
ALL	means all master data sets for the database.

Example

```
>>ADDINDEX  ORDERS/secret FOR ALL

  Found 4 master datasets.
     Adding index to set# 1 (#entries = 162,730, capacity = 218,987)
     Adding index to set# 2 (#entries =  84,164, capacity = 188,517)
     Adding index to set# 3 (#entries =  18,784, capacity =  21,943)
     Adding index to set# 4 (#entries =     783, capacity =    2583)
  Done.
```

If the data set is big and DBUTIL is interactive, a progress report at an interval of every 5% will be displayed.

```
5% done ...
10% done ...
```

It is always displayed on the same line. It will be overwritten by the next permanent line.

```
Adding index to ... or Done.
```

>>CREATE

Creates and initializes a file for each data set in the database.

Once the Schema Processor has created the root file, the database creator must build a file for each data set in the database using the >>CREATE command. DBUTIL initializes each data set to zeros and saves it as a catalogued MPE/iX file in the same logon group as the root file, on the device classes specified in the schema. The data set 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 named *XXXX*01, the second data set is named *XXXX*02, and so on. In order to save files for the maximum of 199 data sets per database, files are incremented from *XXXX*01-99, *XXXX*A0-A9, *XXXX*B0-B9, up to *XXXX*J9.

To execute the DBUTIL program to create and initialize the database, you must be the database 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 database files, it prints a confirmation message on the list file device and prompts for another command.

The CREATE command has been enhanced to create the required chunk control and chunk data files for jumbo data sets. To specify a jumbo data set, the JUMBO option must be included in the schema before defining a jumbo data set. Then any data set whose capacity is greater than 4GB automatically becomes a jumbo data set.

The CREATE command does an implicit ADDINDEX command for each dataset marked by DBSCHEMA as indexed.

Syntax

```
>>C[REATE] database name [/maint word]
```

For example:

```
CREATE ORDERS
```

where ORDERS is the *database name*.

Parameters

database name is the name of a TurboIMAGE/XL database being created.

maint word is the maintenance word that can be defined by the database creator when the database is created. To access the database, anyone other than the database creator must supply this word.

Example (Session Mode)

```
:RUN DBUTIL.PUB.SYS Initiate DBUTIL executions.
          .
          .
          .
>>CREATE ORDERS      Respond to DBUTIL prompt with >> CREATE command and database name.
Database ORDERS has been CREATED
>>
```

DBUTIL creates, initializes, and saves files named ORDERS01, ORDERS02, and so forth, one file for each data set. These constitute the empty database.

Example (Job Mode)

```
:JOB MGR.ACCOUNTA      Initiate job.
:RUN DBUTIL.PUB.SYS    Initiate DBUTIL execution.
CREATE ORDERS          Enter >>CREATE command and database name.
EXIT                   Terminate DBUTIL.
:EOJ                   Terminate job.
```

After the data files are created and initialized, DBUTIL prints the following message on the list file device:

```
DATABASE ORDERS HAS BEEN CREATED
```

NOTE >>CREATE will fail if the native language defined for the database is not supported at the system level. (Refer to appendix A or the *Native Language Support Programmer's Guide* for more information.)

>>DEACTIVATE

Deactivates the database-access file to allow modifications to the file or to disallow remote database access.

This command is used before you change the contents of the database-access file. (Refer to chapter 9 for more information about accessing remote databases.)

If DBUTIL successfully deactivates the file, it prints a confirmation message on the list file device.

Syntax

```
>>DE[ACTIVATE] database-access file name
```

For example:

```
DEACTIVATE ORDDBA
```

where ORDDBA is the *database-access file name*.

Parameter

database-access file name is the name of the database-access file to be deactivated.

Example

```
:RUN DBUTIL.PUB.SYS  Initiate DBUTIL execution.
     .
     .
     .
>>DEACTIVATE ORDDBA  Enter a >>DEACTIVATE command and the database-access file name.
DEACTIVATED
>>
```

>>DETACH

The DETACH command detaches the database from the DBEnvironment(s) the database is attached to. You can use one command of DBUTIL to detach a given database from all of the DBEnvironments to which it is attached; you do not have to specify each DBEnvironment name.

If you copy your database to a different account or different group, then issue a DETACH command for the copied database, you will get an error stating that the detach failed from *dbename.group.account* (ATCERR 32052). It is because the DETACH of the database triggers a lookup of the TC file (which contains the names of the DBE to which the database is attached). However, DBUTIL's subsequent verification of the original DBE shows the name of the original database (not the name of the copied database) as the one attached to this DBE. Due to this discrepancy, the following things happen:

- DBUTIL reports an error and does not perform a real detach of the copied database.
- The attached flag in the rootfile (copy) as well as the entry in the TC file is cleared, in spite of the error.
- The original production database still remains attached to the DBE.

The DETACH command updates the root file. Therefore, use caution when copying an attached database.

Syntax

```
>> DET[ACH]  database name [/maintword]
```

For example:

```
>>DETACH ORDERS
```

where ORDERS is the *database name*.

Parameters

database name	is the name of the database.
maint word	is the maintenance password.

Example

```
>>DETACH ORDERS/secret

Database has been detached from these HP SQL DBEnvironments:
    NEWDBE.BTRTESTS.IMAGESQL
    TEMDBE.BTRTESTS.IMAGESQL
>>
```

>>DISABLE

Disables the access, automatic deferred output, data prefetching, dumping, ILR, indexing, MUSTRECOVER, logging, and recovery options.

Syntax

```
>>DI[SABLE] database name[/maint word] FOR option[,option...]
```

For example:

```
DISABLE ORDERS FOR LOGGING,RECOVERY
```

where ORDERS is the *database name*, and LOGGING and RECOVERY are the *options*.

Parameters

database name is the name of a TurboIMAGE/XL database root file created in the current session or job's account and logon group.

maint word is the maintenance word defined by the database creator when the database is created with DBUTIL. This word must be supplied by anyone other than the database creator.

option is an option from the list provided and described below. More than one option can be specified.

Options

ACCESS disables user access to the database.

AUTODEFER disables automatic deferred output for the database. AUTODEFER must be disabled if ILR or roll-back recovery is to be enabled for a database.

DSEM disables the use of Dependency Semaphore employed to increase concurrency of modification intrinsics (DBPUT, DELETE, and DBUPDATE with CIU ON).

DUMPING disables the automatic dumping of the user's stack and the database control block in the event of a TurboIMAGE/XL abort. Unless requested by Hewlett-Packard support representatives, under most circumstances dumping should be disabled. When enabled, DUMPING creates files (before TurboIMAGE/XL aborts) that can prove helpful in determining the cause of such problems as a corrupted control block.

HWMPUT disables DBPUT action of placing entries at the high-water mark first, instead of at the delete chain head.

ILR disables Intrinsic Level Recovery facility.

INDEXING	disables third-party indexing (TPI) for the database. Third-party indexing provides the capability to do generic key searches, multiple keyword retrievals, and sorted sequential searches on any database. Refer to your vendor documentation for information on TPI.
LOGGING	disables the database roll-forward logging facility. Roll-back and MUSTRECOVER must be disabled first.
MUSTRECOVER	disables the MUSTRECOVER option for the database. Logging is not affected by disabling MUSTRECOVER. If the database needs recovery when you disable MUSTRECOVER, you are prompted to confirm the DISABLE command. If you respond to continue, the consistency of the database cannot be guaranteed. To ensure database consistency, respond with N, recover the database, and then disable MUSTRECOVER after recovering the database.
PREFETCH	disables the prefetching of data blocks required by the DBPUT and DBDELETE intrinsics under certain conditions. Refer to "Coordinating Additions to a Database" or "Coordinating Deletions from a Database" in chapter 4 for additional information.
RECOVERY	disables the database roll-forward recovery facility.
ROLLBACK	disables the database roll-back logging facility. However, logging will not be automatically disabled. To disable logging, use the >> *database name* FOR LOGGING command. Otherwise, logging (roll-forward) will remain enabled.

Default Conditions

```
Access is Enabled
Autodefer is Disabled
Dumping is Disabled
HWMPUT is Disabled
ILR is Disabled
Indexing is Disabled
Logging is Disabled
Mustrecover is Disabled
Prefetch is Disabled
Recovery is Disabled
Roll-Back is Disabled
```

Example

```
:RUN DBUTIL.PUB.SYS
     .
     .
     .
>>DISABLE ORDERS FOR ACCESS
Access is Disabled
>>
```

>>DROPINDEX

The DROPINDEX command drops the associated B-Tree index file and updates the root file. When using the ALL option and there is no master dataset, a warning is generated, but the command is considered to be successful.

Syntax

```
>>DROPI[NDEX] database name[/maintword] FOR { ALL
                                              setnamelist
                                            setnumlist  }
```

For example:

```
>>DROPINDEX ORDERS FOR 2
```

where ORDERS is the *database name* and 2 is set# 2.

Parameters

database name is the name of a TurboIMAGE/XL database.

maintword is the maintenance password.

setnamelist is the name of the set list.

setnumlist is the number of the set list.

ALL means all master data sets for the database.

Example

```
>>DROPINDEX ORDERS/secret FOR 2

   Dropping index from set# 2 (#entries = 84,164, capacity = 188,517)
Done.
```

>>ENABLE

Enables the access, automatic deferred output, data prefetching, dumping, ILR, indexing, MUSTRECOVER, logging, and recovery options.

Syntax

```
>>EN[ABLE] database name[/maint word] FOR option[,option...]
```

For example:

```
ENABLE RETAIL FOR LOGGING
```

where RETAIL is the *database name* and LOGGING is an *option*.

Parameters

database name is the name of a TurboIMAGE/XL database being enabled.

maint word is the maintenance word defined by the database creator when the database is created with DBUTIL. This word must be supplied by anyone other than the database creator.

option is an option from the list provided and described below. More than one option can be specified.

Options

ACCESS enables user access to the database.

AUTODEFER enables automatic deferred output for the database. With deferred output the MPE/iX transaction manager is not used to log database modifications to the transaction manager log file. Instead, AUTODEFER uses the MPE/iX file system default mode. This mode keeps data pages in memory for as long as possible until either lack of memory or the closing of a file causes the pages to be written to disk. In this mode a system failure can cause the loss of database integrity. ILR is not compatible with AUTODEFER; therefore, deferred output should be used only in a batch situation where the database has been backed up prior to batch processing. ILR must be disabled prior to enabling AUTODEFER.

AUTODEFER can be used to increase I/O performance by disabling Transaction Management (XM). However, ILR and roll-back recovery must be disabled. You must consider performance and the ability to recover data when determining whether to use AUTODEFER. Roll-forward logging can be used to preserve consistency.

DSEM enables use of Dependency Semaphore for increased concurrency of

modification intrinsics (DBPUT, DELETE, and DBUPDATE with CIU ON).

DUMPING	is an option for Hewlett-Packard support use, development, and debugging only. When enabled, the TurboIMAGE/XL abort procedure copies the user's stack and the database control blocks to files if a TurboIMAGE/XL procedure aborts.
HWMPUT	enables DBPUT action of placing entries at the high-water mark first, instead of at the delete chain head.
ILR	enables the Intrinsic Level Recovery facility. TurboIMAGE/XL maintains structural integrity without ILR enabled.
INDEXING	enables third-party indexing (TPI) for the database if not already done by the third-party software when configuring the database for TPI. Third-party indexing provides the capability to do generic key searches, multiple keyword retrievals, and sorted sequential searches on any database. Refer to your vendor documentation for information on TPI.
LOGGING	enables the database roll-forward logging facility.
MUSTRECOVER	enables the MUSTRECOVER option for the database. If logging is not already enabled, it is automatically enabled when MUSTRECOVER is enabled. While MUSTRECOVER is enabled, the database cannot be accessed after a system failure until the database is recovered with roll-forward or roll-back recovery.
PREFETCH	enables the prefetching of data blocks required by the DBPUT and DBDELETE intrinsics under certain conditions. Refer to "Coordinating Additions to a Database" or "Coordinating Deletions from a Database" in chapter 4 for additional information.
RECOVERY	enables the database for recovery.
ROLLBACK	enables the database roll-back logging facility. A warning displays if the log file does not exist, and the database remains disabled for roll-back recovery. If logging is not in effect already, it will be enabled automatically.

Default Conditions

```
Access is Enabled
Autodefer is Disabled
Dumping is Disabled
HWMPUT is Disabled
ILR is Disabled
Indexing is Disabled
Logging is Disabled
Mustrecover is Disabled
Prefetch is Disabled
Recovery is Disabled
Roll-Back is Disabled
```

Example

```
:RUN DBUTIL.PUB.SYS
     .
     .
     .
>>ENABLE ORDERS FOR RECOVERY
Recovery is Enabled
>>
```

>>ERASE

Reinitializes all data sets in the database to their empty condition and resets all flags except the access, PREFETCH, and recovery flags (refer to the discussion of the DBUTIL >>ENABLE command earlier in this chapter).

The data sets remain as catalogued MPE/iX files. To execute DBUTIL to reinitialize the data sets, you must be the database creator or supply the correct maintenance word. This utility function should be performed before data that was saved by DBUNLOAD is loaded back into the database unless it was re-created.

After DBUTIL has completely reinitialized the data sets, it prints a confirmation message on the list file device.

NOTE The ERASE command erases any associated B-Tree index (.idx) files, but will not delete them.

Syntax

```
>>ER[ASE] database name [/maint word]
```

For example:

```
ERASE ORDERS/SELL
```

where ORDERS is the *database name* and SELL is the *maint word*.

Parameters

database name	is the name of a TurboIMAGE/XL database being erased.
maint word	is the maintenance word defined by the database creator when the database is created with DBUTIL. This word must be supplied by anyone other than the database creator.

Example

```
:RUN DBUTIL.PUB.SYS      Initiate DBUTIL execution.
     .
     .
     .
>>ERASE ORDERS/SELL      Enter >>ERASE command, database name, and
                         maintenance word.
Database ORDERS has been ERASED
>>
```

DBUTIL reinitializes all the data sets in the ORDERS database to binary zeroes. With the exception of the access, PREFETCH, and recovery flags, the database flags are reset to their default conditions. The logging and MUSTRECOVER options are disabled if they were previously enabled.

NOTE The execution of utilities is not logged. If you use DBUTIL to erase the database, the >>ERASE command automatically disables logging, ILR, third-party indexing, MUSTRECOVER, and roll-back recovery.

>>EXIT

Terminates DBUTIL execution.

Syntax

```
>>E[XIT]
```

Example

```
>>CREATE ORDERS      Create a database.
Database ORDERS has been CREATED

>>EXIT               If no other DBUTIL functions are to be performed,
                    terminate DBUTIL with >>EXIT command.
END OF PROGRAM
```

>>HELP

Displays each of the DBUTIL commands.

```
>>H[ELP] [commandname]
```

Parameter

commandname is the name of a specific DBUTIL command whose format you want to display. The name can be abbreviated to the minimum command abbreviation permitted by DBUTIL.

If you do not specify a command, the >>HELP command lists the names of all valid DBUTIL commands.

If you specify a command, the correct syntax for that command is displayed.

Example

```
>>HELP
Commands are:

ACTIVATE     ADDINDEX     CREATE       DEACTIVATE    DETACH
DISABLE      DROPINDEX    ENABLE       ERASE         EXIT
HELP         MOVE         PURGE        REBUILDINDEX  RELEASE
SECURE       SET          SHOW         VERIFY        REDO

Commands may be abbreviated.
For help on a particular command type:  'HELP command name'.

>>HELP CREATE

C[REATE] database name [/maint word]

>>
```

>>MOVE

Moves TurboIMAGE/XL files across devices within the same volume set.

Syntax

```
>>M[OVE] TurboIMAGE/XL file name TO device
```

For example:

```
MOVE ORDERS05 to DISC2
```

where ORDERS05 is the *file name* and DISC2 is a *device*.

Parameters

file name	is a TurboIMAGE/XL root file, data set, or ILR file. Enter the file name only; no group/account specification is allowed. The user must be the creator of the file.
device	is the class name of the MPE/iX device or the number of the logical device to which the TurboIMAGE/XL file should be moved. The device must be a member of the volume set on which the database resides.

Discussion

It is recommended that you store the database with the DBSTORE command prior to executing a MOVE command. This precaution is advisable in the event a system failure occurs during the move operation. When a move has been initialized, the process checks the root file flag to determine if the database has been modified since the last backup copy was made. The program prompts the user to continue or to terminate the MOVE command and proceed with a DBSTORE of the database before moving TurboIMAGE/XL files to another device. If the user responds NO to the continue prompt, the following message is printed on the terminal:

```
MOVE operation stopped.
```

The following steps outline the process involved in moving TurboIMAGE/XL files from one device to another. The move process does the following:

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 database 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.

For jumbo datasets, the MOVE command enables you to move either the chunk control file or a specific chunk data file to a different device.

The MOVE command does *not* allow a B-Tree index (".idx") file to be moved.

Example

```
>>MOVE ORDERS05 to 3
  Database last stored on FRI, SEP 22, 1989, 8:32 PM
  Database has been modified since last store date.
  The database 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 "ORDERS05"
  New copy of file "ORDERS05" saved as a permanent file
  File "ORDERS05" moved to device 3
```

The data set ORDERS05 has been moved to logical device number 3. This file is the INVENTORY data set and was originally assigned device class named DISC2 in the schema.

To obtain a listing of where all the data sets for the database reside, do a

```
>>SHOW ORDERS DEVICE
```

>>PURGE

Purges the root file and all the data sets of the referenced database. If you use third-party indexing, the >>PURGE command also purges any existing third-party index files regardless of third-party indexing being enabled or disabled.

Purging removes the files from the catalog and returns the disk space to the system. As with >>ERASE, you must be the database creator or must provide the maintenance word to use DBUTIL with the >>PURGE entry. Before running the DBRESTOR program to restore a database, use this utility function to purge the database.

If DBUTIL successfully purges the database, it prints a confirmation message on the list file device.

NOTE The PURGE command purges any associated B-Tree index (.idx) files.

Syntax

```
>>P[URGE] database name [/maint word] [DETACH]
```

For example:

```
PURGE ORDERS/SELL
```

where ORDERS is the *database name* and SELL is the *maint word*.

Parameters

database name	is the name of a TurboIMAGE/XL database being purged.
maint word	is the maintenance word defined by the database creator when the database is created with DBUTIL. This word must be supplied by anyone except the database creator when using DBUTIL to access the database.
DETACH	is an option specific to the IMAGE/SQL users. This option detaches the database from all ALLBASE/SQL database environments (DBEnvironments) to which it is attached via IMAGESQL. When DETACH is not used, the database is purged without detaching from the DBEnvironment(s). For more information, refer to the *IMAGE/SQL Administration Guide*.

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 data set, if any.
`Data set XXXXk has been purged`	DBUTIL successfully purged the root file and the *n* data sets of the database. 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 database *(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 database.
`Incomplete purge`	An error occurred while DBUTIL was attempting to purge the database and any Third-party indexing files. The specific error message is printed above this one. Some of the data sets and, if applicable, index files have been purged.
`Detach failed from DBEname`	This message is returned if you are using IMAGE/SQL. DBUTIL was unable to detach the TurboIMAGE/XL database from the ALLBASE/SQL database environment (DBEnvironment) to which it was attached. However, DBUTIL will continue executing the >>PURGE command. For more information, refer to the *IMAGE/SQL Administration Guide*.

Example

```
:RUN DBUTIL.PUB.SYS     Initiate DBUTIL execution.
      .
      .
      .
>>PURGE ORDERS          Enter >>PURGE command and database name assuming there is
                          no maintenance word.
```

The next messages are returned if you are using IMAGE/SQL, and the DETACH option is used with the >>PURGE command.

```
Database has been detached from these HP SQL DBEnvironments:
    DBEname.group.account
    DBEname.group.account

Database has been PURGED
```

DBUTIL confirms that the user is logged on with the same user name, account, and group which were used to create the database. 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 sets with file names based on the root file name are purged.

>>REBUILDINDEX

REBUILDINDEX rebuilds the B-Tree index file for a specified dataset that should have an index file.

When using the ALL option and there is no master dataset, a warning is generated, but the command is considered to be successful.

The KSAM file built by REBUILDINDEX has the Native Language Support language specified to match the language of the database, if the key is a text (X or U) data type.

Syntax

```
>REBUILDI[NDEX] database name[/maintword] FOR { ALL
                                                setnamelist
                                              setnumlist   455}
```

For example:

```
REBUILDINDEX ORDERS FOR 3
```

where ORDERS is the *database name*, and 3 is set# 3.

Parameters

database name	is the name of a TurboIMAGE/XL database.
maintword	is the maintenance password.
ALL	means all master data sets for the database.
setnamelist	is the name of the set list.
setnumlist	is the number of the set list.

Example 1

```
>>REBUILDINDEX ORDERS/secret FOR SalesrepName,Region,District
```

Example 2

```
>>REBUILDINDEX ORDERS/secret FOR 3
   Rebuilding index for set# 3 (#entries = 18,784, capacity = 21,943)
Done.
```

>>REDO

Redo the previous command or the previous *n*th command. Redo follows exactly the MPE/iX REDO syntax. REDO, LISTREDO, and DO work for DBUTIL the same was as the MPE/iX commands.

Syntax

```
>>REDO [n]
```

For example:

```
>>REDO 8
```

where 8 is the eighth previous command.

Parameter

n is the backwards count of the command being repeated.

>>RELEASE

Suspends file system security provisions for the database root file and data sets, allowing access to the database from other groups and accounts. If you use third-party indexing, the >>RELEASE command suspends the file system security provisions for any existing index files.

Syntax

```
>>R[ELEASE] database name
```

For example:

```
RELEASE ORDERS
```

where ORDERS is the *database name*.

Parameter

database name	is the name of a TurboIMAGE/XL database.

Discussion

The >>RELEASE command suspends file system security provisions for all of the database files at the file, group, and account levels, but leaves TurboIMAGE/XL security and MPE/iX privileged file security intact. Releasing the file system security allows the database to be accessed by users from other groups and accounts, without relinquishing the privacy of all other files in the database group. Only the creator of the database can release security. In addition, the group's home volume set must be mounted.

The database file security remains suspended until the creator issues a >>SECURE command. Suspension remains valid after job termination, or system failure followed by a system boot.

The RELEASE command applies to the associated B-Tree index (.idx) files.

>>SECURE

Restores security provisions that were released by a >>RELEASE command for the database root file and data sets. If you use third-party indexing, the >>SECURE command restores the security provisions for any existing index files.

Syntax

```
>>SE[CURE] database name
```

For example:

```
>>SECURE ORDERS
```

where ORDERS is the *database name*.

Parameter

database name	is the name of a TurboIMAGE/XL database being secured.

Discussion

The >>SECURE command reinstates the file system security provisions for the entire database. These security provisions can only be suspended by the >>RELEASE command. Only the creator of the database can successfully issue the >>SECURE command. In addition, the group's home volume set must be mounted.

The SECURE command applies to the associated ".idx" files.

>>SET

Changes or removes the maintenance word; only the database creator can change or remove the maintenance word. The >>SET command also sets the log identifier into the root file, modifies access class passwords, sets a subsystem flag, and sets the critical item update (CIUPDATE) option. For databases that will be migrated to MPE V, the >>SET command specifies the number of input/output buffers to be allocated by TurboIMAGE in the Database Buffer Area Control Block (DBB) depending on the number of users concurrently accessing the database.

Syntax

```
>>SET database name [/maint word]{BUFFSPECS=num buffers (from-users/to-users)
                                   [,num buffers(from-users/to-users)] ...
                                 LOGID=log identifier
                                 MAINT=maintenance word
                                 SUBSYSTEMS={NONE
                                             READ
                                             RW    }
                                 PASSWORD classnum=[password]
                                 LANGUAGE=language id
                                 CIUPDATE=(DISALLOWED
                                           ALLOWED
                                           ON         }
                                 BTREEMODE1={ON
                                             OFF}[,[WILDCARD=]c]            }
```

For example:

```
SET ORDERS MAINT=SELL
```

or

```
SET ORDERS/SELL CIUPDATE=DISALLOWED
```

where ORDERS is the *database name* and SELL is the *maintenance word*.

Parameters

database name	is the name of a TurboIMAGE/XL database root file catalogued in the current session or job's account and logon group.
maint word	is the current maintenance word for the database, and must be given by anyone using DBUTIL to access the database other than the database creator.
BUFFSPECS	is for MPE V compatibility only, because the TurboIMAGE/XL buffer specifications are fixed. For

databases that will be migrated to MPE V, it sets the number of buffers to be allocated by TurboIMAGE in the Database Buffer Area `Control Block (DBB). Refer to "Moving from MPE/iX to MPE V" in appendix H for a discussion of BUFFSPECS and a description of its parameters.

LOGID		sets the MPE/iX log identifier. The *log identifier* is obtained using the MPE/iX GETLOG command. Note that DBUTIL prompts for the *logid* password specified in the GETLOG command *before* it checks the validity of the log identifier. Entry of the correct *logid* password causes the valid log identifier to be stored in the root file and used whenever the logging capability is enabled. However, if the log identifier is left blank, it is removed from the database.
MAINT		sets the maintenance word for the database. The *maintenance word* is the new maintenance word for the database. If omitted, the currently defined maintenance word is removed and the database has no maintenance word. Only the database creator can change or remove the maintenance word.
SUBSYSTEMS		sets subsystem access to the database. The following options are valid:
	NONE	is the option used to prohibit use of any subsystem (for example, QUERY) on TurboIMAGE/XL.
	READ	is the option that allows only read access to the database by subsystems. The subsystem checks the root file flag to determine what access a subsystem is allowed.
	RW	is the option that allows read/write access to the database by subsystems. The subsystem checks the root file flag to determine what access a subsystem is allowed.
PASSWORD		sets the password. The following parameters are used with the *PASSWORD* parameter.
	classnum	is the access class whose password is being changed. It can be a number from 1 to 63, inclusive.
	password	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 database creator.)
LANGUAGE		sets the native language for the database. The following parameter is used with the *LANGUAGE* parameter:
	language id	is the number that identifies the native language. Refer to the *Native Language Support Programmer's Guide* for name and number information. The message "Language changed" appears after using the >>SET command to

change the language ID. This command can be issued only on a virgin root file or an empty database.

NOTE When reloading the database, the language must match the language ID stored in the backup media. A DBLOAD issued in a job fails if the language of the media differs from the database language. A DBLOAD in session mode provides a prompt to allow you to complete the DBLOAD operation when you reply **Y**.

CIUPDATE sets critical item update for the database. The following option settings are valid:

- DISALLOWED prevents any process from using the critical item update option on this database.
- ALLOWED indicates that programmatic enabling of the option is possible through a call to DBCONTROL mode 5, but programs that do not make this call are prevented from using critical item update on this database. Programs that enable the option do so temporarily for the duration of the process but can subsequently disable it through a call to DBCONTROL mode 6.

NOTE ALLOWED is now the default setting. It was DISALLOWED in releases prior to C.07.00.

- ON allows all processes to use the critical item update option on this database without the need to call DBCONTROL mode 5. Any process can explicitly disable the option temporarily for the duration of the process through a call to DBCONTROL mode 6 but can subsequently enable it through a call to DBCONTROL mode 5. This setting allows the critical item update option to be disabled in selected programs while enabling it for the majority.

  Critical item update is useful for those processes that need to update the values of detail data set search or sort items; master data set key items cannot be updated regardless of the CIUPDATE setting.

BTREEMODE1 sets the root file flag for B-Trees to ON or OFF. If it is ON, then a DBFIND mode 1 on ASCII types having a B-Tree index (explicit or implicit) and having the wildcard character in the argument will be treated as a B-Tree search. If set to OFF, then a DBFIND mode 1 will not be treated as a B-Tree search, even if the item has a B-Tree. See chapter 11, " B-Tree Indices," for more detailed information on BTREEMODE1.

WILDCARD is set with any printable ASCII character for *c*. When

doing a B-Tree DBFIND, mode-1-style arguments are scanned to find the first occurrence of the current wildcard character in the argument text. If the wildcard is not found, a non- B-Tree search is done (even if the DBFIND mode was 21). If the wildcard is found, the rest of the argument text is ignored.

NOTE This does *not* match the functionality of "@" in commands such as the MPE LISTF, but does match the functionality of current TPI implementations.

If the wildcard character is found at character k+1 (1-based), then the qualified entries will consist of all keys that match the argument in character positions 1..k.

When not doing a B-Tree find (even when mode=1 and BTREEMODE1=ON; or mode=10), the entire argument, including any wildcard characters, will be treated as the actual argument for a DBFIND mode 1.

The length of the argument may not exceed the item length.

Example 1

```
:RUN DBUTIL.PUB.SYS          Initiate DBUTIL execution.
   .
   .
   .
>>SET ORDERS MAINT           Remove current maintenance word.
Maintenance word changed
>>
```

Example 2

```
:RUN DBUTIL.PUB.SYS          Initiate DBUTIL execution.
   .
   .
   .
>>SET ORDERS CIUPDATE=ON     Indicates that processes can  update the values of
                             detail data set search or sort items in the ORDERS
                             database without a need to do DBCONTROL mode 5.
CIUPDATE is ON.              DBUTIL confirms the setting.
```

Example 3

```
:RUN DBUTIL.PUB.SYS Initiate DBUTIL.
   .
   .
   .
>>SET ORDERS BTREEMODE1 = ON,% Sets BTREEMODE1 option on for B-Tree search for
                               DBFIND mode 1 on X or U type having a B-Tree
                               index (explicit or implicit) and in the presence
                               of a wildcard character in the argument. Sets
                               the wildcard character as % in the ORDERS
                               database.
```

>>SHOW

Displays information about the database on a terminal or line printer. This can include a list of processes that have the database open, the status of locks in the database, the log identifier and flags, the current buffer specifications, and the setting for the critical item update option. Displays the capacity expansion information for the database and the data sets. If you are using IMAGE/SQL,

displays the names of any ALLBASE/SQL database environments (DBEnvironments) to which the database is attached. This command should be used with care because it obtains exclusive control of the database for several seconds preventing all other access.

Syntax

```
>>SH[OW] dbname [.group[.account]][/maint word] {ALL
                                                BUFFSPECS
                                                CAPACITY
                                                CIUPDATE
                                                DEVICE
                                                FLAGS
                                                INDEX
                                                INDEXES
                                                INDICES
                                                LANGUAGE
                                                LOCKS
                                                LOGID
                                                LOGINFO
                                                MAINT
                                                PASSWORDS
                                                SUBSYSTEMS
                                                USERS      } [OFFLINE]
```

For example:

```
SHOW ORDERS/SELL ALL OFFLINE
```

where ORDERS is the *database name* and SELL is the *maint word*.

Parameters

dbname is the name of a TurboIMAGE/XL database root file catalogued in the current session or job's account and logon group. If you have account manager (AM) capability, you can qualify the database with the *group* name. If you have system manager (SM) capability, you can qualify the database with the *group* and *account* name.

group is the group where the database resides. To qualify the database name by group, you must have AM or SM capability. If you have AM capability and want to qualify the database name by group, the database must have a maintenance word as you will be required to supply one.

account is the account where the database resides. To qualify the database name

by account, you must have SM capability.

maint word is the current maintenance word for the database. The database creator or the user with SM capability can omit the maintenance word.

ALL displays all the information provided with MAINT, BUFFSPECS, LANGUAGE, LOCKS, USERS, LOGID, SUBSYSTEMS, FLAGS, and the last-stored date. Displays number of detail data sets using dynamic capacity expansion and if dynamic capacity expansion is used for master data sets. Also, if the database is attached to any ALLBASE/SQL database environments (DBEnvironments) via IMAGESQL, the DBEnvironment names are displayed (refer to the *IMAGE/SQL Administration Guide* for information).

BUFFSPECS displays the current buffer specifications which can be either the TurboIMAGE/V default setting or the values specified with the DBUTIL >>SET command for those databases that will be migrated to MPE V; the TurboIMAGE/XL default setting is not displayed. Refer to the discussion of BUFFSPECS in the earlier section on the >>SET command and also in "Moving from MPE/iX to MPE V" in appendix H.

CAPACITY displays, for all data sets, capacity information including dynamic capacity fields. Information includes: data set name, type, number of entries, entries as a percentage of the maximum capacity for the data set, maximum capacity, current capacity, initial capacity, incremental number of entries and whether or not dynamic capacity expansion is used.

CIUPDATE displays the setting for the critical item update option. Valid settings are DISALLOWED, ALLOWED, and ON.

DEVICE displays the TurboIMAGE/XL root and data sets and where files reside (device class name or logical number) for a database.

FLAGS displays the state (enabled or disabled) of the logging, roll-back, ILR, recovery, restart, subsystem access, AUTODEFER, access, dumping, HWMPUT, PREFETCH, MUSTRECOVER, and third-party indexing (TPI) options. If MUSTRECOVER is enabled, the flags option also displays a message if the database needs recovery. In addition, it displays the last database store date, and information on whether the database has been modified since the last-stored date (DBSTORE flag).

INDEX INDEXES INDICES displays the B-Tree indices for the database.

LANGUAGE displays state of the native language declaration for the database.

LOCKS displays the status of locks currently obtained (or requested).

LOGID displays the current MPE/iX log identifier for the database.

LOGINFO displays information about all types of logging available with TurboIMAGE/XL.

MAINT displays the maintenance word, if any.

PASSWORDS displays the access class numbers from 1 through 63 together with the passwords assigned to them. (You must be the database creator.)

SUBSYSTEMS indicates whether subsystems, including user programs, can access the TurboIMAGE/XL database and, if access is allowed, whether it is read only or both read and write. Subsystem access is enforced by the subsystem.

USERS displays a list of the processes that have the database 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.)

The >>SHOW *database name* USERS command can be executed at any time. The remaining >>SHOW commands can be executed at any time except when another process has the database 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     INVENTRY.IMAGE.DATAMGT     #S116   1
22    1     BROWSE.IMAGE.DATAMGT       #S118   5
28    1     BROWSE.IMAGE.DATAMGT       #S112   5
29    1     INVENTRY.IMAGE.DATAMGT     #S115   1
31    1     ORDENTRY.IMAGE.DATAMGT     #S117   1
```

Example Discussion

The columns of information are described 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 ORDERS database.

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 chapter 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 database is open.

DBUTIL does not call DBOPEN so it is not listed as an executing program.

Example (Show All)

```
      .
      .
      .
>>SHOW ORDERS ALL Display all information for ORDERS database.
For database ORDERS

MAINTENANCE WORD:  SELL

Access is enabled.
Autodefer is disabled.
Dumping is disabled.
Rollback recovery is disabled.
Recovery is enabled.
ILR is disabled.
Mustrecover is enabled.
Logging is enabled.
Prefetch is disabled.
Indexing is disabled.
HWMPUT is disabled.
Restart is disabled.
Database last stored using True-Online Backup and
   logfile NLOG001  on WED, MAR 26, 1997,  8:07 AM.
Database has been modified since last store date.
Shadowing is disabled.
Subsystem access is READ/WRITE.

CIUPDATE is allowed.
Dynamic capacity expansion is not used.
Database has at least one indexed dataset.
BTREEMODE1 is off, wildcard = "@".

Logid:   NLOGID   is valid.
         password is correct.

XM log set      :  default XM user log set
                   for volume set MPEXL_SYSTEM_VOLUME_SET
XM log set type :  circular
XM log set size :  32 megabytes

The language is  0 - NATIVE-3000.

Buffer specifications:
50(1/120)

No other users are accessing the database.

>>
```

where *volname* is the name of the volume set in which the database resides.

Example Discussion

The listing above indicates that the current buffer specifications provide for 50 buffers to be allocated when there are between 1 and 120 concurrent users of the database. On TurboIMAGE/XL the buffer specifications remain fixed, so the information shown in the

above listing is useful only for databases that will be migrated to an MPE V system. The list above also shows that the database is enabled for roll-forward recovery, logging, MUSTRECOVER, and user access. It shows that the database was backed up using TurboSTORE/iX 7x24 True-Online Backup (with ONLINE=START or ONLINE=END option), and the logfile in use at the time was NLOG001. In the example, PREFETCH and third-party indexing (TPI) are disabled, critical item update (CIUPDATE) is allowed, and the restart flag is disabled. The restart flag is set by DBRECOV when the user has requested to suspend recovery between log files. The *logid* is shown, the password is verified, and the maintenance word is displayed. The messages that appear during the SHOW command can vary depending on what information is available on the database. If the maintenance word and *logid* are not present, the following messages display:

```
Logid is not present.

Maintenance word is not present.
```

The message regarding the transaction manager (XM) log set will also vary. The following message is printed if the database is not attached to transaction manager:

```
XM log set:  this database is not attached to an XM log set
```

The following message is printed if the database does not support NLS:

```
This database has been created before support of Native Languages.
```

If an error has occurred during dynamic roll-back recovery, the following message is displayed:

```
Database is logically inconsistent.
```

The following message is printed if the database is attached to an ALLBASE/SQL database environment (DBEnvironment) via the IMAGESQL utility:

```
Attached to these HP SQL DBEnvironments:
    DBEname
    DBEname
```

The following message is printed if the third-party indices are registered in the DBEnvironments.

```
Third Party Indexes are registered in these HPSQL DBEnvironments:

    DBEname
    DBEname
```

Refer to the *IMAGE/SQL Administration Guide* for more information.

The example displays, "Dynamic capacity expansion is used for 2 detail sets." Enable this feature for *new* databases by using the capacity parameters for detail data sets. See chapter 3 for more information. Enable this feature for *existing* databases by using DBChange Plus or a third-party utility. Refer to the *MPE/iX Release 5.0 Communicator* for information on using DBChange Plus.

Example (Show Capacity)

```
    .
    .
    .
>>SHOW ORDERS CAPACITY

                      No. of  %Max  --------------Capacity-------------- Dyn
Data Set Name  Type   Entries  Cap  Maximum  Current  Initial  Increment Exp
                                                   (Hashing)

CUSTOMER         M    20       10   200      200      200      0         NO
DATE-MASTER      A    111      53   211      211      200      0         NO
PRODUCT          M    15       5    300      300      200      0         NO
SALES            D    100      2    5012     140      140      70        YES
SUP-MASTER       M    60       3    2000     600      400      200       YES

INVENTORY        D    4998     1    500000   5000     5000     10000     YES
>>
```

The example shows Maximum, Current, Initial and Increment Capacities. The "%Max Cap" column shows how full the data set is as a percent of the maximum capacity for that set. "Dyn Exp" column shows whether dynamic capacity expansion is enabled or not. Enable this feature for *new* databases by using the capacity parameters when defining data sets. See chapter 3 for more information. Enable this feature for *existing* databases by using DBChange Plus or a third-party utility which supports this feature.

Format of Show Device List

The following example lists the TurboIMAGE/XL files for the ORDERS database, along with the data set names and the device where each resides. In the following example, the root file ORDERS and the data sets are shown. The file devices are listed as specified with the device class names. The ORDERS05 data set was moved using the DBUTIL >>MOVE command to logical device number 3; the device class name is displayed.

Example (Show Device)

```
>>SHOW ORDERS DEVICE
For database ORDERS

Volume set:  volname

MPE/iX  File Name          Data Set Name        Device

ORDERS.IMAGE.DATAMGT                            DISC
ORDERS01.IMAGE.DATAMGT     Date-Master          DISC1
```

```
ORDERS02.IMAGE.DATAMGT     Customer            DISC1
ORDERS03.IMAGE.DATAMGT     Product             DISC1
ORDERS04.IMAGE.DATAMGT     Sup-Master          DISC2
ORDERS05.IMAGE.DATAMGT     Inventory           DISC3
ORDERS06.IMAGE.DATAMGT     Sales               DISC2

>>
```

where *volname* is the name of the volume set in which the database resides.

Format of Show Indices

The following example lists the data set names, type, and whether they are indexed.

Example (Show Indices)

```
>>SHOW ORDERS INDICES
For database ORDERS

Data Set Name Type Indexed?

DATE-MASTER       A     YES
CUSTOMER          M     YES
PRODUCT           M     YES
SUP-MASTER        M     YES

4 indexed datasets

>>
```

Format of Show Locks List

DBUTIL lists the locking information sequentially by locking level: database locks followed by data set locks, followed by data entry locks. The names of locked entities (for example, the database, data set, or lock descriptor for data entries) appear in uppercase followed by a list of other processes waiting at that locking level. DBUTIL indicates in lowercase 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 lowercase.

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/XL 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/XL 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 MPE/iX commands, such as SHOWJOB and SHOWQ.

Example 1 (Show Locks)

```
        .
        .
        .
   >>SHOW ORDERS LOCKS OFFLINE     List the status of locks requested and held in
                                   the ORDERS database on the line printer.
```

The line printer listing looks similar to this:

```
                HP30391C.00.00 TurboIMAGE/XL: DBUTIL  THURS, SEP 21, 1989, 5:06 PM

                For database ORDERS

                                                                  PIN/   PROGRAM
                LOCKED ENTITY - ( - waiting process)              PATH   NAME      JOBNUM

1               DATA SET SALES                                    30/1   BROWSE    #S126

2               -waiting for data set unlock:                     17      INVENTRY #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
```

Example 1 Discussion

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 INVENTRY executing in session 128) is waiting. Because it is listed first in the queue, it will be the next process to resume execution after the SALES data set is unlocked. It could 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.

Example 2 (Show Locks)

Here is another example of a locking list that might appear when the >>SHOW LOCKS command is entered.

```
              HP30391C.00.00 TurboIMAGE/XL: DBUTIL  THURS, SEP 21, 1989, 5:15 PM

              For database ORDERS

                                                          PIN/    PROGRAM
              LOCKED ENTITY - ( - waiting process)        PATH    NAME      JOBNUM
1             DATABASE (PENDING)                          22      BROWSE    #S118
              -waiting for zero locks within database:22      BROWSE    #S118

2             DATA SET INVENTORY                          29/1    INVENTRY  #S115

3             SALES: QUANTITY<= 50                        28/1    BROWSE    #S112
4             CUSTOMER: CUST-NAME = DON'S MERCANTILE      31/1    ORDENTRY  #S117
```

Example 2 Discussion

1 Indicates process 22 (program BROWSE, session 118) has requested a lock on the database and yet it cannot continue until existing locks held in the database are released. In this example, the reason for the pending lock is listed on the line below.

2 Indicates process 29 (program INVENTRY, 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.

All subsequent requests for locks must be made to wait until process 22 releases its database lock.

>>VERIFY

Reports whether a remote database-access (RDBA) file is activated or deactivated and checks the validity of the RDBA file.

Syntax

```
>>V[ERIFY] database-access file name
```

For example:

```
VERIFY ORDDBA
```

where ORDDBA is the *database-access file name*.

Parameter

database-access file name is the name of a remote database-access file.

Example

```
:RUN DBUTIL.PUB.SYS     Initiate DBUTIL execution.
   .
   .
   .
>>VERIFY ORDDBA         Enter >>VERIFY command and database-access file name.
Database-access file
ORDDBA is ACTIVATED
>>
```

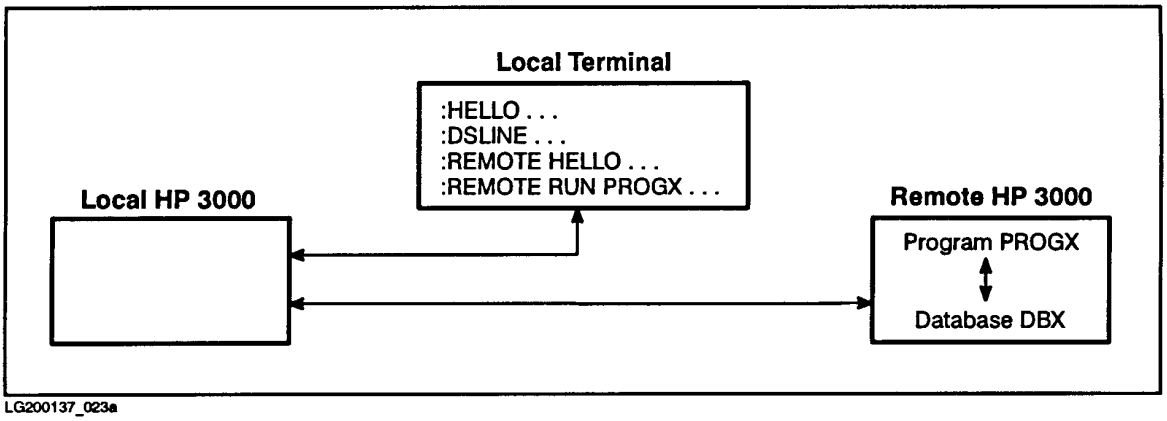
When an RDBA file is activated, it is changed to a privileged file and cannot be edited; it is changed back to an editor file when it is deactivated.

9 Using a Remote Database

You can access a TurboIMAGE/XL database that resides on one HP 3000 computer system while operating a session on another HP 3000 computer system if both systems are configured with Network Services capability. You can access a TurboIMAGE/V database from an MPE/iX system or a TurboIMAGE/XL database from an MPE V system. Programmatically accessing an IMAGE/3000 database from a TurboIMAGE/XL database requires another MPE V system with NS/3000 and TurboIMAGE/V acting as an intermediate node. The limits on the remote database must not exceed those allowed on the local system, or programmatic access will not be successful.

You can use a database 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 remote session through a communications link then run a remote program accessing a database on the remote machine as illustrated in Figure 9-1.

Figure 9-1. Using a Remote Program



For details about using this method, refer to the *NS3000/XL User/Programmer Reference Manual*.

Access Through a Local Application Program

If you want to access a remote database using a local application program, you have three methods to choose from.

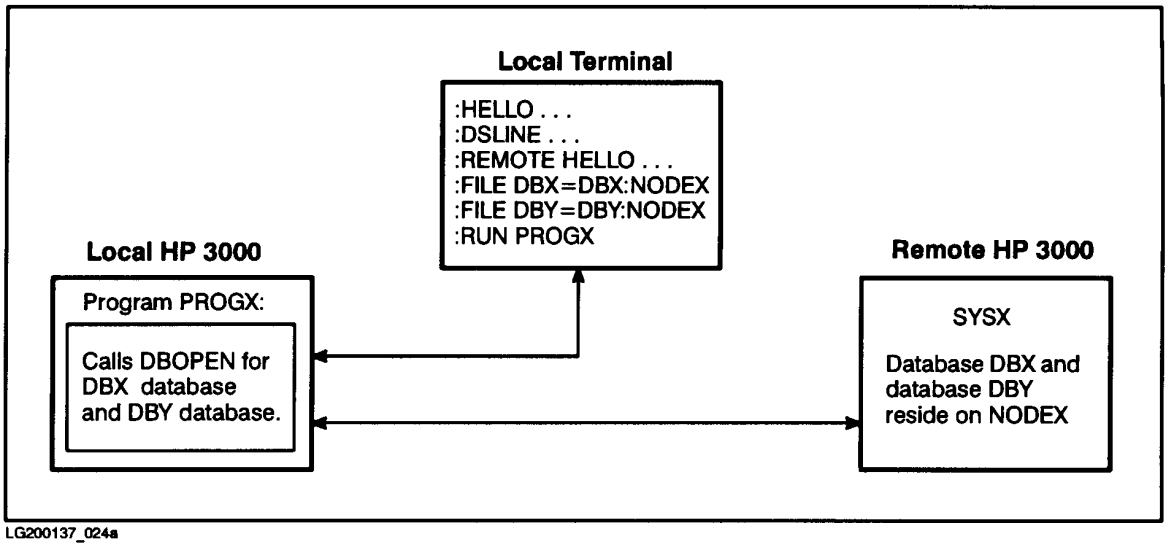
- Establishing communications link and remote session interactively.
- Using the COMMAND intrinsic.
- Using a database-access file.

In all three cases, a local program accesses a remote database and the data is passed across the communication line.

Method 1--Establishing Communications Link and Remote Session Interactively

To use the first method, interactively establish a communications link and a remote session and enter a FILE equation for each remote database. The FILE equation specifies which database is to be accessed on which remote system and device. A local application program can now access a remotely located database, as shown in Figure 9-2.

Figure 9-2. Using Method 1



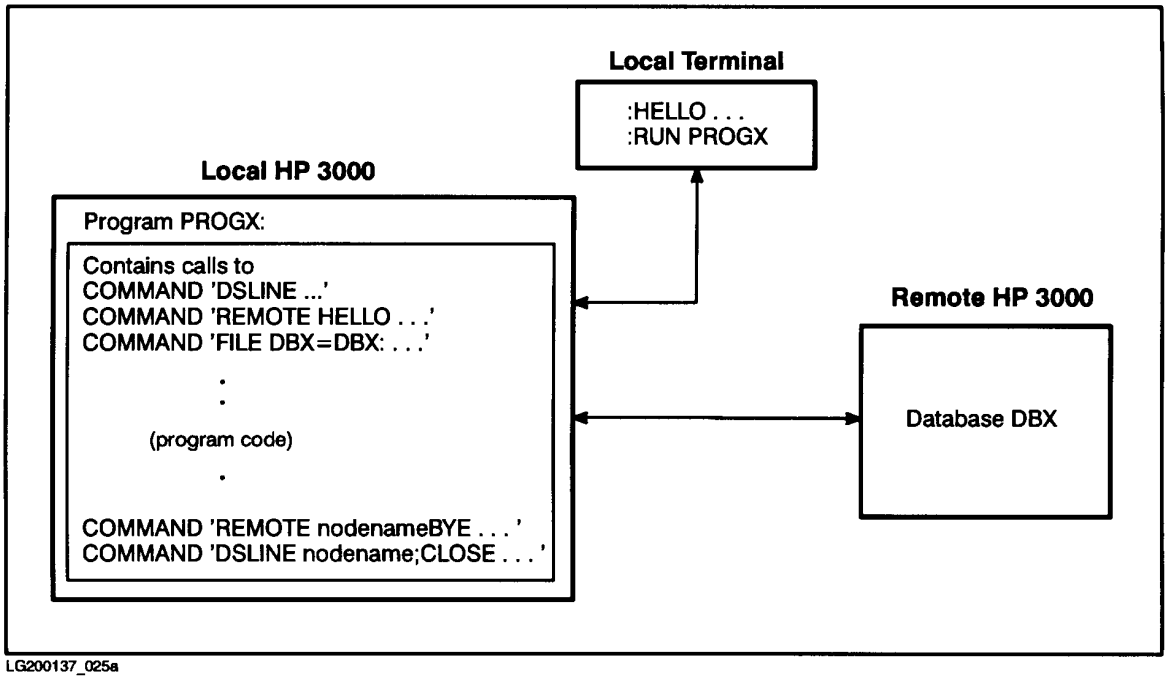
For details about using this method refer to the *NS3000/XL User/Programmer Reference Manual.*

Method 2--Using the COMMAND Intrinsic

The second method is very similar to the first, but you use the MPE/iX COMMAND intrinsic within your application program to establish the communications link, remote session and remote database access.

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/iX Intrinsics Reference Manual*, and information about accessing remote files is given in the *NS3000/XL User/Programmer 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 database with an application program, you must enter one FILE equation for each remote database.

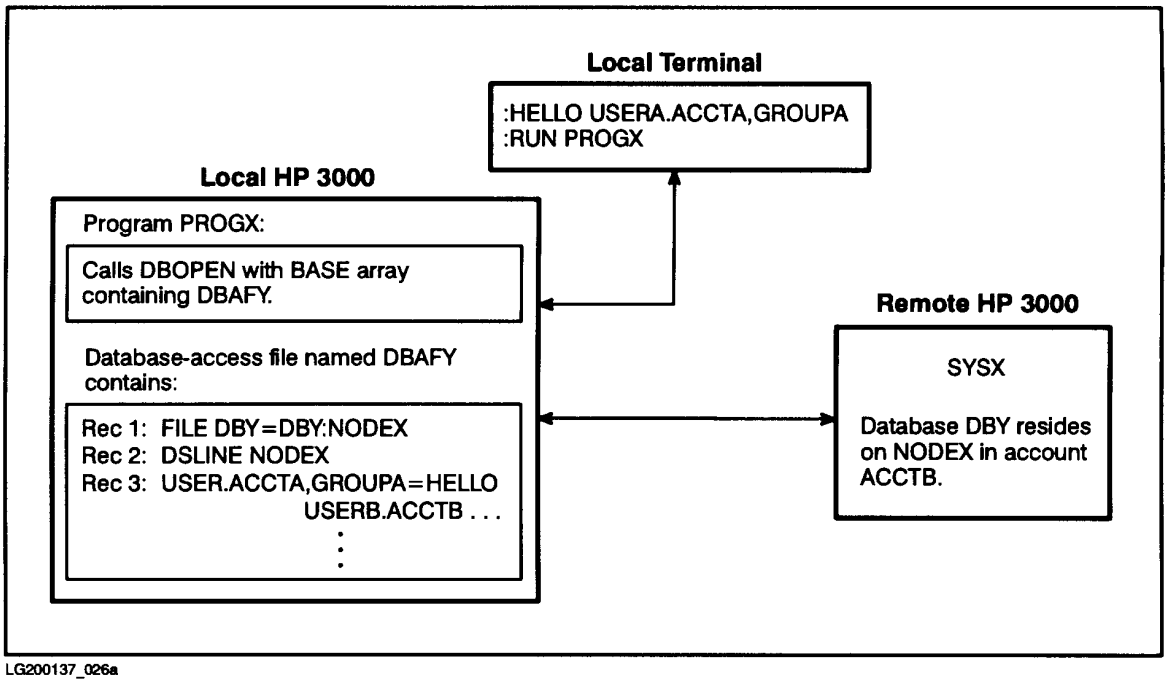
To terminate execution perform the following steps:

1. Close the database.
2. Log off (REMOTE BYE).
3. Close the communication line.

Method 3--Using a Database-Access File

The third method involves creating a special privileged (PRIV) file which is called the database-access file (DBA file). This file provides TurboIMAGE/XL with the necessary information to establish a communications link and a remote session. It also specifies the remote database or database-access file name so that the necessary TurboIMAGE/XL intrinsics can be executed on the remote computer. Figure 9-4. illustrates method 3.

Figure 9-4. Using Method 3



Using the database-access file, only one database can be accessed using each database-access file per DSLINE. For example, if two computers are linked through two DSLINEs, you can open one database on each line.

When the user or an application program calls DBOPEN with the database-access file name, the remote session is established and the remote database is opened. Then other TurboIMAGE/XL intrinsics can perform desired operations on the database. Under this method the remote session is automatically released when the database is closed (with or without an explicit DBCLOSE call). A second REMOTE HELLO on one DSLINE terminates the previous REMOTE HELLO. For multiple remote database access, method 1 or method 2 is recommended. If the database-access file is used, an automatic REMOTE BYE and DSLINE commands are issued on the communications line specified in the database-access file when the application program closes the database or terminates execution.

Using method 3, the database administrator can set up a user-table that provides more control over the database access and enhances database security. To create the database-access file, use an editor such as EDIT/3000. First use the SET LENGTH command to accommodate the largest record to be included in the database-access file if the record exceeds the default length specified for your editor. The length must be less than or equal to 128 characters.

The following sections discuss how to create, activate, deactivate, and reference a database-access file.

Creating a Database-Access File

The content of the database-access file should be created in the format shown below.

Syntax

Record 1 FILE *dbname1=dbname2:nodename*

Record 2 `DSLINE` *nodename*`[; . . .]`
(See NS3000/XL document for applicable parameters.)

Record 3 *lusername.lacctname*`[,`*lgroupname*`]=HELLO` *rusername*
`[/`*rupasw*`]` *.racctname*`[/`*rapasw*`][,`*rgroupname*`[/`*rgpasw*`]] . . .`
(See MPE/iX commands document for additional parameters.)

Record 4 Same format as record 3. Specifies other "user.account,group" identification.

.
.
.

Record *n*

Parameters

dbname1 is the name of the database-access file on the local system or the database on the remote system you want to access, or is the formal file designator used in the program if *dbname2* is specified. (Required parameter.)

dbname2 is the name of the database-access file or the database on the remote system you want to access. (Required parameter.)

nodename is the remote location of the database. This parameter is preceded by a colon when used with the FILE command; if followed by options with the `DSLINE` command, it is followed by a semicolon. (Required parameter.)

Refer to the *NS3000/XL User/Programmer Reference Manual* for additional information.

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 (@) can be used to indicate the logon user name. (Required parameter.)

lacctname 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 (@) can be used to indicate the logon account. (Required parameter.)

lgroupname is the name of a file group to be used for the local file domain as established by an account manager. It contains from 1 to 8 alphanumeric characters, beginning with a letter. An at-sign (@) can be used to indicate the logon group. (Optional parameter.)

rusername 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 *username*. An at-sign (@) can be used to indicate *rusername* as with *lusername*. (Required parameter.)
rupasw	is the password assigned to *username*. (Optional parameter.)
racctname	is the name of the logon account on the remote HP 3000. It follows the same rules as *lacctname*. An at-sign (@) can be used to indicate *racctname* is the same as *lacctname*. (Required parameter.)
rapasw	is the password assigned to *racctname*. (Optional parameter.)
rgroupname	is the name of the logon group on the remote HP 3000. It follows the same rules as *lgroupname*. An at-sign (@) can be used to indicate *rgroupname* is the same as *lgroupname*. (Optional parameter.)
rgpasw	is the password assigned to *rgroupname*. (Optional parameter.)

Discussion

The following syntax rules apply:

- No spaces are allowed around any periods that could exist in the second file reference in record 1 (for example, *dbname2.group*).
- Passwords are not allowed with the local user, account, and group names. They are not necessary because the local user passes the security password checks when logging onto the local system.

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 database-access file and must be supplied in the HELLO command.

After you have created the file with an editor, you must KEEP it UNNumbered. The file name must follow the same rules as a database name. It must be an alphanumeric string from 1 to 6 characters; the first character must be alphabetic.

Records 3 through *n* define a table that tells TurboIMAGE/XL which user, account, and group names on the local computer can access which user, account, and group names on the remote computer. You can specify remote user identification for more than one local user by creating a record for each local "user.account,group" using the record 3 format shown earlier. An at-sign (@) can be substituted for any user, account, or group name in the record. If an at-sign is substituted for *lusername*, *lacctname*, or *lgroupname*, the name is replaced with the corresponding name specified at logon time.

TurboIMAGE/XL 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/XL performs a REMOTE HELLO using the corresponding *rusername, racctname, rgroupname*, 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/XL 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 logon identification.

Example

Record 1 `FILE ORDERS=ORDERS:NODEX`

Record 2 `DSLINE NODEX`

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 logon identification indicated in the first column below, TurboIMAGE/XL uses the corresponding `USER.ACCT,GROUP` identification in the second column to establish communication with the remote system.

Logon 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 logon 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/XL looks for another table entry with account ACCTA. Because 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 because 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 database with this application program.

Activating a Database-Access File

After you have constructed a database-access file, you must use the DBUTIL utility program to activate the file. The complete syntax for running the utility program is given in chapter 8. Here is a summary of the operating instructions:

```
:RUN DBUTIL.PUB.SYS
     .
     .
     .
>>ACTIVATE database-access file name

Verification follows:

FILE command:    <result>
DSLINE command:  <result>
HELLO command:   <result>

ACTIVATED

>>EXIT
```

DBUTIL verifies that the file to be activated adheres to the following criteria:

- Has a file code of zero.
- Is an unnumbered, ASCII file.
- Has a record length <=128 characters.
- Has at least three records.

If any of these conditions is not satisfied, activation fails, and the following message is printed:

```
filename is NOT a suitable database-access file
```

If the file has already been activated, the following message is printed:

```
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 the following records:

- Record 1.
- Record 2 through *nodename*; must be identical to the *nodename* 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 keyword parameters are checked by the command interpreter at DBOPEN time.

If all of the above conditions are met, DBUTIL successfully activates the database-access file, by changing the file code to the TurboIMAGE/XL reserved code –402, which makes it a privileged (PRI V) file.

Deactivating a Database-Access File

In order to deactivate the database-access file, you use the DEACTIVATE command of the DBUTIL utility program. Complete syntax for this program is given in chapter 8. Here is a summary of the operating instructions:

```
:RUN DBUTIL.PUB.SYS
      .
      .
      .
>>DEACTIVATE database-access file name
DEACTIVATED
>>EXIT
```

You can do this if you want to edit the content of the database-access file or to prevent access through this file to the remote database.

Referencing the Database

To reference the database from your local application program, use the database-access file name instead of the root file name when calling the TurboIMAGE/XL procedure. The word array specified as the base parameter must contain a pair of blanks followed by the left-justified database-access file name and terminated by a semicolon or blank ( ). TurboIMAGE/XL recognizes the –402 file code and establishes a communications link to the remote HP 3000. If the database is successfully opened, TurboIMAGE/XL replaces the pair of blanks with the *baseid* of the assigned Remote Database 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.

Example

The example in Figure 9-5. illustrates how to create and activate a database-access file. A description follows the example.

In this sample case, the file named ORDDBA is to be used to gain access to the ORDERS database residing on a remote system in the PAYACCT account. The remote system is referenced by *nodename* NODEX.

After the database-access file is created, it is enabled by using the DBUTIL utility program.

Figure 9-5. Preparing a Database-Access File

```
1     :HELLO MEMBER1.PAYACCT
          .
          .
          .
2     :EDITOR

       HP32201A.00.00 EDIT/3000  FRI, SEP 6, 1991, 3:30 PM
       (C) HEWLETT-PACKARD CO. 1985
3     /ADD
          1     FILE ORDERS=ORDERS:NODEX
          2     DSLINE NODEX
          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
```

Description

1 Initiate an MPE/iX 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 database-access file.

4 Save the work file in a disk file (ORDDBA in the above example). Remember to keep it unnumbered. Then exit the Editor.

5 Initiate execution of DBUTIL and activate the database-access file ORDDBA. Verification messages will follow (in session mode). Exit from DBUTIL.

Example

Figure 9-6. illustrates use of the database-access file through a program named APPLICAN. A description follows the example.

In this sample case, 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 ORDDBA. When a call to DBOPEN is executed, TurboIMAGE/XL establishes a communication line and remote session. When the program closes the database, TurboIMAGE/XL closes the line and terminates the remote session.

Figure 9-6. Using a Database-Access File

```
1     :HELLO MEMBER2.PAYACCT
              .
              .
              .
2     :RUN APPLICAN
3     HP3000 / MPE/iX  G.00.00  FRI SEP 6, 1991,  3:55 PM
      ENVIRONMENT 1:  ELECTRA.DOC.ITG
4     WELCOME TO SYSTEM B.
              .
              .
              .
      CPU=2.  CONNECT=1. FRI, SEP 6, 1991,  3:59 PM
      ENVIRONMENT 1: ELECTRA.DOC.ITG CLOSED
5     :BYE
```

Description

1 Initiate an MPE/iX session on the local system by logging on with the appropriate user name and account.

2 Execute the application program APPLICAN. (The program calls DBOPEN using ORDDBA as the *baseid*.)

3 TurboIMAGE/XL establishes a communications line and remote session.

4 When the database is closed, TurboIMAGE/XL closes communications line and ends remote session.

5 Log off local system.

Access Using QUERY/3000

When you use QUERY/3000 to retrieve information from a database, you must specify a database name, password and access mode before you can actually access the database. The "DATABASE=" prompt can be answered with a remote database name or the database-access file name. Note, however, that performance can be significantly improved if you run QUERY/3000 in a remote session, thereby accessing the database on the system where it resides, rather than running QUERY/3000 locally to access a remote database.

For more information, refer to the *QUERY/V Reference Manual*.

10 Internal Structures and Techniques

In addition to the data elements discussed in chapter 2, TurboIMAGE/XL uses a number of internal structures and techniques to provide rapid and efficient access to the database content. This chapter describes these structures and techniques to give you an overview of the way TurboIMAGE/XL works.

NOTE In this manual a **word** is a 32-bit storage unit and a **halfword** is a 16-bit storage unit. One byte is 8 bits.

Data Set Internal Structures

The following internal structures are used by TurboIMAGE/XL to manage the information in data sets.

Pointers

TurboIMAGE/XL uses pointers to link one data set record to another. A **pointer** is a value containing the block number in the first three bytes plus one byte that contains the offset within the block for a given data entry.

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 with 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 links 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 can consist of at most 2^{31}–1 (2,147,483,647) entries.

Chain Heads

TurboIMAGE/XL 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 in the corresponding master data set with the entry whose key item value is the same as the detail search item value. Each chain head is 12 bytes long. The first four bytes contain a count of the number of member entries in the referenced chain. The count is zero if the chain is empty. The remaining eight bytes contain two 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 one, these pointers have the same value.

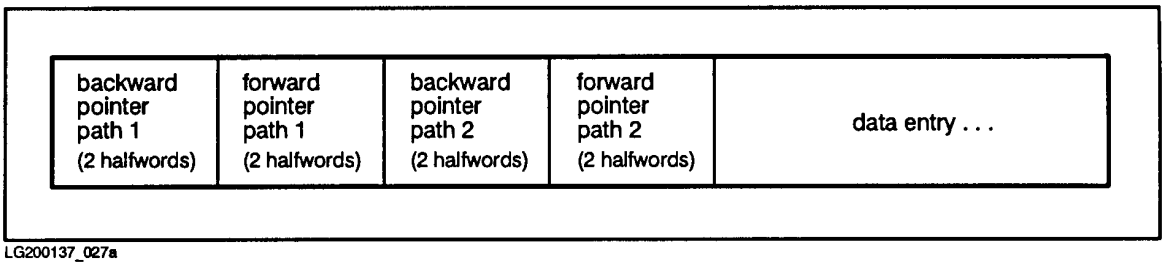
Media Records

TurboIMAGE/XL transfers information to and from a storage location on disk in 4096-byte pages using the system storage manager. Access to the data in memory is synchronized through blocks of media records. A **media record** consists of both a data 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 data 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.

Figure 10-1. Media Record for Detail Entry



Media Records of Master Data Sets

Media records of master data entries are composed of the following:

- A 10-byte field serving as a synonym chain head for primary entries or a synonym chain link for secondary entries.
- A 3 times *n* word field in which the chain heads of all related detail chains are maintained. *n* is the number of paths defined for the master data set. Between 0 and 16 paths can be defined.
- The data 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

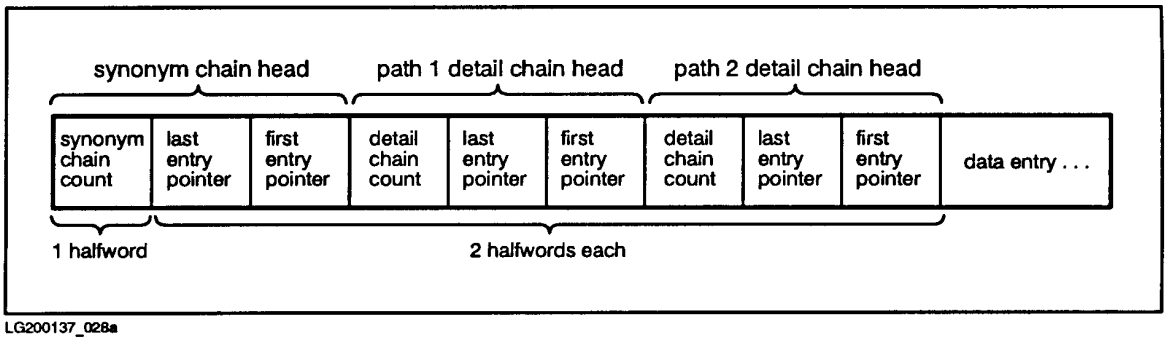
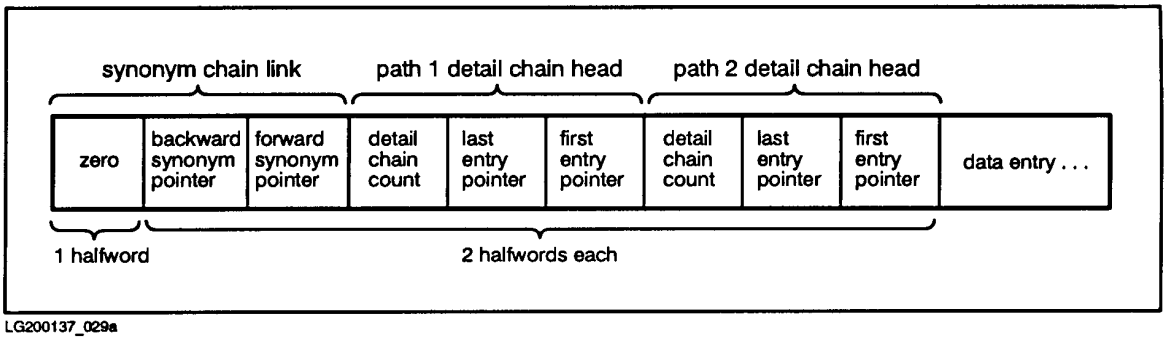


Figure 10-3. illustrates a media record for a secondary entry of a master data set with two paths defined.

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.

Primary Entries

Selection of record addresses for master entries begins with a calculated address determined by a hashing algorithm applied to the value of each entry's key item. The algorithm is described later in this chapter. 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 key item value will be assigned the same primary address as an existing primary entry whenever the key item values of both entries generate the same calculated address. When this occurs, the entries are considered **synonyms** of one another. TurboIMAGE/XL assigns the new entry a **secondary address** obtained from unused records in the vicinity of the primary entry. For master sets which are designated for dynamic expansion, a secondary address may also be obtained from the expanded area. Each entry residing at a secondary address is called a **secondary entry**.

Synonym Chains

When multiple data entries "arrive" at the same primary area, they are linked together to form a synonym chain. A **synonym chain** consists of the primary entry and all of the data entries with key values that hash to the same key location. Each synonym chain is maintained by a 10-byte chain head in the media record of the primary entry and 10-byte links in the media records of the secondary entries. A master data set entry can 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 one (for the total number of entries that hash to this location) and the pointers to the first and last secondary entries are zeros. If one or more secondaries are present, the synonym chain count is equal to the total number of entries that hash to the same primary address and the pointers reference the first and last secondary entries.

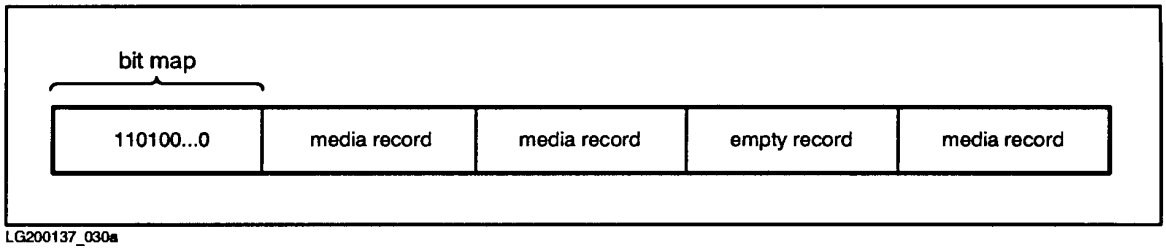
The first two bytes of the 10-byte link in the media record of each secondary entry are always zero. The remaining eight bytes consist of two pointers bi-directionally linking the secondary entries of the synonym chain to each other. As with detail chains, the first member of this chain of secondary entries contains a zero backward pointer, and the last member of the chain contains a zero forward pointer.

Blocks and Bit Maps

Each group of media records involved in a single MPE file record is a **block**. The first few halfwords of each block contain a bit map employed by TurboIMAGE/XL to indicate whether the corresponding media record is full or 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 halfwords 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



Dynamic Data Set Expansion

The capacity of non-jumbo (less than or equal to 4 GB in size) master and detail data sets can be expanded dynamically during DBPUT, or implied DBPUT for automatic master. The expansion can occur only if the data set is enabled for dynamic expansion, that is, if the required capacity parameters are set prior to the expansion.

The capacity parameters are:

- maximum capacity

  is a required parameter and is a maximum number of entries the data set can contain. It must be less than or equal to 2^{31}–1 (2,147,483,647). The maximum capacity both for masters and details is adjusted by TurboIMAGE/XL to represent an even multiple of the blocking factor.

- initial capacity

  is a required parameter and is the initial capacity for the data set, that is, the number of entries for which space will be allocated and initialized when the data set is created. For a master data set, this is also the primary or hashing capacity. This number must be between 1 and 2^{31} –1 inclusive but must be less than or equal to the maximum capacity. This parameter should be used to closely approximate the current volume of data. If it is very low, there can be frequent expansions leading to severe disk fragmentation. If it is very high, DBPUT may take a long time to complete which could impact other database users. The initial capacity is adjusted to represent an even multiple of the blocking factor. If initial capacity is not specified, or if initial capacity is either zero or equal to the maximum capacity, then dynamic capacity expansion is not enabled for the data set, and maximum capacity is used for the data set file creation.

- increment

  is an optional parameter and is either the number of entries or the percentage of the initial capacity by which the data set will be expanded each time its initialized space is exhausted. If a percentage is used, the percent sign (%) must follow the incremental amount. This increment parameter can only be used if the initial capacity parameter is also specified. This number must be 1 to 32767 inclusive for percent, or 1 to 2^{31} –1 (2,147,483,647) inclusive for number of entries. If it is very low, there can be frequent expansions leading to severe disk fragmentation. If it is very high, DBPUT may take a long time to complete which could impact other database users.

  The number of entries defined, or the entries calculated from the percent, cannot exceed the maximum entry count minus the initial allocation. That is, the expansion must not result in exceeding the maximum capacity. The increment is adjusted to represent an even multiple of the blocking factor. If the increment is not specified for the data set, or is zero, but the initial capacity is greater than zero, then the increment for each expansion is defaulted to ten percent (10%) of the initial capacity for the data set. If the initial capacity is equal to the maximum capacity, or the initial capacity is zero, then this indicates the data set cannot be expanded and increment is ignored.

In brief, a data set is enabled for dynamic expansion when both are true:

- Both maximum capacity and initial capacity are specified
- Initial capacity is neither zero nor equal to the maximum capacity

You can select data sets which need dynamic expansion based on anticipated growth.

For new databases, the expansion parameters for data sets can be specified using the CAPACITY statement of DBSCHEMA as follows:

```
{ CAPACITY:
  C:        } maximum capacity[(blocking factor)][ ,initial capacity
                                                   [,increment]      ] ;
```

To specify the expansion parameters for data sets of existing databases, use DBChange Plus, or other third-party tools which support this feature.

In order to use this feature from the user point of view, the only required step is to identify and enable the data set(s) for dynamic expansion as described above.

When a data set enabled for dynamic expansion is first created, disk space for only the initial capacity is allocated and initialized. Later when the data set is expanded during DBPUT, or implied DBPUT for automatic master, additional disk space based on the increment is allocated and initialized. It can grow up to the maximum capacity specified for the set. Following the expansion, TurboIMAGE/XL updates various fields related to the expansion such as current capacity including the expansion and free space counter in the user label.

The instant when the expansion is triggered and how record addresses are assigned to the new entry vary in detail data sets from master data sets.

Detail Data Sets

For a detail data set, expansion takes place during DBPUT when the free space counter for the set is zero. Following the expansion, TurboIMAGE/XL updates the expansion related fields in the user label such as the end-of-file pointer (high-water mark) and the free space counter. The expanded data set is perceived as one larger data set and the new record addresses are assigned using the pointer to delete chain head and the end-of-file pointer (high-water mark) as done prior to the dynamic expansion feature. The new entry is assigned the first record address in the expanded area.

Master Data Sets

For a master data set, expansion is triggered when the set is almost full. That is, when the set has approximated its maximum capacity which will not allow DBPUT to be successful. For an example, a DBPUT to a detail set with multiple paths to the same automatic master will require multiple record additions to the automatic master.

Once the expansion is done, the master data set can be perceived as having two areas, original area and expanded area, managed differently. The original area can have primary entries and secondary entries, while the expanded area can only have the secondary entries. Note that the primary capacity (also called hashing capacity or initial capacity), which is used to calculate the primary address does not change. In essence, the dynamic expansion results in allocating additional space for secondary entries which implicitly makes room for additional primary entries in the original area. The primary address of a

new entry is calculated using the value of the entry's key item in the hashing algorithm. The new entry can reside either at its primary address in the original area, or in the close vicinity of the primary address in the original area, or in the expanded area. The use of expanded area is controlled using the pointer to a delete chain head and an end-of-file pointer (high-water mark) as in detail data sets.

To elaborate further, when a DBPUT for manual master, or implied DBPUT to an automatic master, is processed, the address where the new entry resides is based on the following:

1. If the primary address in the original area is not occupied, the new entry resides there. The new entry also becomes a synonym chain head with a count of one.
2. If the primary address is occupied by another primary entry, TurboIMAGE/XL scans a few blocks (quick search) in the close proximity of the primary address to find an unoccupied address (empty record). If found, the new entry resides at this secondary address in the original area. Otherwise, the pointer to the delete chain head and the end-of-file pointer for the expansion area are interrogated to determine the secondary address in the expansion area for the new entry. When there is room, the new entry is assigned a secondary address in the expanded area. If the expanded area is full and can be expanded, it is further expanded to accommodate the new entry. When there is no room in the expanded area and cannot be expanded further, however, there is room in the original area, the original area is scanned once again (long search) to find a secondary address for the new entry. If an unoccupied address is found, the new entry resides there, and it becomes the last entry in the synonym chain. Otherwise, the set is full and DBPUT will fail.
3. If the primary address is occupied by a secondary entry, a secondary entry is relocated to another secondary address in either the original area or the expansion area as described above. The new entry becomes a synonym chain head with a count of one. TurboIMAGE/XL performs synonym chain maintenance for the secondary entry which relocates.

Scalability

In versions prior to C.07.00, TurboIMAGE/XL serialized the execution of DBPUT, DBDELETE, and DBUPDATE when critical item update (CIU) is enabled, to protect the structural integrity of the database. This serialization is done via a semaphore known as PUTDELETE semaphore. This translates into processing only one DBPUT, DBDELETE, or DBUPDATE with CIU enabled at a time for every database. This is acceptable for the low-end machines but not for the high-end and multi-processor machines. It became imperative that TurboIMAGE/XL scale to the performance of these high-end and multi-processor machines. That is, TurboIMAGE/XL must increase the throughput of the intrinsics, DBPUT, DBDELETE, and DBUPDATE with CIU enabled, which are collectively termed **modify intrinsics**. In order to achieve this, the approach taken is to increase the concurrency of the modify intrinsics.

Approach in Version C.07.00

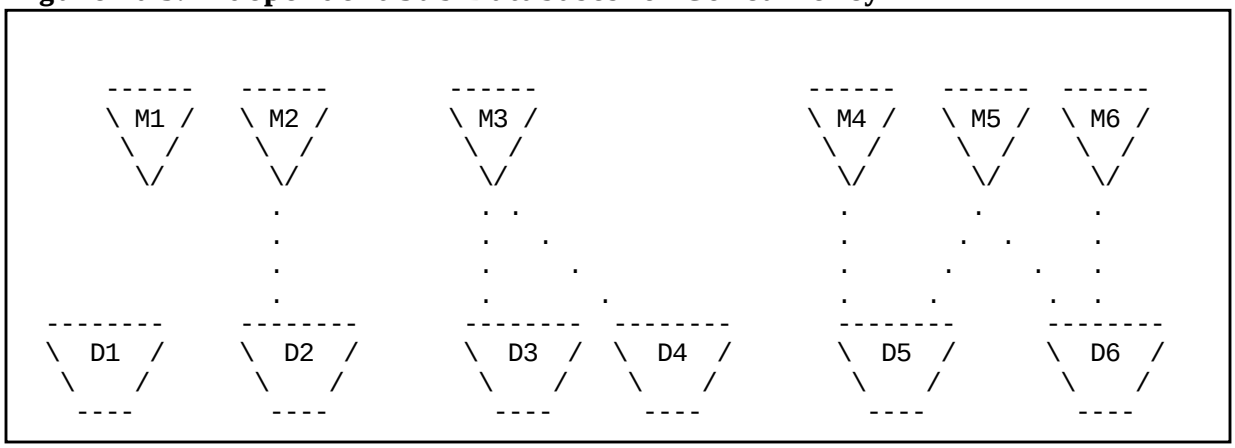
The enhanced approach in TurboIMAGE/XL version C.07.00 (and later) is based strictly on the database design and is optional. By default, TurboIMAGE/XL continues to work the way it did in versions prior to C.07.00. For the revised approach, the database is internally grouped by TurboIMAGE/XL into independent sub-databases based on physical linkages/relationships or dependency of master and detail data sets. In addition, it uses multiple semaphores as well as a specific criteria to lock the necessary semaphores for a sub-database. The result is that the modify intrinsics can execute concurrently for these independent sub-databases.

In brief, the criteria pertaining to semaphores is as follows:

1. Assign ONE semaphore for each data set.
2. When modifying a master set, lock only the semaphore for this master.
3. When modifying a detail data set, lock the semaphore for the detail as well as all its related master sets, manual as well as automatic. This can result in locking 1 (minimum, standalone detail) up to 17 (maximum, 1 + 16) semaphores.

To illustrate this, look at the following diagram:

Figure 10-5. Independent Sub-Databases for Concurrency



In the above diagram, the independent sub-databases are:

1. M1
2. D1
3. D2 and M2
4. D3 and M3
5. D4 and M3
6. D5, M4 and M5
7. D6, M5 and M6

There are several different scenarios when concurrent modifications can be in progress. For example:

M1	at all times as there is no dependency.
D1	at all times as there is no dependency.
Other masters	M2, M3, M4, M5, and M6 if the depending detail is not being modified.
Other details	D2, D3 or D4, and D5 or D6. This is if the dependent master(s) are not being modified. Note that both D3 and D4, or D5 and D6 cannot be concurrently modified.

Based on this approach, the throughput is dependent on the number of sub-databases, the more, the better. The worst case is a database which can be sub-divided into only one sub-database. For example, one detail linked to 16 masters. The best case is when there are several stand-alone masters or details.

With this approach, some databases may benefit in throughput and some may not. Therefore, by default, TurboIMAGE/XL continues to work the way it did in versions prior to C.07.00.

To use this enhancement, your database must be activated to use the **Dependency Semaphore**. To achieve this, use the option, DSEM, with the `ENABLE` command of `DBUTIL`. The default for DSEM is DISABLED.

New syntax:

`EN[ABLE]` database name [/maint word] FOR DSEM

`DI[SABLE]` database name [/maint word] FOR DSEM

After enabling your database for DSEM, if you feel that your database cannot benefit by using this feature, you may DISABLE it.

Run-Time TurboIMAGE/XL Control Blocks

As mentioned in chapter 4, TurboIMAGE/XL uses control blocks resident in privileged mapped files to provide and control user access to a database through the TurboIMAGE/XL procedures. The contents of these control blocks are maintained by TurboIMAGE/XL. Although it is not necessary to know the details in order to use the TurboIMAGE/XL procedures, the following descriptions are provided for those who prefer to understand the control blocks and their functions.

Local Database Access

The following structures are involved in local database access:

- Database System Control Block **(DBS)** -- Contains pointers to all of the DBGs on the system.
- Database Globals Control Block **(DBG)** -- Contains global information required during run-time and space for DBLOCK's lock descriptors. The DBG also contains a pointer to the DBB and pointers to the DBUs.
- Database Buffer Area Control Block **(DBB)** -- Contains a set of addresses and temporary locks used to coordinate access to the data set, a set of buffer headers, and a pointer to the DBG.
- Database User Local Control Block **(DBU)** -- Contains information pertaining to each access path (each DBOPEN) to the database. The DBU also contains pointers to the DBG, DBB, and DBS.
- Database User Local Index Control Block **(DBUX)** -- Contains the addresses of all the DBUs and DBRs (Remote Database Control Blocks) belonging to a specific process. The DBUX also contains dynamic roll-back information.
- Database Lock Table **(TURBOLKT)** -- Contains information pertaining to locks on the database and is used to avoid deadlocks.
- Multi-Database Transactions **(TURBOGTX)** -- Contains information pertaining to dynamic transactions constituting multiple databases.
- QOPEN Table **(QOPEN)** -- Contains information about user logging process for active DBOPENS of modes 1-4.
- QLOCK Table **(QLOCK)** -- Contains information about all writers to databases and a pointer to QOPEN.

The DBS is a permanent file, TURBODBS.PUB.SYS, that is created by DBOPEN if it does not exist beforehand. Thereafter, it is opened when the first user opens any database on the system. It is reinitialized after a system abort. Each system has only one DBS; it contains pointers to the current DBGs for any currently open database(s).

Both the DBG and the DBB are created when the first user opens a database (DBOPEN). They remain allocated until the last user closes the database. Each opened database has only one DBB and one DBG, regardless of the number of users. Both of these control blocks are part of a permanent mapped file called *dbname*GB located in the same group and

account as the database.

NOTE If you have an existing file with the same name that TurboIMAGE/XL would assign to the permanent mapped file (that is, *dbname*GB), you will get the following message for status code –9:

```
CANNOT CREATE DBG : MPE ERROR <nnn>
```

The DBG is derived mostly from the root file and contains global information required by TurboIMAGE/XL 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 is used to retrieve, log, and update data located within the data sets. It contains a set of buffer headers which are shared by all concurrent users accessing the database. These buffer headers contain information about data set blocks. The DBB also contains information pertaining to logging and recovery.

A two-level resource priority locking scheme is used within the DBB to allow single data block operations to access the control block concurrently. This involves DBGET, DBFIND and DBUPDATE processes. DBPUT and DBDELETE operations, and DBUPDATE operations on detail data set search or sort items made possible through the critical item update (CIUPDATE) option, are unable to access the data blocks concurrently with other DBPUTs, DBDELETEs, and search or sort item DBUPDATEs. These block put-delete lock operations must hold a global lock on the DBB throughout part of the operation; therefore, there is less concurrency when using DBPUT and DBDELETE and when using DBUPDATE on search or sort items via the CIUPDATE option. When you update a search or sort item value, you get the same net effect as performing a DBDELETE and DBPUT.

One DBU is created and stored in a privileged, unnamed temporary file each time a user issues a DBOPEN, and remains allocated until the corresponding DBCLOSE is issued. The DBU contains information pertaining to the user's individual access to the database. This includes information about the user's access mode, record position, list specifications, and security table. All TurboIMAGE/XL intrinsics process on the DBU, except accesses for global and buffer area information found in the DBG and DBB.

The DBUX is created and stored in a privileged, unnamed temporary file the first time the user's process calls DBOPEN. One DBUX exists for each process. Its purpose is to keep track of the addresses of all the DBUs belonging to that process. Because 127 entries are allowed in the DBUX, each process is allowed a maximum of 127 DBOPENs (63 per database) depending on the availability of system resources. The DBUX remains allocated until the user's process is terminated.

When accessing a local database, the TurboIMAGE/XL procedures usually make use of, and can modify information in all of the control blocks.

The TURBOLKT is a permanent file, TURBOLKT.PUB.SYS, that is created by DBOPEN (if it does not exist beforehand). Thereafter, it is opened when the first user opens any database on the system. It is purged when the system is rebooted. Each system has only one TURBOLKT file. It is used to avoid deadlocks for all IMAGE/SQL users. Additionally, it is also used to detect potential deadlock for TurboIMAGE/XL users if, and only if, deadlock detection is activated by DBCONTROL mode 7.

The TURBOGTX is a permanent file, TURBOGTX.PUB.SYS, that is created by DBXBEGIN (if it does not exist beforehand). Thereafter, it is opened for all users who employ dynamic multi-database transaction(s) (DMDBX). Each system has only one TURBOGTX file, and it remains on the system even after the system is rebooted. It is used for tracking DMDBX.

The QOPEN table is an unnamed global object and is first created by DBOPEN when the first writer of any database enabled for user logging opens a database. It is subsequently accessed only by writers of databases enabled for user logging. Each system has only one QOPEN table, and it is purged when all processes accessing it are terminated. It contains information pertaining to user logging process for DBOPEN. This information is used to write to log records for DBRECOV and to coordinate with DBQUIESCE called by TurboSTORE/iX 7x24 True-Online Backup.

The QLOCK is an unnamed permanent global structure that is created by DBOPEN (if it does not exist beforehand). There is one per system, and it is accessed by all writers to all databases. It is purged only at system reboot time. It is used for containing database information required to quiesce database(s) for TurboSTORE/iX 7x24 True-Online Backup.

Remote Database Access

TurboIMAGE/XL provides the capability of accessing a database on a remote HP 3000 system from a user program running on the local HP 3000 system, as described in chapter 9. This capability is provided in conjunction with NS3000/XL and is accomplished by transmitting TurboIMAGE/XL database access requests (DBGET, 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/XL for the remote computer which contains the database are those described in the preceding section.

On the local computer running the user application program, TurboIMAGE/XL constructs and uses a structure called the Remote Database Control Block (DBR). One DBR is created each time a user's process issues a DBOPEN accessing a remote TurboIMAGE/XL database (each access path to a remote database); this DBR is released when the corresponding DBCLOSE is issued. The DBR resides in a privileged, unnamed temporary file associated with the user application process on the local computer, and contains database, set, and item information plus the work areas necessary to set up communication with the remote computer. Returned data and status information is also processed in the DBR and is transferred to the appropriate user stack areas before TurboIMAGE/XL returns to the local calling process.

Access to a TurboIMAGE/V database from an MPE/iX system or access to a TurboIMAGE/XL database from an MPE V system is allowed provided both systems are configured with Network Services (NS3000). To access an IMAGE/3000 database, an MPE V system (with NS/3000 and TurboIMAGE/V) acting as an intermediary is required. However, if the limits on the remote database exceed those allowed on the local system, access will not be successful. This is because the DBR will be too small to handle remote capacities that exceed the local limits.

Control Block Sizes

It is not necessary to predict the exact length of the control blocks used by TurboIMAGE/XL to manage user accesses to databases. However, the exact length of the DBU and the exact current length of the DBG are returned in the status array by DBOPEN. These lengths can be truncated to 32767 if the control block is greater than 32767.

Internal Techniques

Although it is not necessary to know the following techniques to use TurboIMAGE/XL, an understanding of them can help you design a more efficient database.

Primary Address Calculation

TurboIMAGE/XL employs two distinct methods of calculating primary addresses in master data sets. The intent of the first method is to spread master entries as uniformly as possible throughout the record space of the data file. This method is referred to as "hashing" and applies only to master data sets with key items of type U, X, Z, or P. In this case, the entire key item value regardless of its length is folded into a positive 32-bit value. This value is reduced modulo the hashing capacity and then incremented by one to form a primary address. The hashing capacity is the maximum capacity for a set not enabled for dynamic expansion, or initial capacity for a set enabled for dynamic expansion of a new database, or the original capacity (excluding expansions) for a set enabled for dynamic expansion of an existing database.

NOTE Because values are folded from right to left, applications should place values in a type U, X, Z, or P field as follows:

- Values that change should be placed first in the field.
- Values that are static should be placed last in the field.

The intent of the second method is to permit you to control the primary address assignment. This method applies only to master data sets with key items of type I, J, K, R, or E. In this case, the right-most block of 32 bits of the key value is treated as a double integer. (If its sign bit is not zero, it is set to zero. If the key item is only 16 bits long, a 32-bit value is created by prefacing these 16 bits with 16 zero bits.)

This 32-bit integer is decremented by 1 and reduced modulo the data set hashing capacity after which 1 is added to the result to form the primary address. If the application provides key values whose right-most 32 bits take on values between 1 and N (where N is no greater than the data set hashing capacity), the corresponding entries will be assigned primary addresses 1 through N which is identical to the Direct Access Method (DAM). In this event, there are no secondaries and performance is outstanding. However, if the application has no control of the key value assignment and/or if N exceeds the hashing capacity, secondaries will occur; this, along with the clustering typical of this method, may

result in poor DBPUT performance. This method should be used only if you have determined that the potential poor performance consequences cannot occur.

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.

NOTE In general, a master data set with a capacity equal to a prime number or to the product of two or three primes can yield fewer synonyms than master data sets with capacities consisting of many factors. Refer to appendix C for a list of prime numbers. More information on dynamic data set expansion is given earlier in this chapter.

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 that has one or more secondary entries is deleted. Because 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/XL 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 free space counter resident in the user label of the data set, by each bit map that monitors each block of the data set, and by enabling of the data set for dynamic expansion.

When a data set is enabled for dynamic expansion, the expansion is triggered at run-time during DBPUT (implied for automatic master) when the data set has reached its almost-full capacity. Once an expansion is done, the master data set is partitioned into two areas: the original area corresponding to the original initial capacity (hashing) and the expansion area corresponding to all extents allocated as a result of run-time expansion. The original area has a mixture of primary and secondary entries. The expansion area has only secondary entries which are added using a delete chain head and end-of-file pointer (high water mark), similar to detail data sets.

The master data sets which are not enabled for dynamic expansion have only one area which contains both primary and secondary entries.

When a new entry is added, TurboIMAGE/XL decrements the free space counter and sets the bit corresponding to the newly assigned record address to a one. 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. If this existing entry is a primary entry, a search is done to find a free location, secondary address, for the new entry. However, if the existing entry is a secondary entry, this secondary entry is relocated to another free secondary address, and the new entry is added at this location. If the data set is enabled for dynamic expansion, the search for a free secondary address is done in the original as well as expansion area. For the original (primary, hashing, or initial) area, a secondary address is identified by a serial search of the bit maps of blocks for a zero indicating an unused record. For the expanded area, a secondary address is identified using the pointer to a delete chain and end-of-file pointer, as in detail data sets.

Space Allocation for Detail Data Sets

Space allocation for each detail data set is controlled by a free space counter, an end-of-file pointer and a 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 contains 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/XL assigns to it the record address referenced by the delete chain pointer, unless the pointer is zero or HWMPUT has been enabled either using `DBCONTROL` or using `DBUTIL`. If the delete chain pointer is zero or if the HWMPUT flag is enabled, 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 a new entry is to be added to a detail data set and the free space counter is zero, at run-time, the data set is expanded according to the capacity expansion parameter specified for this data set (if any). The capacity expansion parameters can be set using DBSCHEMA for new databases or by using `DBCHANGE` Plus or other third-party utilities for existing databases.

When an existing entry is deleted, its media record is zeroed, the first word is replaced with the current delete chain pointer, and the block is written to disk. 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 of reusable media record space. Reusable space is allocated in preference to the unused space represented by record addresses beyond the end-of-file pointer except when HWMPUT is enabled.

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/XL maintains a set of buffer partitions in the DBB for all users of an open database. DBFIND, DBGET, DBUPDATE, DBPUT, and DBDELETE locate a buffer header from one of these partitions.

Each partition is allocated its own buffer header pool, hash table, and free list. The buffer header pool is a set of buffer headers allocated for the accessors of its corresponding partition. The hash table consists of linked lists of buffer header addresses either in use or ready to be released. The free list is a linked list of available buffer headers. Initially, when the DBB is created, all of the buffer headers belonging to a partition are linked to a free list and all the hash table chains are empty. TurboIMAGE/XL uses a two-level hashing algorithm based on the block number of the data set to determine the partition number as well as the hash table entry to be used.

When an intrinsic issues a request for a data set block, the buffer manager starts the search from its hash table entry. If the hash table chain is empty, it acquires a buffer header from the free list. The buffer header is first allocated from the free list to build the buffer header for the data set block and link it to its appropriate hash table chain. When the hash chain, as well as the free list search, is exhausted, the process pauses to wait for other processes to release buffers then retries the buffer header pool scan.

Locking Internals

Within the DBG is a large lock area that provides space for the entries described below.

Accessor Entries

One of these is created for each successful call to DBOPEN (each access path). Although located in the lock area, each accessor entry is the link with which TurboIMAGE/XL controls access to the database. An accessor entry is deleted when DBCLOSE is called for the access path, and the space is reused.

Set Entries

One of these 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 database. These entries are *never* deleted.

Descriptor Entries

These entries contain the internal form of the lock descriptors specified in locking mode 5 or 6. They disappear when the locks are released (when DBUNLOCK is called) and the space is reused.

In addition to DBG, the global database lock table, TURBOLKT, is used to avoid deadlocks by IMAGE/SQL users, and from TurboIMAGE/XL users (if deadlock detection is activated by DBCONTROL mode 7). The TURBOLKT contains information pertaining to every lock by every user on the system. In the event of a potential deadlock, an error is returned instead of causing a system hang.

MPE/iX Transaction Management

When AUTODEFER is not enabled, TurboIMAGE/XL uses an internal MPE/iX service, called Transaction Management (XM), to ensure physical consistency of the database, to reduce physical disk I/O, to perform Intrinsic Level Recovery (ILR), and to perform dynamic roll-back recovery.

When XM is used, modifying intrinsics (DBPUTs, DBDELETEs, and DBUPDATEs) are bracketed as XM transactions. XM logs these transactions in its own XM log file. The **volume set** is the XM unit of logging and recovery. Each volume set has one XM log file.

In default recovery mode, TurboIMAGE/XL uses XM to ensure the structural integrity of the database. The writes of each database modification (every DBPUT, DBUPDATE and DBDELETE) are bracketed as an XM transaction and logged to XM log file pages. XM log file pages are written to disk when one of the following occurs:

1. A system-specified time has elapsed.
2. A request is made by a subsystem, such as, TurboIMAGE/XL, or another process to flush the log file to disk.
3. The XM buffer (or log file pages) is full.

Thus, intrinsics that have completed may not yet be written to disk. If a system failure occurs, only completed modifications that have been written to disk are recovered.

When the operating system is restarted, the physical integrity of the database is recovered via XM recovery. However, the "user defined" logical transactions affecting logical consistency and needing recovery are not processed at this time. Instead, they are posted to a hidden global file called the "AUX" file. When the very first DPOPEN is processed on the system, this "AUX" file is processed, and the transactions pertaining to the same database are posted to their respective "00" file (*dbname*00). Subsequently, when the first DBOPEN for a specific database is done, its "00" file will be processed to restore the logical consistency of the database.

A dynamic transaction is denoted by the DBXBEGIN and DBXEND intrinsics. With dynamic roll-back recovery, a dynamic transaction is rolled back with a program abort or with a call to DBXUNDO. DBXUNDO can be called when other database activity is occurring.

ILR is similar to TurboIMAGE/XL in that it also uses XM. When ILR is enabled, changes to data sets are bracketed as XM transactions and logged to XM log file pages. With ILR, however, XM log file pages are written to disk at the end of each completed DBDELETE and DBPUT. If a system failure occurs, only one DBPUT or DBDELETE will not be recovered. (A completed DBUPDATE does not force a log write to disk.)

When AUTODEFER is enabled, MPE/iX Transaction Management is not used. Instead, AUTODEFER uses the MPE/iX file system defaults which keep data pages in memory until lack of memory or the closing of a file forces the pages to be written to disk. In this mode, a system failure can cause the loss of database integrity *unless roll-forward recovery is used*.

System Limits

The following is a table that shows the current limits in TurboIMAGE. The left column describes the resource(s) based on the unit. The right column is the maximum number it can have. For example, ‘Opened databases/system’ means ‘numbers of databases that can concurrently be opened on a system’; ‘Data items/database’ means ‘numbers of items that can be defined within a database’.

Table 10-1. Static System Limits

Resource	Limit
Opened databases / system	2004
Data items / database	1023
Data sets / database	199
Passwords / database	63
Data items / data set	255
Paths / data set	16
No. of chunks / data set	99
Locked item / data set	1
Data entries / data set	2 ** 31 – 1
Data entries / chain	2 ** 31 – 1
DBOPENs / process	127
DBOPENs / process / database	63
Remote DBOPENs / process	127
Remote DBOPENs process / database	63
Sub item count / item	255
Sub item length	510 bytes
Item length	4096 bytes
Entry length	4756 bytes
Block length	5120 bytes
Data set size	80 GB
Databases / transaction	15
Lock descriptors size / DBLOCK	4096 bytes
Dynamic transaction size / process	4MB
Dynamic transaction size / volume set	64MB
B-tree index key size	252 bytes

11 B-Tree Indices

This chapter is new with the release which includes the enhancement to support B-Tree indices (C.07.00). This chapter gives an overview of the B-Tree index enhancement. It addresses changes in TurboIMAGE/XL utilities and intrinsics, and presents a quick start for using B-Tree indices.

This chapter has the following major sections:

- Overview of B-Tree Indices
- External Commands and Utilities Affected
- Limits
- Quick Start for Using B-Tree Indices

Overview of B-Tree Indices

An index on an item allows generic and range searches. To optimize the search of an index, a technique commonly know as "Binary Tree Searching" is used. Hence, the name **B-Tree index** is used to refer to an index in TurboIMAGE/XL.

You can create a B-Tree index only on the master data set's key item. Nevertheless, you are able to perform index searches using all of its corresponding detail data set search items as well. A master data set key item is perceived as having an explicit B-Tree index and all of its corresponding detail data set search items are perceived as having implicit B-Tree indices. The index searches are done using `DBFIND`. If you create an index on a key item of the master data set, you can use this master data set in the `DBFIND` intrinsic.

Terminology

Explicit B-Tree index is an index actually created on the key item of the master data set. The index searches can be done using this key item in the `DBFIND` intrinsic.

Implicit B-Tree index is an implied index, which does not physically exist, but index search is allowed on the detail data set search item whose corresponding master data set key item has a B-Tree index.

B-Tree DBFIND is a DBFIND on a master or detail data set using a key item or a search item that has a B-Tree index (explicit or implicit). For a master set, it is a set of entries in the master which satisfy the DBFIND criteria. For a detail data set, DBFIND is also referred to as a **super-find** which locates a set of master data set entries, all of which

	have keys that satisfy the DBFIND criteria. The total of all corresponding chains in a detail data set is referred to as a **super-chain**. TurboIMAGE/XL fetches each qualifying master entry in order to determine the total number of associated detail entries to calculate the sum of the chain counts. To retrieve all qualified entries in a detail data set using DBGET, TurboIMAGE "walks" (traverses) this super-chain.
B-Tree Search	is the same as B-Tree DBFIND.
Super-chain	is the sum of all detail data set chains involved in the B-Tree DBFIND.
Trailing-@ Search	is a B-Tree search where only the characters left of the @ (wildcard character) of the argument are compared to qualify an entry. (For example, "cat@" is a trailing-@ search, but "cat@dog" is not and the result will be the same for both arguments. That is, "dog" will be ignored.)
Wildcard Character	refers to a printable ASCII character. The default is @ which means "matches all trailing characters" (unlike "?" or "#" in MPE terms).
BTREEMODE1	is an option which can be turned ON or OFF using the SET command of DBUTIL or programmatically using DBCONTROL. When it is ON, DBFIND mode 1 of an item of X or U type having a B-Tree index (explicit or implicit), and having a wildcard character in the argument will result in a B-Tree search. The ON option allows applications to benefit from B-Tree indices without modifying applications. When it is OFF, which is also the default, DBFIND mode 1 described above works as it did in releases prior to a release with the B-Tree index feature (C.07.00).
Simple Argument	is a DBFIND argument that consists of a sequence of bytes, as in DBFIND mode 1 argument in releases prior to a release with the B-Tree index feature (C.07.00). This is used predominantly for modes 1, 10, and 21. For DBFIND modes 1 and 21, an X or U type item having B-Tree index (explicit or implicit) and having the BTREEMODE1 option ON, the text is scanned for a wildcard character, if any. If a wildcard is found, DBFIND is treated as a B-Tree DBFIND. Otherwise, it is treated as DBFIND in releases prior to a release with the B-Tree index feature (C.07.00).
Structured Argument	is a DBFIND fixed format argument construct introduced with B-Trees enhancement. It is defined as a record structure, containing search control information as well as key data. This is used for DBFIND modes 4 and 24, and is described later in this chapter.

Key Points

These are the key points of TurboIMAGE/XL B-Tree indices:

- You can create a B-Tree index only on the key item of the master data set.
- Although you can create a B-Tree index only for the key item of a master data set, you can still perform a B-Tree search using the search items of all of its corresponding detail sets as well. A master data set is perceived as having an **explicit B-Tree index** and each of its corresponding detail data sets is perceived as having an **implicit B-Tree index**.
- You may create B-Tree indices for zero, one, or more data sets.
- DBSCHEMA has a new option, INDEXED, for the SET specification.
- DBUTIL has new commands and options.

  a. New ADDINDEX, DROPINDEX, and REBUILDINDEX commands (to be used for one, more than one, or all masters)

  b. The SET command has a new BTREEMODE1 option to set DBFIND mode 1 access ON or OFF for a B-Tree wildcard search for X and U types. The default is OFF. The ON setting allows you to use B-Tree indices for generic search without making application changes.

  c. The SET command allows you to define your own database-wide wildcard character.

  d. CREATE, ERASE, PURGE, SECURE, RELEASE, and SHOW commands include B-Tree index files in their operation.

  e. The MOVE command does not allow moving the index file.

- DBCONTROL has these modes pertaining to B-Tree indices:

13	is for B-Tree index file control. That is, to ADD, DROP, REBUILD, ATTACH, or DETACH a B-Tree index file.
14	is used to set database-wide BTREEMODE1 and wildcard character.
15	sets BTREEMODE1 ON for the current DBOPEN (base parameter), and optionally allows the wildcard character to be set for the current DBOPEN.
16	sets BTREEMODE1 OFF for the current DBOPEN (base parameter).

- DBINFO has these modes pertaining to B-Tree indices:

209	informs whether or not a B-Tree index exists for a master.
113	gives BTREEMODE1 setting and the wildcard character for the database as well as current DBOPEN.

- DBPUT or DBDELETE to an indexed master triggers a similar operation to indexed master's B-Tree index file. (DBUPDATE to a master data set to modify a key item, despite CIPUDATE setting, is not allowed.)
- DBFIND has these features pertaining to B-Tree indices:

  a. Can be used for details as well as masters to specify B-Tree index searches.

b. DBFIND modes with added functionality pertaining to B-Tree indices:

1	can continue to work as it does in releases prior to the release of B-Tree index feature (C.07.00), despite the presence of a B-Tree index, or it can be used for a B-Tree search if BTREEMODE1 is ON.
4	used for B-Tree index searches on numeric as well as ASCII types and returns accurate chain (super-chain) counts. Requires a structured argument.
10	allows you to simulate the DBFIND mode 1 in releases prior to the release with B-tree index feature even when the item has a B-Tree index (explicit or implicit), BTREEMODE1 is on, and the argument contains a wildcard character. It returns accurate chain count. This is the same as TPI mode 10. Requires a simple argument.
21	is the same as B-Tree index search using DBFIND mode 1, except it is a faster version and does not return accurate chain counts. Requires a simple argument.
24	is the same as B-Tree index search using DBFIND mode 4, except it is a faster version and does not return accurate chain counts. Requires a structured argument.

c. Allows wildcard search, as well as range search (< , <=, > , >=,"PK", or [ ] for between).

d. If BTREEMODE1 is ON, DBFIND mode 1 of X or U type item having a B-Tree index (explicit or implicit) and the argument containing a wildcard character will result in a B-Tree index search.

e. For wildcard (generic) search, the wildcard should be the terminating character in the argument. Characters beyond that will be ignored.

f. Status array reflects information based on data set, mode, and search type. B-Tree index searches for modes 1 and 4 for details give super-chain (multiple detail chain) counts and record numbers of first entry in first detail chain and last entry in last detail chain.

- DBGET modes 5 and 6 can be used for B-Tree index retrieval for masters or details. Super-chains are traversed for detail data sets.
- KSAM/iX files are used for B-Tree index files. These are KSAM related key points:

a. There is one KSAM/iX privileged file with a filecode of -412 for each B-Tree index file. The size limit for this B-Tree index file is 4 Gbytes. A jumbo master (larger than 4 Gbytes) can have a B-Tree index file provided the B-Tree index file remains within its 4 Gbyte limit.

b. KSAM/iX B-Tree index file is named using the POSIX file format with the "idx" extension (lowercase). For example:

```
/ACTSALES/GRPSALES/ORDERS03.idx
```

c. KSAM/iX B-Tree index file records consist only of a single key, which is a duplicate of

the master data set key value. No pointer information is present in the B-Tree index records.

d. The KSAM file has the Native Language Support language specified to match the language of the database, if the key is a text (X or U) data type.

- Record zero of the root file contains "C4" for the root file version if at least one B-Tree index file exists for the database. When all B-Tree indices are dropped, it reverts to the appropriate version: "C3" if at least one jumbo set exists, "C2" otherwise.
- A new bit map is added in the root file for B-Tree indices.
- Third-party indices can coexist with B-Tree indices, that is, on the same item of the data set.

External Commands and Utilities Affected

The root file, utilities, and intrinsics affected by the implementation of B-Tree index files include the following:

- Root file
- DBSCHEMA
- DBUTIL
- DBCONTROL
- DBFIND
- DBGET
- DBINFO
- DBPUT
- DBDELETE
- DBUPDATE

Root File

The root file version level will be "C4" if at least one master data set has a B-Tree index. If there are no B-Tree indices, the root file version will be "C3" if any jumbo data sets exist, and "C2" otherwise. In other words, a value greater than "C2" indicates that at least one POSIX named file exists as part of the database.

DBSCHEMA

DBSCHEMA has an INDEXED option added to the data set NAME specification.

New Syntax for DBSCHEMA

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

Example

```
NAME:  Employeename, MANUAL /INDEXED(10,20/30);
```

DBUTIL

DBUTIL has three new commands:

- ADDINDEX
- DROPINDEX
- REBUILDINDEX

Syntax for ADDINDEX

ADDI[NDEX] *database name* [/*maintword*] FOR { ALL
setnamelist
setnumlist }

Syntax for DROPINDEX

DROPI[NDEX] *database name* [/*maintword*] FOR { ALL
setnamelist
setnumlist}

Syntax for REBUILDINDEX

REBUILDI[NDEX] *database name* [/*maintword*] FOR { ALL
setnamelist
setnumlist}

Parameters

setnamelist	is the list *setname*[,...]
setnumlist	is the list *setnum*[,...]
ALL	means all master data sets for the database.

Examples

```
>>ADDINDEX  ORDERS/secret for ALL

>>DROPINDEX ORDERS/secret for 1,7

>>REBUILDINDEX ORDERS/secret FOR SalesrepName,Region,District
```

Discussion

The ADDINDEX command adds the associated B-Tree index file and updates the root file.

The DROPINDEX command drops the associated B-Tree index file and updates the root file.

REBUILDINDEX rebuilds the index file for a master data set that has an index file.

The KSAM file has the Native Language Support language specified to match the

language of the database, if the key is an ASCII (X or U) data type.

When using the ALL option and there is no master data set, a warning is generated, but the command is considered to be successful.

Other DBUTIL Commands

The following lists the impact of B-Tree indices on other DBUTIL commands:

- CREATE does an implicit ADDINDEX command for each data set marked by DBSCHEMA as indexed.
- ERASE erases any associated B-Tree index (.idx) files, but will not delete them.
- MOVE does NOT allow a B-Tree index (.idx) file to be moved.
- PURGE purges any associated B-Tree index (.idx) files.
- SECURE and RELEASE apply to the associated B-Tree index (.idx) files.
- SET has a new option, BTREEMODE1.
- SHOW has a new option: INDEX, INDEXES, or INDICES.

New Syntax for SET

SET *database name* [/*maintword*] BTREEMODE1={ ON
OFF} [,[WILDCARD=]*c*]

where *c* is any printable ASCII character, and the default character is @.

The BTREEMODE1 option sets DBFIND mode 1 access ON or OFF for a B-Tree index search for X and U types. The default is OFF, in which case DBFIND mode 1 with the argument containing a wildcard character will continue to work the way it did in releases prior to a release with the B-Tree index feature (C.07.00). When BTREEMODE1 is ON, DBFIND mode 1 with the argument containing a wildcard character will be treated as a B-Tree index search. Refer to the DBFIND section for more information on BTREEMODE1.

Examples

```
>>SET ORDERS/secret BTREEMODE1 = ON

>>SET ORDERS/secret BTREEMODE1 = ON,%
```

The DBUTIL SHOW ALL command shows if any B-Tree index files exist and the value of BTREEMODE1. Example output:

```
>> show ORDERS all
For database ORDERS
...
Dynamic capacity expansion is not used.
Database has at least one indexed data set.
BTREEMODE1 is off, wildcard = "@"

Logid is not present.
...
```

Example if no B-Tree index files exist:

```
>> show ORDERS all
For database ORDERS
...
Dynamic capacity expansion is not used.
BTREEMODE1 is off, wildcard = "@"

Logid is not present.
...
```

New Syntax for SHOW

SHOW has a new option INDEX, alternatively INDEXES or INDICES.

SHOW *database name* [/*maintword*] INDEX

```
>>SHOW ORDERS INDEX
  For database ORDERS
  Database root version < "C"4; there are no indexes.

>>addindex orders for all
   Found 4 master datasets.
   Adding index to set# 1 (#entries = 162,730, capacity = 218,987)
   Adding index to set# 2 (#entries =  84,164, capacity = 188,517)
   Adding index to set# 3 (#entries =  18,784, capacity =  21,943)
   Adding index to set# 4 (#entries =     783, capacity =    2583)
Done.
>>

>>SHOW ORDERS INDICES
  For database ORDERS

  Data Set Name Type Indexed?

  DATE-MASTER      A     YES
  CUSTOMER         M     YES
  PRODUCT          M     YES
  SUP-MASTER       M     YES

  4 indexed datasets
>>
```

DBCONTROL

DBCONTROL allows DBUTIL and privileged callers to perform several B-Tree index file related functions such as the addition, deletion, or rebuilding of a B-Tree index file for a specified master data set. Some functions require exclusive database access.

There are four additional DBCONTROL modes pertaining to B-Trees:

Mode 13	B-Tree index file functions.
Mode 14	obtain/control database-wide B-Tree index information.
Mode 15	sets BTREEMODE1 option ON and optionally sets the wildcard character for the current DBOPEN.
Mode 16	sets BTREEMODE1 option OFF for the database for the current DBOPEN.

NOTE General use of privileged mode can seriously damage your system, if not used wisely.

Mode 13

Mode 13 is used to perform functions related to B-Tree index files. The caller must be privileged. Exclusive database access is required for adding, dropping, and rebuilding B-Tree index, that is functions 1, 2, and 3. Qualifier has a structured record containing data set information and directives. This is the qualifier layout (an element is 16-bits):

Element	Contents	
1	Function code:	
	0	mode 13 inquiry (returns "OK" in status indicating mode 13 is a valid mode for the TurboIMAGE/XL version on the system. It also means that the B-Tree index feature is available in the version of TurboIMAGE/XL. Ignores the rest of the record).
	1	add B-Tree index file.
	2	drop B-Tree index file.
	3	rebuild B-Tree index file.
	4	attach XM to B-Tree index.
	5	for internal use only.
	6	release B-Tree index file (file system security)
	7	secure B-Tree index file (file system security)
2	ignored when function code is 0 in element 1 data set number (1..199) when function code is 1 through 7 in element 1	
3-4	ignored on input, zeroed at entry to DBCONTROL mode 13, and used to return a 32-bit status at exit if at least one of the internal B-Tree index file routines was executed.	
5	flags for internal use only: bit 15: 1 = report progress, if appropriate, when function code is 1 through 7 in element 1	
6-16	reserved; should be 0.	

Mode 14

Mode 14 is used to set BTREEMODE1 option ON or OFF for the database and set the wildcard character for the database. The changes will be made in the root file, unlike mode 15 and 16 of DBCONTROL. The caller must be privileged. The result is same as using the SET command with BTREEMODE1 option of DBUTIL. This is the qualifier layout (16 halfwords):

Element	Contents	
1	Function code:	
	0	mode support inquiry (returns "OK" in status, and ignores the rest of the record). That is, to validate if mode 14 is supported in the TurboIMAGE software on the system.
	1-6	reserved for internal use.
	7	set wildcard (in root file, not DBU). Wildcard is in lower 8 bits of the second element of qualifier. An error is returned if the value is less than ASCII 33, or greater than ASCII 126.
	8	set BTREEMODE1 option ON (in root file, not DBU).
	9	set BTREEMODE1 option OFF (in root file, not DBU).
2	ignored for function codes 0-6, 8, or 9. wildcard in lower 8 bits for function code 7.	
3-4	ignored on input, zeroed at entry to DBCONTROL mode 14, and used to return a 32-bit status at exit ONLY IF the B-Tree index routines are executed. (This is useful for debugging purposes only.)	
5-16	reserved; should be 0.	

Mode 15

Mode 15 sets BTREEMODE1 ON for the current database. It examines the first byte of the qualifier. If it is null or blank, then the current wildcard character is not changed. If it is in the ASCII range (33..126), then the wildcard character is changed to that value for the current database open. If the qualifier byte is any other value, DBCONTROL returns an error. Mode 15 affects just the current database open, not the root file. The caller need not be privileged. The setting remains in effect until the database is closed or the application terminates.

Mode 16

For mode 16, the qualifier is ignored, and BTREEMODE1 is set to OFF for the current database open. Mode 16 affects just the current database open, not the root file. The caller need not be privileged. The setting remains in effect until the database is closed or the application terminates.

DBFIND

Prior to a release with the B-Tree index feature, DBFIND required the caller to specify detail data set on the qualifier and a search item in the item parameter. It used that pair of data to determine the master set to search for the exact value in the item parameter. The B-Tree index feature extends DBFIND to allow a master data set as well as the key item in that data set. Also, it allows for searches other than just equality. This is useful for obtaining sorted access by key value to entries in a master data set as well as detail data set.

Supported Modes of DBFIND

These are modes of DBFIND supported by the B-Tree index feature:

1	do a B-Tree DBFIND if all of the following are true: • BTREEMODE1 is on. • The item type is X or U. • The item has a B-Tree index (explicit or implicit). • The argument contains wildcard character. Otherwise execute DBFIND as done without this feature. For B-Tree DBFINDs, the argument is scanned for the first occurrence of the wildcard character, and a trailing-@ search is done. Subsequent text in the key value is ignored. For B-Tree DBFINDs pertaining to detail data sets, the chain-count is accurate, and status halfwords 7-8 and 9-10 give the record numbers of the last entry in the last chain, and the first entry in the first chain of the super-chain. For B-Tree DBFINDs pertaining to master data sets, the status halfwords 5-6 (chain-count for detail) reflect the total number of entries qualified in the master data set. All other fields of the status array return zeroes.
4	do a B-Tree index search, and give accurate chain counts and record numbers in the status array. For masters, the status array returns the same information as in B-Tree DBFIND mode 1. The argument is in structured format described later.
10	do a non-B-Tree DBFIND mode 1 equivalent. This gives the same result as if there were no B-Tree index.
21	is the same as mode 1, above, but the chain count and record numbers for last entry and first entry are not accurate. (This is a high-speed version of mode 1.) For detail data set, the halfwords 5-6, 7-8, and 9-10 have 2^{31}–1. For master data set, the halfword 5-6 has 2^{31}–1 and 7-8 and 9-10 have zeroes. Following this DBFIND, you may do DBGET mode 5 or 6.
24	is the same as mode 4, above, but the chain count and record numbers for last entry and first entry are not accurate. (This is a high-speed version of mode 4.) For detail data set, the halfwords 5-6, 7-8, and 9-10 will have 2^{31}–1. For master data set, the halfword 5-6 has 2^{31}–1 and 7-8 and 9-10 have zeroes. The argument is in structured format described later.

TPI Modes of DBFIND Not Supported for B-Tree Indices

Third-Party Indexing (TPI) adds more DBFIND modes. The TPI modes 1 and 10 are similar to B-Tree index modes 1 and 10.

NOTE These are the known Third Party Indexing (TPI) modes of DBFIND which are **NOT supported** for B-Tree searches:

11	do a B-Tree index DBFIND when the key is binary.
12	is a keyword search.
other	1nn, 2nn, 3nn, 4nn, and 5nn are other unsupported TPI DBFIND modes.

A B-Tree index search will never be done with a DBFIND mode 1 style argument if the key item is binary (not X or U) (this affects DBFIND modes 1 and 21). For a B-Tree index search on binary (non-ASCII) data, use modes 4 or 24 in conjunction with a structured argument.

A DBFIND mode 21 on a non-ASCII item returns an error.

All TurboIMAGE data types are allowed as keys for B-Tree searches.

DBFIND Modes

Following is a DBFIND mode summary table (see footnotes for explanations):

Table 11-1. DBFIND Mode Summary Chart

DBFIND\Mode	B-Tree index \search?	Accurate\Chain counts?	Argument\style
1	MAYBE[a]	YES[b] or YES[c]	mode 1[d]
4	YES	YES[c]	mode 4[e]
10	NO	YES[b]	mode 1[d]
21	YES[f]	NO[g]	mode 1[d]
24	YES	NO[g]	mode 4[e]

a. If BTREEMODE1 is ON, a DBFIND mode 1 on an ASCII item with a B-Tree index (explicit or implicit), and mode 1 style argument contains a wildcard character, it will be treated as a B-Tree index search. If BTREEMODE1 is OFF, a DBFIND mode 1 in the above scenario will be treated as a non-B-Tree index search (as if there were no B-Tree index feature). DBFIND mode 1 on binary items (not X and U) will be treated as non-B-Tree search regardless of the presence of a B-Tree index as well as the BTREEMODE1 option. To do B-Tree searches on binary items, use modes 4 or 24 in conjunction with a structured argument.

b. The chain count is the number of entries in the single chain (non-B-Tree DBFIND). Record number for the last entry and first entry is obtained from the chain head.

c. The chain count is the sum of all chain counts (that is, the number of entries in the super-chain for a B-Tree DBFIND). Record number for the last entry is obtained from the last entry in the last chain. For first entry, record number of the first entry in the first chain is obtained. For masters, the chain count reflects the total number of master entries qualified, and last entry and first entry values are zeroes.
d. When doing a B-Tree index search, mode-1-style argument is scanned to find the *first* occurrence of the wildcard character in the argument text. If the wildcard is not found, a non-B-Tree index search is done. If the wildcard is found, the rest of the argument text is ignored. When not doing a B-Tree search (mode is 1 and `BTREEMODE1` is OFF, or mode is 10), the entire argument, including any wildcard characters, will be treated as the actual argument is for a `DBFIND` mode 1 in releases prior to the release with B-tree index feature.
e. See "`DBFIND` Structured Argument" description later in this chapter.
f. For items that are text types (X, U), a B-Tree index search is done if the wildcard is present in the argument. If the key item is a non-ASCII item, then a B-Tree index find is not done, and a non-B-Tree index find (mode 1 with `BTREEMODE1` is OFF) is done instead. A programmer can do a B-Tree index find with non-ASCII items by explicitly using `DBFIND` modes 4 or 24. (See "`DBFIND` mode 1-style and non-ASCII keys" below.) This mode is similar to current TPI mode 21.
g. For detail data set, the halfwords 5-6, 7-8, and 9-10 have 2^{31}–1. For master data set, the halfword 5-6 has 2^{31}–1 and 7-8 and 9-10 have zeroes.

NOTE The length of the argument may not exceed the item length.

DBFIND Arguments

There are two distinct argument styles. The simple argument of the basic `DBFIND` mode 1 argument is generally interpreted as a sequence of bytes, perhaps containing a wildcard character.

Another argument style, the **structured argument**, has been defined for `DBFIND` modes 4 and 24. This structure allows ranges of item values to be requested, and is explained below.

DBFIND Structured Argument

The structure of the **structured argument** for `DBFIND` modes 4 and 24:

Bytes	Meaning	
1-2	Type of generic search. An ASCII character pair is in this field:	
	=	search for key values equal to argument1
	<	search for key values less than argument1
	<=	search for key values less than or equal to argument1
	>	search for key values greater than argument1
	>=	search for key values greater than or equal to argument1
	[ ]	search for key values greater than or equal to argument1 AND less than or equal to argument2

	@*c*	wildcard search. Scan argument for the first wildcard character. (Call that character position *n*, 1-based). Search for keys that match first *n*–1 characters of argument. If *c* is non-blank and non-null, then it is the wildcard character that will be used. Some examples are: @* and @@. If *c* is a blank or null, then the current default wildcard (stored in the root file) is used. The wildcard character is changeable via the DBUTIL SET command or DBCONTROL mode 15.
	PK	Partial Key search. Search for key values that match *n* characters in argument1 (*n* is length of argument1 provided in bytes 5-6). Argument1 need not contain a wildcard. If it does within the *n* characters, it will be included in the search. For example, if argument1 is ABC@, bytes 5-6 have 4 for length, and the wildcard for the database is @, DBFIND will return records containing ABC@ as the first four characters in the key value.
3-4		version number. It must be numeric zero, or an error will be returned.
5-6		The size (in bytes) of argument1 (not including these two bytes) for search types < , <=, = , >=, > , @*c*, PK.
7-8		The size (in bytes) of argument2 (not including these two bytes) for the between search type [ ].
9... 9+n-1		Argument1. The *n* bytes of argument data (for example, for an X10 field, *n* = 10).
9+n... 9+n+m-1		Argument2. For search-type [ ] only. The *m* bytes of the second argument's data (for example, for an X10 field, *m* = 10, *n* must match *m*). Must be numeric zero for other search types, or an error is returned.

If a wildcard character is present in the argument(s), the wildcard will be considered as part of the value for these B-tree search types: = , < , <=, > , >=, [ ], and PK on ASCII types.

Pascal/iX Example

A Pascal/iX view of the above is:

```
type
 dbfind_structured_arg_type = $alignment 2$ record
    dbf_type                : pac2;        {e.g., "<="}   {0 @ 2}
    dbf_version             : shortint;    {must be 0}    {2 @ 2}
    dbf_arg1_bytes          : shortint;                   {4 @ 2}
    dbf_arg2_bytes          : shortint;                   {6 @ 2}

        {NOTE: arg1 data is variable sized...2 to 256 bytes}
        {      and, arg2 data might not even be present.   }
        {      Still, the following serve to define a record}
        {      that can hold the worst case arg1 & arg2... }

    dbf_arg1_data           : packed array [1..256] of    {8 @ x}
                                      char;
```

```
        {*REAL* dbf_arg1_data is variable sized}
        {*REAL* dbf_arg2_data is variable sized}

    spare_area              : packed array [1..256] of    {8 @ x}
                                        char;
        {if present, arg2 data starts at record + 8 + arg1_bytes}

    end;        {8 + 256 + 256}
```

Examples of Structured Argument

Three examples of using the structured argument type are given below.

Example 1

X4 field, looking for <= "JOHN", using SPL:

```
move arg' := ("<=", 0, 4, 0, "JOHN");
```

Example 2

X4 field, looking for keys >= JOHN, <= STAN, using Pascal:

```
type
   structured_argument_type = crunched record
      search_type           : pac2;        {bytes 0, 1}
      version               : shortint;    {bytes 2, 3}
      arg1_bytes            : shortint;    {bytes 4, 5}
      arg2_bytes            : shortint;    {bytes 6, 7}
      arg1                  : pac4;
      arg2                  : pac4;
      end;

var
   arg        : structured_argument_type;

fast_fill (addr (arg), 0, sizeof (arg));    {optional}

arg.search_type := '[]';           {in-range search}
arg.arg1_bytes := sizeof (arg.arg1);
arg.arg1 := 'JOHN';
arg.arg2_bytes := sizeof (arg.arg2);
arg.arg2 := 'STAN';

mode4 := 4;
item := 'FIRSTNAME;';

dbfind (base, set, mode4, status, item, arg);
```

Example 3

X20 field, looking for keys that start with "SMITH", using C:

```
char
   arg         [29];       /* 2 + 2 + 2 + 2 + 20 + 1 trailing null */

arg [0] = '@';             /* want a wildcard search */
arg [1] = '@';             /* use an @ as a wildcard character */
```

```
arg [2] = (char) 0;        /* upper half of version */
arg [3] = (char) 0;        /* bottom half of version */
arg [4] = (char) 0;        /* upper half of arg1 size field */
arg [5] = (char) 20;       /* bottom half of arg1 size field */
arg [6] = (char) 0;        /* upper half of arg2 size field */
arg [7] = (char) 0;        /* bottom half of arg2 size field */

strcpy (arg [8], "SMITH@");
```

Example 4

I2 field, looking for keys >= 123 and <= 45698, using SPL:

```
double array
   arg'd       (0 : 3);    ! 4 byte prefix + three I2 fields

integer array
   arg'i       (*) = arg'd;

integer
   mode4       := 4;

byte array
   arg'        (*) = arg'd;


      ! find super-chain for SALARY in range [123..45698]...

arg'i := "[]";                      ! range search
arg'i (1) := 0;                     ! version
arg'i (2) := 4;                     ! size of arg1 (in bytes)
arg'i (3) := 4;                     ! size of arg2 (in bytes)
arg'd (2) := 123d;                  ! arg1
arg'd (3) := 45698d;                ! arg2

move item := "SALARY;";
dbfind (base, set, mode4, status, item, arg'i);
```

DBGET

DBGET has no changes to the calling sequence. It has the semantic change that chained access modes can be used for both masters and details. For master data sets, DBGET mode 5 or 6 following a B-Tree DBFIND implies the next or previous qualified record of the master data set. Also, DBGET mode 5 (forward chained read) and mode 6 (backward chained read) may now traverse super-chains for a B-Tree index DBFIND on a detail data set.

NOTE After a super-chain (B-Tree index DBFIND) read has started, a directed DBGET (mode 4), may perturb the super-chain.

DBINFO

DBINFO has new modes 113 and 209.

Mode 113

DBINFO mode 113 reports the wildcard character in use for the database, and the value of BTREEMODE1. The qualifier is ignored.

The output buffer for mode 113 must be at least 32 bytes in size and is shown below (elements are 16-bits wide and are counted starting with 1):

Element	Contents
1	0 if BTREEMODE1 is off in the root file 1 if BTREEMODE1 is on in the root file
2	The first byte (8 bits) is always 0. The second byte (8 bits) represents *c*, where *c* is the current wildcard character. For example, if the current wildcard character is @, the element's hex value will be $0040, or decimal 64.
3	Highest B-Tree index argument version supported (currently 0)
4	Number of sets with B-Tree indices attached
5	0 if BTREEMODE1 is OFF for current DBOPEN 1 if BTREEMODE1 is ON for current DBOPEN
6	The first byte (8 bits) is always 0 for current DBOPEN. The second byte (8 bits) represents *c*, where *c* is the current wildcard character.
7-16	(reserved)

Mode 209

DBINFO mode 209 reports whether or not a data set has a B-Tree index attached. The qualifier is a data set number or name. The output buffer is required to be 64 bytes (32 halfwords) or larger. The output buffer for mode 209 is shown below (elements are 16-bits wide, and are counted starting with 1):

Element	Contents
1	0 if no B-Tree index exists 1 if B-Tree index exists
2	1 if attached B-Tree is not damaged or index does not exist
3-32	For internal use

DBPUT, DBDELETE, and DBUPDATE

DBPUT and DBDELETE trigger similar operation to the associated B-Tree index file for all adds or deletes to or from a master data set. DBUPDATE to a master is not allowed even when critical item update is on.

Adding a record to the B-Tree index file is done with an FWRITE call. The KSAM files do not have the DUP option, so KSAM will reject FWRITEs that attempt to add a duplicate key value. The KSAM files are built with the REUSE option, which means that the space occupied by deleted records will eventually be reused.

Limits

These are some limits on the use of B-Tree indices:

- Key limits

  KSAM limits keys up to 255 bytes. TurboIMAGE/XL limits keys to 510 bytes (255 halfwords). TurboIMAGE items are always an even number of bytes in length, with 2 bytes for internal use. This means that the maximum TurboIMAGE key that can be indexed is 252 bytes. Attempting to build an index for a master data set whose key is greater than 252 bytes will not succeed, and will generate an error message.

- Maximum number of file opens

  MPE/iX currently limits a single process to opening a maximum of 1023 files at a time. For a database with 199 sets, 100 of which are master data sets with B-Tree indices, if all sets are open at the same time, an additional 100 files will be open.

- Pattern matching

  The only types of generic key pattern matching supported are:

=	search for item values equal to the specified value
<	search for item values less than specified value
<=	search for item values less than or equal to the specified value
>	search for item values greater than specified value
>=	search for item values greater than or equal to the specified value
[ ]	search for item values between and including two specified values.
@c	search for items with a common prefix.
PK	search for items with partial key.

  The operator < > for "not equal to" and the range search ( ) are not supported.

Quick Start for Using B-Tree Indices

If you are interested in generic trailing-@ searches only and want to get started quickly without making any application changes, you may use the following steps.

1. Identify the masters with ASCII key item and can benefit from B-Tree indices.
2. Create B-Tree indices using either of these methods:
   a. Use the INDEXED option of DBSCHEMA for new databases.
   b. Use the ADDINDEX command of `DBUTIL` for existing databases.
3. Set the BTREEMODE1 option ON using `DBUTIL` as follows:

```
:Run DBUTIL.PUB.SYS

>SET database name [/maintword] BTREEMODE1=ON
```

If your database is new, you will need to add data. Then you are ready to perform B-Tree index searches. You can include the wildcard character in your `DBFIND` argument and observe the results.

A Error Messages

TurboIMAGE/XL issues three different types of error messages:

- Schema Processor Error Messages
- Library Procedure Error Messages
- Utility Error Messages

Schema processor error messages result from errors detected during processing of the database schema. The library procedure error messages are returned to the calling program from the library procedures. The utility error messages are caused by errors in execution of the database utility programs.

Schema Processor Error Messages

The Schema Processor accesses three files:

- The text file (DBSTEXT) containing the schema records and Schema Processor commands for processing.
- The list file (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 list file, if the list file is different from $STDLIST).

Schema Processor command errors can 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 chapter 3), will cause the Schema Processor to terminate execution.

Note that if the LIST option is active (see chapter 3), error messages for command errors and syntax errors appear in the list file following the offending statement. If the NOLIST option is active, only the offending statement, followed by the error message, is listed.

Schema Processor File Errors

Various file error messages are listed in this section. 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.

MESSAGE FILE ALREADY EXISTS; UNABLE TO CLOSE *file name*

MEANING FCLOSE error occurred on specified file. Can be caused by duplicate file in group with same name as root file.

ACTION Change database name or purge file of same name. Or, be sure correct file and file name used. Check MPE FILE commands used. If other cause, consult *MPE/iX Intrinsics Reference Manual* for similar message.

MESSAGE READ ERROR ON *file name*

MEANING FREAD error occurred on the specified file.

ACTION Check text file or MPE FILE command.

MESSAGE UNABLE TO USE *file name*

MEANING Specified file cannot be opened with FOPEN, or its characteristics make it unsuitable for its intended use.

ACTION Change database name, or purge file of same name. Or, be sure correct file and file name used. Check MPE FILE commands used. If other cause, consult *MPE/iX Intrinsics Reference Manual* for similar message.

MESSAGE UNABLE TO WRITE LABEL OF *file name*

MEANING FWRITELABEL error occurred on specified file.

ACTION Change database name or purge file of same name. Or, be sure correct file and file name used. Check MPE FILE commands used. If other cause, consult *MPE/iX Intrinsics Reference Manual* for similar message.

MESSAGE UNEXPECTED END-OF-FILE ON *file name*

MEANING Call to FREAD or FWRITE on specified file has yielded unexpected end of file condition.

ACTION Change database name, or purge file of same name. Or, be sure correct file and file name used. Check MPE FILE commands used. If other cause, consult *MPE/iX Intrinsics Reference Manual* for similar message.

MESSAGE WRITE ERROR ON *file name*

MEANING FWRITE error occurred on the specified file.

ACTION Change database name or purge file of same name. Or, be sure correct file and file name used. Check MPE FILE commands used. If other cause, consult *MPE/iX Intrinsics Reference Manual* for similar message.

Schema Processor Command Errors

Various Schema Processor command error messages are listed in this section. Each such message is preceded by the character string:

```
***** ERROR ******
```

MESSAGE COMMAND CONTINUATION NOT FOUND

MEANING 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 ($).

ACTION Examine the schema text file to find any incorrect commands. Edit the text file, and run the Schema Processor again.

MESSAGE COUNT HAS BAD FORMAT

MEANING The numbers in *ERRORS, LINES,* or *BLOCKMAX* parameters of the $CONTROL command are not properly formatted integer values.

ACTION Examine the schema text file to find any incorrect commands. Edit the text file, and run the Schema Processor again.

MESSAGE ILLEGAL COMMAND

MEANING The Schema Processor does not recognize the command. Valid commands are $PAGE, $TITLE, and $CONTROL.

ACTION Examine the schema text file to find any incorrect commands. Edit the text file, and run the Schema Processor again.

MESSAGE IMPROPER COMMAND PARAMETER

MEANING One of the commands in the parameter is not valid.

ACTION Examine the schema text file to find any incorrect commands. Edit the text file, and run the Schema Processor again.

MESSAGE MISSING QUOTATION MARK

MEANING Character string specified in $PAGE or $TITLE command must be bracketed by quotation marks (“).

ACTION Examine the schema text file to find any incorrect commands. Edit the text file, and run the Schema Processor again.

MESSAGE SPECIFIED TITLE TOO LONG

MEANING Character string in $TITLE or $PAGE command exceeds 104 characters.

ACTION Examine the schema text file to find any incorrect commands. Edit the text file, and run the Schema Processor again.

Schema Syntax Errors

Database definition syntax errors can be detected by the Schema Processor. Their existence does not cause termination but does prohibit root file creation. Discovery of one can trigger others which disappear after the first is corrected. Also, detection of one can 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.

Various syntax error messages are listed in this section. As with command errors, each syntax error is preceded by the character string:

```
***** ERROR ******
```

MESSAGE	AUTOMATIC MASTER MUST HAVE SEARCH ITEM ONLY
MEANING	Automatic master data sets must contain entries with only one data item. The data item must be a key item.
ACTION	Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE	BAD BLOCKING FACTOR OR TERMINATOR
MEANING	A bad blocking factor or terminator was specified.
ACTION	Examine the schema text file to find any incorrect commands. Edit the text file, and run the Schema Processor again.

MESSAGE	BAD CAPACITY (MAX, INIT, INCREMENT) OR TERMINATOR
MEANING	One of the capacity parameters (maximum, initial, or increment) is incorrect, or the terminator is not ";".
ACTION	Change the maximum, initial, or increment capacity parameter, and use ";" as the terminator. See "Set Part" in chapter 3 for correct values for the parameters.

MESSAGE	BAD CHARACTER IN USER CLASS NUMBER
MEANING	User class number in password is not an integer from 1 to 63.
ACTION	Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE	BAD DATABASE NAME OR TERMINATOR
MEANING	Database name in BEGIN DATABASE statement is not a valid database name beginning with an alphabetic character and having up to 6 alphanumeric characters. Or, the name is not followed by a semicolon (;).
ACTION	Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE BAD DATA SET TYPE

MEANING The data set type designator is not AUTOMATIC (or A), MANUAL (or M), or DETAIL (or D).

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE BAD DEVICE CLASS NAME

MEANING The device class name specified contains an invalid character. The name must be less than eight characters and begin with a letter.

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE BAD DEVICE CLASS NAME OR TERMINATOR

MEANING The device class name specified contains an invalid character or was not ended with a semicolon ";".

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE BAD LANGUAGE

MEANING Language name contains invalid characters, or the language number is not a valid integer.

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE BAD PATH CONTROL PART DELIMITER

MEANING Data item defined as sort item in detail data set is not properly delimited with parentheses ().

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE BAD PATH COUNT OR TERMINATOR

MEANING The path count in the master data set definition is not an integer from 1 to 16 (for an automatic master), or 0 to 16 (for a manual master). This message can also mean the path count is not followed by a quotation mark (“).

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE BAD PATH SPECIFICATION DELIMITER

MEANING Name of master data set following search item name in detail data set definition is not followed by a right parenthesis ")" or by a sort item name in parentheses.

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE BAD READ CLASS OR TERMINATOR

MEANING 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 (/).

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE BAD SET NAME OR TERMINATOR

MEANING The data set name does not conform to naming rules. (Names must start with a letter and can have up to 16 alphanumeric characters including + – * / ? ə # % & @.) Or the data set name is not terminated by the correct character for the context in which it appears.

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE BAD SUBITEM COUNT OR TERMINATOR

MEANING Subitem count for a data item defined in schema item part is not an integer from 1 to 255.

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE BAD SUBITEM LENGTH OR TERMINATOR

MEANING Subitem length for data item defined in schema item is not an integer from 1 to 255.

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE BAD TERMINATOR-';' EXPECTED

MEANING Password or capacity was not followed by a semicolon.

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE BAD TERMINATOR- ';' OR ',' EXPECTED

MEANING Items within an entry definition must be separated from each other with commas and terminated with a semicolon.

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE BAD TYPE DESIGNATOR

MEANING Data item defined in schema item part is not defined as type I, J, K, R, U, X, Z, or P.

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE BAD WRITE CLASS OR TERMINATOR

MEANING 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.

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE 'BEGIN' EXPECTED

MEANING Missing 'Begin' in the schema.

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE BLANKS WILL BE REMOVED

MEANING A blank '' character in a password has been suppressed before writing the password to the root file.

ACTION Edit and remove.

MESSAGE 'CAPACITY:' EXPECTED

MEANING CAPACITY statement must follow entry definition in the definition of data sets in the set part of schema.

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE DATABASE HAS NO DATA SETS

MEANING No data sets were defined in the set part of schema. The database must contain at least one data set.

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE DATABASE NAME TOO LONG

MEANING Database name has more than six characters.

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE DATA SET SIZE EXCEEDS MPE/XL FILE SIZE LIMITS

MEANING The data set size, which is calculated using the media record length, the capacity, and the blocking factor, exceeds the limitation of the MPE/iX file size.

ACTION Decrease the capacity of the data set. Edit the text file, and run the Schema Processor again.

MESSAGE DUPLICATE ITEM NAME

MEANING A duplicate item name is specified.

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE DUPLICATE ITEM SPECIFIED

MEANING The same data item name was used more than once in the entry definition of data sets.

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE DUPLICATE SET NAME

MEANING The same data set name was used to define more than one data set in the set part of schema.

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE 'ENTRY:' EXPECTED

MEANING 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.

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE ENTRY LENGTH SHOULD BE LESS THAN 2048

MEANING Entry length exceeds 2047 halfwords.

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE ENTRY TOO BIG

MEANING 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 the $CONTROL or BLOCKMAX= command, or by default.

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE ENTRY TOO SMALL

MEANING A detail data set that is not linked to any master data set must have a data entry length of two or more halfwords. This length is determined by adding the size in halfwords of each data item defined in the data entry.

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE ILLEGAL CHARACTER IN PASSWORD

MEANING The password cannot contain a period, semicolon, slash, or carriage return.

ACTION Edit and remove.

MESSAGE ILLEGAL ITEM NAME OR TERMINATOR

MEANING The data item name does not conform to naming rules. (Names must start with a letter and can have up to 16 alphanumeric characters including + – ? / # ∃ & * @). Or if in the item part, it is not followed by a comma.

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE ILLEGAL USER CLASS NUMBER

MEANING User class number defined in schema password part is not an integer between 1 and 63.

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE INCREMENT NUMBER TOO LARGE

MEANING The increment number of entries, or the entries calculated from the percent, exceeded maximum capacity minus initial capacity.

ACTION Change the increment amount to a smaller number.

MESSAGE INCREMENT PARAMETER IGNORED (INITCAP = MAXCAP)

MEANING The increment parameter is not used, since initial capacity is equal to maximum capacity and expansion is not on.

ACTION Warning only; delete the increment parameter.

MESSAGE INCREMENT PARAMETER NOT ALLOWED

MEANING The increment capacity parameter can only be defined if the initial capacity parameter is also defined.

ACTION Either specify a number for initial capacity along with the increment, or do not specify the increment parameter. Defining initial capacity and increment option means allowing dynamic expansion for the data set.

MESSAGE INITIAL CAPACITY = MAXIMUM CAPACITY, NO EXPANSION

MEANING The initial capacity is equal to the maximum capacity, no dynamic expansion for the data set.

ACTION Warning only.

MESSAGE INITIAL CAPACITY EXCEEDS MAXIMUM CAPACITY

MEANING The initial capacity defined is greater than maximum capacity.

ACTION Either change the maximum capacity or the initial capacity. Initial capacity must be less than or equal to the maximum capacity.

MESSAGE ITEM LENGTH NOT INTEGRAL WORDS

MEANING Item length is not in an even number of bytes.

ACTION Examine the schema text file to find the error. Edit the text file, and run

the Schema Processor again.

MESSAGE `ITEM TOO LONG`

MEANING The length of a single data item cannot exceed 2047 halfwords.

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE `'LANGUAGE:' EXPECTED`

MEANING The Schema Processor expected to find a `LANGUAGE` statement after the comma following the `BEGIN DATABASE` name statement.

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE `LANGUAGE NOT SUPPORTED`

MEANING Language specified is not supported on your system, or is not a valid language.

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE `MASTER DATA SET LACKS EXPECTED DETAIL(S)`

MEANING 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.

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE `MASTER DATA SET LACKS KEY ITEM`

MEANING A master data set was defined without defining one of the data items in the set as a key item.

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE `MAXIMUM CAPACITY EXCEEDS MPE/IX FILE SIZE LIMITS`

MEANING The maximum capacity defined exceeds MPE/iX file size limits (currently MPE/iX file limit is 4 gigabytes).

ACTION Change the maximum capacity to a smaller number.

MESSAGE MORE THAN ONE KEY ITEM

MEANING A master data set cannot be defined with more than one key item.

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE MORE THAN ONE PRIMARY MASTER

MEANING User has defined more than one primary path for a detail data set.

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE 'NAME:' OR 'END.' EXPECTED

MEANING Schema Processor expected, at this point, to find the beginning of another data set definition, or the end of schema.

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE NATIVE LANGUAGE SUPPORT ERROR

MEANING NLS/3000 returned an error.

ACTION Notify the system manager

MESSAGE 'PASSWORDS:' NOT FOUND

MEANING PASSWORDS statement must immediately follow the BEGIN DATABASE statement in schema. If it does not, DBSCHEMA terminates execution.

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE PASSWORD TOO LONG

MEANING A password defined in data schema cannot exceed eight characters.

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE PHYSICAL RECORD (BLOCK) TOO LARGE

MEANING Although the specified data entry is within the limit of 2048 halfwords, the path pointers cause the physical record size to exceed 2048 halfwords.

ACTION Examine the schema text file to find the error. Edit the entry to a smaller size, and run the Schema Processor again.

MESSAGE REFERENCED SET NOT MASTER

MEANING The data set referenced by the detail data set search item is another detail data set instead of a master data set.

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE SCHEMA PROCESSOR LACKS NEEDED TABLE SPACE

MEANING 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.

ACTION Ask system manager to increase maximum stack size.

MESSAGE SEARCH OR KEY ITEM NOT SIMPLE

MEANING All data items defined in data schema as master data set key or detail data set search items must be simple items.

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE SEARCH AND KEY ITEMS NOT OF SAME LENGTH

MEANING Master key item must be the same length as any related detail data set search item.

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE SEARCH AND KEY ITEMS NOT OF SAME TYPE

MEANING Master key item must be of the same type as any related detail data set search item.

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE SET HAS NO PATHS AVAILABLE

MEANING More detail data set search items have specified a relationship with a master data set than the number specified in the master set's path count.

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE SORT ITEM OF BAD TYPE

MEANING Data item defined as sort item must be of type U, K, or X.

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE SORT ITEM NOT IN DATA SET

MEANING Detail data set's entry definition does not include an item which is specified as a sort item for another item in the entry.

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE SORT ITEM SAME AS SEARCH ITEM

MEANING The same item cannot be both a search and a sort item for the same path.

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE TOO MANY DATA ITEMS

MEANING The item part of schema cannot have more than 1023 data item names.

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE TOO MANY DATA SETS

MEANING The database cannot have more than 199 data sets.

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE TOO MANY ERRORS

MEANING The specified or default number of errors has been exceeded. Processing is terminated.

ACTION Correct the errors, or increase the *ERROR* parameter value.

MESSAGE TOO MANY ITEMS SPECIFIED

MEANING The data set entry cannot have more than 255 data items.

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE TOO MANY PATHS IN DATA SET

MEANING Detail data set entries cannot have more than 16 search items.

ACTION Examine the schema text file to find the error. Edit the text file, and run the Schema Processor again.

MESSAGE UNDEFINED ITEM REFERENCED

MEANING A data item appearing in the data set definition was not previously defined in the item part of schema.

ACTION Examine the schema text file and find the incorrect statement. Edit the text file, and run the Schema Processor again.

MESSAGE UNDEFINED SET REFERENCED

MEANING Master data set referenced by detail search item was not previously defined in the set part of schema.

ACTION Examine the schema text file and find the incorrect statement. Edit the text file, and run the Schema Processor again.

Library Procedure Error Messages

The success of each call to a TurboIMAGE/XL library procedure is reflected upon return to the user by the condition code and the value of the return status in the first element of the status area.

If the procedure fails to execute properly, the condition code is set to CCL (Condition Code Less) and the return status is a negative integer. In this section, "Library Procedure File System and Memory Management" describes the negative integers resulting from file system and memory management failures, while "Library Procedure Calling Errors" describes the negative integers resulting from calling errors and communications errors respectively.

If the procedure operates properly but encounters an exceptional condition, such as end-of-file, the condition code is set to CCG (Condition Code Greater) and the return status is a positive integer. "Library Procedure Exceptional Conditions" describes the positive integers resulting from exceptional conditions. If the procedure operates properly and normally, the condition code is set to CCE (Condition Code Equal) and the return status is zero.

In addition to the return status, all TurboIMAGE/XL library procedures put information about the procedure call into the fifth through tenth elements of the status area. This information can 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/XL calls.

In a few cases this information is not returned by the TurboIMAGE/XL procedure because it uses the same locations in the status area for returning other data. Specifically, successful execution of DBFIND, DBGET, DBUPDATE, DBPUT, or DBDELETE puts other information here as described in chapter 5 of this manual. For all other returns from a library procedure, the specified elements of the status area have the following contents:

Table A-1. Status Area Changes for MPE/iX Native Applications

Element	TurboIMAGE/V and Compatibility Mode	TurboIMAGE/XL
5	PB-relative address of the caller.	0
6	Bits 6–15: Intrinsic number of called library procedure. Bits 0–3: Zero or access mode in which database is opened.	No change.
7	16-bit address of the database.	First 16 bits of the database address.
8	16-bit address of the data set name or qualifier.	Second 16 bits of database address.
9	Value of the mode parameter.	No change.
10	PB-relative address of the library procedure or the Compatibility Mode switch stub.	0

Abort Conditions

In general, four types of error conditions can cause TurboIMAGE/XL to abort the calling process:

1. A call from a Compatibility Mode user process with the hardware DB register not pointing to the process stack.
2. A structurally damaged database.
3. An internal error in an MPE file intrinsic which the calling procedure cannot correct.
4. An internal inconsistency in the database or the DBG, DBB, or DBU discovered by a library procedure.

In case 1, the procedure prints the standard MPE run-time abort message. In cases 2, 3, and 4, TurboIMAGE/XL 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 DATABASE *name;*

where *procedure name* is the name of the library procedure which caused the abort and *name* is the name of the database being accessed at the time of the abort. "Library Procedure Abort Condition Messages" describes additional lines of information which could 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, FWRITELABEL, FWRITEDIR, or FCLOSE. Aborts of this type generally occur after the procedure has possibly altered the database so that the database structure has been damaged in some way. Each of the messages in the section entitled "Library Procedure Abort Condition Messages," which refer to a TurboIMAGE/XL data file, are followed by an MPE file information display listing all of the characteristics of the MPE data set or root file where the error occurred, along with an MPE error number.

I and J Files

When TurboIMAGE/XL detects an internal inconsistency or other abnormal situation and the database is enabled for dumping, it can create special "I" and "J" files before it terminates. The "I" file consists of the user's stack and procedure call trace markers; the "J" file consists of the TurboIMAGE/XL database control blocks. TurboIMAGE/XL only creates these "I" and "J" files if a database user has run DBUTIL and specified ENABLE *database name* FOR DUMPING. So, if you want "I" and "J" files, you must specifically request them through this DBUTIL command. Note that "I" and "J" files are useful for debugging only if the database is known to be structurally sound.

Library Procedure File System and Memory Management

For return status values –1 through –6, the second element of the calling program's status area is the data set number for which file error occurred (zero indicates root file). The third element is the MPE failure code returned by the FCHECK intrinsic. Refer to MPE documentation for meaning of this code.

-1	MESSAGE	MPE file error *nn* returned by FOPEN on root file or data set *nn*
	MEANING	For DBOPEN, error can indicate that database could not be opened. Possible reasons: • Database name string not terminated with semicolon or blank. • Database does not exist or is secured against access by its group or account security. • Database is already opened exclusively or in mode incompatible with requested mode. • MPE file system error occurred. • MPE file system limit has been reached. For DBOPEN, DBINFO, DBFIND, DBGET, DBUPDATE, DBPUT, and DBDELETE, error can occur if: • The process has too many files open external to the database. • Data set does not exist or is secured against access. • Some other MPE file system error has occurred.
	ACTION	Determine which of probable causes applies and either modify application program or see system manager about file system error.
-2	MESSAGE	MPE file error *nn* returned by FCLOSE on root file or data set *nn*
	MEANING	This is an exceptional error (should never happen) and is returned only by DBOPEN or DBCLOSE. Indicates a hardware or system software failure.
	ACTION	Notify system manager of error.

-3	MESSAGE	MPE file error *nn* returned by FREADDIR on root file or data set *nn*
	MEANING	This is an exceptional error (as –2 above) and is returned by DBOPEN, DBFIND, DBGET, DBUPDATE, DBPUT, DBDELETE.
	ACTION	Notify system manager of error.
-4	MESSAGE	MPE file error *nn* returned by FREADLABEL on root file or data set *nn*
	MEANING	This is an exceptional error (as –2 above) and is returned by DBOPEN, DBINFO, DBFIND, DBGET, DBUPDATE, DBPUT, DBDELETE, and DBUNLOCK.
	ACTION	Notify system manager of error.
-5	MESSAGE	MPE file error *nn* returned by FWRITEDIR on root file or data set *nn*
	MEANING	This exceptional condition could be returned when DBPUT, DBDELETE, DBUPDATE, or DBCLOSE calls FWRITEDIR.
	ACTION	Notify system manager of error.
-6	MESSAGE	MPE file error *nn* returned by FWRITELABEL on root file or data set *nn*
	MEANING	This exceptional condition could be returned when DBPUT, DBDELETE, DBUNLOCK, or DBCLOSE calls FWRITELABEL.
	ACTION	Notify system manager of error.
-7	MESSAGE	Previous MPE file error *nn* found in desired buffer
	MEANING	This exceptional condition could be returned due to corrupted buffer for multi-user environment from DBPUT, DBUPDATE, or DBOPEN.
	ACTION	Notify your system manager of
-8	MESSAGE	MPE file error *nn* returned by FUNLOCK on root file
	MEANING	This exceptional condition could be returned when DBOPEN calls FUNLOCK on a remote database-access file.
	ACTION	Notify system manager of error.

-9	MESSAGE	Cannot create *control block name*: MPE file error %*nn*
	MEANING	This is an exceptional error returned when DBOPEN fails to call HPFOPEN to create a control block.
	ACTION	Notify system manager or HP support personnel.
-10	MESSAGE	MPE file error %*nn* returned by FFILEINFO on root file or data set *nn*
	MEANING	This is an exceptional error returned when DBOPEN fails to call FFILEINFO on the root file.
	ACTION	Notify system manager or HP support personnel.

Library Procedure Calling Errors

-11	MESSAGE	Bad *base* reference
	MEANING	For DBOPEN, the first two characters in *base* are not blank, or database name contains special characters other than period. For all other procedures, either first two characters in *base* do not contain the value assigned by DBOPEN or the parameters passed to the procedure are incorrect in type, sequence, or quantity.
	ACTION	Check application program's procedure call. Correct error in call.
-12	MESSAGE	*intrinsic name* called without covering lock in effect (See below for additional status code –12 message.)
	MEANING	For DBUPDATE, DBPUT, and DBDELETE, database has been opened in DBOPEN mode 1 but there is no lock to cover entry. DBPUT or DBDELETE to master requires data set or database be locked. In all other cases, entry, set, or database can be locked.
	ACTION	Modify program to apply proper lock or change mode.
-12	MESSAGE	Database must be in logon group and account (See above for additional status code –12 message.)
	MEANING	For DBOPEN, when the database has been opened in access mode -2, the database must be in the user's logon group and account.
	ACTION	Retry call to DBOPEN from group and account containing database.
-13	MESSAGE	Not allowed; must be creator of root file or database
	MEANING	DBOPEN failed because the caller is not the creator of the database and no maintenance word is specified.
	ACTION	Supply the correct maintenance word and retry call to DBOPEN.
-14	MESSAGE	Illegal intrinsic in current access mode
	MEANING	DBPUT and DBDELETE cannot be used with DBOPEN mode 2, 5, 6, 7, or 8. DBUPDATE cannot be used with DBOPEN

		mode 5, 6, 7, or 8.
	ACTION	Modify program or notify current user that operation cannot be performed.
-15	MESSAGE	`Setup for RDBA failed`
	MEANING	DBOPEN will issue a DSLINE command and a REMOTE HELLO on behalf of the user if a DBA file is being used for RDBA. One of these commands failed.
	ACTION	Check your DBA file to make sure the DSLINE and REMOTE HELLO commands are correct.
-21	MESSAGE	`Bad password` (See below for additional status code –21 messages.)
	MEANING	For DBOPEN, user class granted does not permit access to any data in database. This is usually due to an incorrect or null password or maintenance word.
	ACTION	Supply correct password and/or maintenance word.
-21	MESSAGE	`Bad data set reference` (See above or below for additional status code –21 messages.)
	MEANING	For DBINFO (modes 104, 201, 202, 301, and 302), DBCLOSE, DBFIND, DBGET, DBUPDATE, DBPUT, DBDELETE, when data set reference is: • Numeric but out of range of the number of data sets in database • An erroneous data set name • A reference to data set which is inaccessible to user class established when database opened For DBFIND, this error is also returned if referenced data set is a master. Erroneous data set name can arise when a terminating semicolon or blank is omitted.
	ACTION	Check application program's procedure call. Correct error in call.

-21	MESSAGE	Bad data item reference (See above for additional status code –21 messages.)
	MEANING	For DBINFO (modes 101, 102, and 204), data item reference is: • Numeric but out of range of the number of data items in database • An erroneous data item name • A reference to data item which is inaccessible to user class established when database opened. An erroneous data item name can arise when a terminating semicolon or blank is omitted.
	ACTION	Check application program's procedure call. Correct error in call.
-22	MESSAGE	Maintenance word required
	MEANING	For DBOPEN, maintenance word is required for non-creator to access the database.
	ACTION	Supply the correct maintenance word.
-23	MESSAGE	Data set not writable
	MEANING	For DBPUT and DBDELETE, database has been opened in DBOPEN mode 1, 3, or 4 and user has read but not write access to the referenced data set.
	ACTION	Modify access mode set in procedure call or notify current user operation cannot be performed.
-24	MESSAGE	Operation not allowed on automatic master data set
	MEANING	For DBPUT and DBDELETE, the referenced data set is an automatic master.
	ACTION	Modify data set name in call or in data set type in schema.
-30	MESSAGE	ILR enabled and bad access
	MEANING	The database has been ported from MPE V with ILR enabled. The only allowable access to the database is DBOPEN, mode 1–8, or DBUTIL to disable ILR.
	ACTION	Open the database with mode 1–8 or disable ILR.

-31	MESSAGE	Bad *mode*
	MEANING	This error occurs in all procedures when the *mode* parameter is invalid. For DBGET, *mode* is 7 or 8 and referenced data set is a detail, or *mode* is 5 or 6 and referenced data set is a detail without search items.
	ACTION	Correct mode in procedure call.
-32	MESSAGE	Unobtainable access mode; AOPTIONS requested: *n*, granted: *m*
	MEANING	For DBOPEN, root file cannot be opened with FOPEN using the access options (AOPTIONS) requested for the specified access mode. The second element of the calling program's status area contains the requested AOPTIONS, and the third element contains the AOPTIONS granted to DBOPEN by the MPE file system. This error usually occurs either due to concurrent database access by other users or due to MPE account or group security provisions.
	ACTION	See the *MPE/iX Intrinsics Reference Manual* for meaning of AOPTIONS values. Action depends on program's design. Normally notify user that requested access mode is not available.
-33	MESSAGE	Mode 7 diagnostics not allowed
	MEANING	For DBGET, mode 7 is not appropriate for diagnostics.
	ACTION	Use a different mode for DBGET when diagnostic item list is in effect.
-34	MESSAGE	Database must be recovered before access is allowed.
	MEANING	The system failed while the MUSTRECOVER option was enabled. The MUSTRECOVER option prevents write access to the database until it is recovered with DBRECOV.
	ACTION	Recover the database with DBRECOV.

-51	MESSAGE	Bad *list* length
	MEANING	For DBGET, DBUPDATE, and DBPUT, the list is too long. This can occur if list is not terminated with a semicolon or blank. It can also occur for otherwise legitimate lists which are too long for TurboIMAGE/XL's work area. It will never occur for numeric lists.
	ACTION	Shorten *list* array contents. If necessary, change to numeric list.
-52	MESSAGE	Bad *list* or bad *item*
	MEANING	For DBGET, DBUPDATE, or DBPUT, the *list* parameter is invalid. *list* either has a bad format or contains a data item reference which meets one of the following conditions: • It is out of range of the number of data items in the database. • It refers to an inaccessible data item. • It duplicates another reference in the list. For DBFIND, the *item* parameter contains a data item reference which meets one of the following conditions: • It is out of range of the number of data items in the database. • It is not a search item for the referenced data set.
	ACTION	Check procedure call. Correct error in call or parameter.
-53	MESSAGE	Missing search or sort item
	MEANING	For DBPUT, a search or sort item of the referenced data set is not included in *list* parameter.
	ACTION	Check procedure call. Correct error in call or parameter.
-60	MESSAGE	Illegal file equation on root file
	MEANING	When using an MPE FILE command with the database name or a database-access file name, only the file designators and DEV= parameters are allowed.
	ACTION	Reenter the MPE FILE command without illegal parameters.

-61	MESSAGE	Error while obtaining information about file equation.
	MEANING	HPSWITCHTOCM procedure failed while trying to evaluate the file equation used for DBOPEN.
	ACTION	Correct the file equation. If that does not resolve the error, notify your system manager.
-80	MESSAGE	Output Deferred not allowed with ILR enabled
	MEANING	DBCONTROL (mode 1) was used to request deferred output, but deferred output cannot be used when ILR is enabled. Deferred output is not initiated.
	ACTION	Do not use deferred output; or run DBUTIL and disable ILR.
-81	MESSAGE	Output Deferred not allowed with ROLLBACK enabled
	MEANING	DBCONTROL (mode 1) was used to request deferred output, but deferred output cannot be used when ROLLBACK is enabled. Deferred output is not initiated.
	ACTION	Do not use deferred output; or run DBUTIL and disable ROLLBACK.
-82	MESSAGE	CIUPDATE is set to DISALLOWED; cannot use critical item update
	MEANING	A process issued a call to DBCONTROL in mode 5 to use the critical item update option, but CIUPDATE is set to DISALLOWED for this database.
	ACTION	Do not use CIUPDATE. Or request that your database administrator set CIUPDATE to ALLOWED for this database.
-88	MESSAGE	Database bad: Third party indexing was in process (index again).
	MEANING	Third party indexing is in progress or was in process but failed.
	ACTION	Notify database administrator. Re-index if failed.

-89	MESSAGE	Database bad: Restructuring was in process (restore database).
	MEANING	DBChangePlus is restructuring the database or was restructuring the database but failed in the middle.
	ACTION	Notify database administrator. Restore the database and redo restructuring if failed.
-90	MESSAGE	Root file bad: unrecognized state: *%octal integer*
	MEANING	For DBOPEN, this error is returned if the root file is in an unrecognized state. The octal integer represents an ASCII error code.
	ACTION	Restore old copy of the database.
-91	MESSAGE	Bad root modification level
	MEANING	For DBOPEN, the software version of the DBOPEN procedure is incompatible with version of Schema Processor which created root file.
	ACTION	Check with system manager that you have correct TurboIMAGE/XL software. If necessary ask HP support personnel about conversion.
-92	MESSAGE	Database not created
	MEANING	For DBOPEN, the referenced database has not yet been created and initialized by the DBUTIL CREATE command.
	ACTION	Run DBUTIL to create database. Try application program again.
-93	MESSAGE	DATABASE ALREADY EXISTS.
	MEANING	Database already exists when attempted to create a new one.
	ACTION	Change the database name in the schema. If you want to create in the same name, purge the existing one and then recreate.

-94	MESSAGE	Database bad - Output deferred; may not be accessed in mode *nn*
	MEANING	For DBOPEN, referenced database was damaged while being modified in deferred output.
	ACTION	Either DBLOAD from backup tape; or DBUNLOAD to ERASE data and then DBLOAD.
-95	MESSAGE	Database bad - Creation was in process (create again)
	MEANING	For DBOPEN, database was damaged by a file system failure, system failure, or TurboIMAGE/XL abort while DBUTIL CREATE command was creating the database.
	ACTION	Because database was not created, run DBUTIL CREATE command to create the database.
-96	MESSAGE	Database bad - Erase was in process (erase again)
	MEANING	For DBOPEN, database was damaged by a file system failure, system failure, or TurboIMAGE/XL abort while DBUTIL ERASE command was erasing the database.
	ACTION	Because data was not erased, run DBUTIL ERASE command to erase the data from the database.
-97	MESSAGE	Database bad - ILR enable in process (enable again)
	MEANING	DBOPEN attempted to open a database but prior disable of ILR was not complete.
	ACTION	Run DBUTIL with ENABLE command to enable ILR.
-98	MESSAGE	Database bad - ILR disable in process (disable again)
	MEANING	DBOPEN attempted to open a database, but prior disable of ILR was not complete.
	ACTION	Run DBUTIL with DISABLE command to disable ILR.

-99	MESSAGE	UNSUPPORTED FEATURE.
	MEANING	A feature unavailable in TurboIMAGE is attempted.
	ACTION	Change the application to not use the feature that has not been implemented.

Return statuses –100 through –107 are communication errors. For –100 through –102, the third element of the calling program's status area is the MPE failure code returned by DSCHECK intrinsics.

-100	MESSAGE	DSOPEN failure
	MEANING	While executing a DBOPEN, TurboIMAGE/XL has encountered a hardware failure trying to obtain a communications line.
	ACTION	Try opening the database again. If error persists, contact your HP customer engineer.
-101	MESSAGE	DSCLOSE failure
	MEANING	This is an exceptional error returned by DBOPEN or DBCLOSE. It indicates a hardware or system software failure.
	ACTION	Notify system manager of problem.
-102	MESSAGE	DSWRITE failure
	MEANING	A line failure has occurred while attempting an operation on a remote database. Can be returned by DBOPEN, DBFIND, DBGET, DBPUT, DBUPDATE, DBDELETE, DBLOCK, DBUNLOCK, DBINFO, or DBCLOSE.
	ACTION	Try calling the procedure again. If error persists, notify system manager.
-103	MESSAGE	Remote 3000 stack too small
	MEANING	Command Interpreter on remote HP 3000 cannot obtain stack space necessary to execute a DBOPEN or DBLOCK.
	ACTION	Ask system manager of remote system to increase available stack size.

-104	MESSAGE	Remote 3000 does not support TurboIMAGE/XL
	MEANING	This is an exceptional error and is returned by DBOPEN on the remote system.
	ACTION	Notify system manager of problem.
-105	MESSAGE	Remote 3000 cannot create TurboIMAGE control block.
	MEANING	This is an exceptional error and is returned by DBOPEN on the remote system.
	ACTION	Notify system manager of problem.
-106	MESSAGE	Remote 3000 data inconsistent
	MEANING	This is an exceptional error returned by same intrinsics as –102 (see message listed earlier in this appendix). It indicates a hardware or system software failure.
	ACTION	Notify system manager of problem.
-107	MESSAGE	NS 3000 or DS 3000 system error.
	MEANING	This is an exceptional error returned by same intrinsics as –102 (see above). It indicates a hardware or system software failure.
	ACTION	Notify system manager of problem.
-108	MESSAGE	HPUNLOADCMPROCEDURE call failed.
	MEANING	This is an internal error encountered during switching to Compatibility Mode.
	ACTION	Notify HP support personnel.
-109	MESSAGE	ERROR RETURNED BY LOGINFO INTRINSIC.
	MEANING	LOGINFO intrinsic does not support the feature used by DBOPEN. It is available in MPE/iX 5.5 and later releases.
	ACTION	The version of TurboIMAGE/XL can only be used on MPE/iX 5.5 or later. Notify your system administrator.

-110	MESSAGE	MPE OPENLOG intrinsic failure
	MEANING	OPENLOG returned error number *nn* to DBOPEN. This error can occur following a call to DBOPEN when a database is enabled for logging. Refer to *MPE/iX Intrinsics Manual* for listing of second values of status array and error messages.
	ACTION	Notify system manager or HP support personnel.
-111	MESSAGE	MPE WRITELOG intrinsic failure
	MEANING	When a database is enabled for logging, this error can be returned by DBOPEN, DBCLOSE, DBPUT, DBUPDATE, DBDELETE, DBMEMO, DBBEGIN, DBEND, DBXBEGIN, DBXEND, DBXUNDO.
	ACTION	Notify database administrator.
-112	MESSAGE	MPE CLOSELOG intrinsic failure
	MEANING	When a database is enabled for logging, this error can be returned by DBCLOSE. Or CLOSELOG returned error number *nn* to DBCLOSE.
	ACTION	Consult MPE error message documentation. Or notify database administrator.
-113	MESSAGE	FLUSHLOG returned error number *nn* to DBEND
	MEANING	User called DBEND in mode 2 to write the logging buffer to disk, but the MPE logging facility returned an error.
	ACTION	Consult MPE error message documentation.
-114	MESSAGE	ROLLBACK without logging
	MEANING	DBOPEN has detected that ROLLBACK is enabled but logging is not.
	ACTION	Notify your database administrator or HP support personnel.
-120	MESSAGE	OUT OF STACK SPACE FOR A LOCK.
	MEANING	Available space is not sufficient to process a DBLOCK.
	ACTION	Notify HP support personnel.

-121	MESSAGE	Descriptor count error
	MEANING	DBLOCK detected an error in the descriptor count (first element of qualifier array) in locking mode 5 or 6.
	ACTION	Count must be a positive integer.
-123	MESSAGE	Illegal *relop* in a descriptor
	MEANING	DBLOCK encountered a *relop* field containing characters other than >=, <=, = or =.
	ACTION	Check contents of *qualifier* array.
-124	MESSAGE	Descriptor too short. Must be greater than or equal to 9
	MEANING	DBLOCK encountered a lock descriptor less than 9 halfwords long.
	ACTION	Check contents of *qualifier* array.
-125	MESSAGE	Bad set name/number
	MEANING	DBLOCK *qualifier* array contains an invalid data set name or number. (Refer to error -21 for rules.)
	ACTION	Check contents of qualifier array. Be sure names are delimited by semicolon or space if less than 16 bytes long.
-126	MESSAGE	Bad item name/number
	MEANING	DBLOCK *qualifier* array contains an invalid data item name or number. (Refer to error -21 for rules.)
	ACTION	Check contents of qualifier array. Be sure names are delimited by semicolon or space if less than 16 bytes long.
-127	MESSAGE	Attempt to lock using a compound item
	MEANING	DBLOCK does not allow compound items in lock descriptors.
	ACTION	Modify locking strategy to lock on a non-compound item.

-128	MESSAGE	Value field too short in a descriptor
	MEANING	A *value* field in a DBLOCK lock descriptor must be at least as long as the data item for which it is specified.
	ACTION	Check the length of the value field.
-129	MESSAGE	P-type item longer than P28 specified
	MEANING	DBLOCK does not allow P-type data items longer than 28 in lock descriptors (27 digits plus sign).
	ACTION	Modify locking strategy to lock on a different item.
-130	MESSAGE	Illegal digit in a P-type value
	MEANING	DBLOCK has encountered a P-type value in a lock descriptor with an invalid packed decimal digit.
	ACTION	Check *qualifier* array contents to determine why data is invalid.
-131	MESSAGE	Lowercase character in type-U value
	MEANING	DBLOCK has encountered a lowercase character in a type-U value specified in a lock descriptor.
	ACTION	Check *qualifier* array contents to determine why data is invalid.
-132	MESSAGE	Illegal digit in type Z value
	MEANING	Lock descriptor value specified to DBLOCK contains an invalid zoned decimal digit.
	ACTION	Check *qualifier* array contents to determine why data is invalid.
-133	MESSAGE	Illegal sign in type Z value
	MEANING	Lock descriptor value specified to DBLOCK contains an invalid zoned decimal sign.
	ACTION	Check *qualifier* array contents to determine why data is invalid.

-134	MESSAGE	Two descriptors conflict
	MEANING	DBLOCK has detected two lock descriptors in the same call that lock the same or part of the same database entity. (For example, lock on set and database in same request.)
	ACTION	Check *qualifier* array contents for conflicting lock descriptors.
-135	MESSAGE	Second lock without CAP=MR
	MEANING	A second call to DBLOCK has been made without an intervening DBUNLOCK call and program does not have MR capability.
	ACTION	Read discussion of multiple calls to DBLOCK in chapter 4 of this manual if you plan to use CAP=MR.
-136	MESSAGE	Descriptor list exceeds 4094 bytes
	MEANING	DBLOCK allows a maximum length of 4094 bytes for lock descriptor lists (*qualifier* array).
	ACTION	Change *qualifier* array contents so lock descriptor list is shorter.
-137	MESSAGE	USER ABOUT TO WAIT FOR SELF.
	MEANING	There is a potential for an imminent deadlock situation.
	ACTION	Examine the locking scheme in your application; may need to call DBUNLOCK and then DBLOCK.
-139	MESSAGE	Invalid number of base IDs.
	MEANING	The number of base IDs involved in the multiple database transaction is greater than 15.
	ACTION	Modify your application program so that not more than 15 databases are involved in a multiple database transaction.
-140	MESSAGE	Bad base ID list.
	MEANING	The base IDs are not correctly formatted for a multiple database transaction.
	ACTION	Modify your application program so that the base ID list is correctly formatted. Refer to the discussion of DBBEGIN in chapter 5 for more information.

-141	MESSAGE	All MDBX databases must be on same system.
	MEANING	An attempt was made to begin a multiple database transaction (MDBX) on remote databases when all databases were not located on the same system (node).
	ACTION	Modify your application program to only perform multiple database transactions on databases which are located on the same system.
-142	MESSAGE	All MDBX databases must log to the same log file.
	MEANING	All databases involved in a multiple database transaction (MDBX) must have synchronized logging facilities.
	ACTION	Ask your database administrator or application programmer to synchronize the logging facilities.
-143	MESSAGE	Logging must be enabled or disabled for all MDBX databases.
	MEANING	All databases involved in a multiple database transaction (MDBX) must have synchronized logging facilities.
	ACTION	Ask your database administrator or application programmer to synchronize the logging facilities.
-144	MESSAGE	MUSTRECOVER must be enabled or disabled for all MDBX databases.' ' \MEANING\All databases involved in a multiple database transaction (MDBX) must have synchronized logging facilities.
	ACTION	Ask your database administrator or application programmer to synchronize the logging facilities.
-145	MESSAGE	Roll-back recovery must be enabled or disabled for all MDBX databases.
	MEANING	All databases involved in a multiple database transaction (MDBX) must have synchronized logging facilities.
	ACTION	Ask your database administrator or application programmer to synchronize the logging facilities.

-146	MESSAGE	Invalid transaction ID.
	MEANING	The transaction ID used in a call to DBBEGIN and in the call to its corresponding DBEND do not match.
	ACTION	Contact your database administrator or application programmer.
-147	MESSAGE	Mode doesn't match DBBEGIN mode.
	MEANING	The DBBEGIN mode and its corresponding DBEND mode must match.
	ACTION	Contact your database administrator or application programmer.
-148	MESSAGE	Base ID list doesn't match DBBEGIN base ID list.
	MEANING	The base ID list passed to DBBEGIN for a multiple database transaction does not match the DBEND base ID list for the same transaction.
	ACTION	Contact your database administrator or application programmer.
-151	MESSAGE	Text length greater than 512 bytes
	MEANING	Text provided to DBBEGIN, DBEND, DBMEMO, DBXBEGIN, DBXEND, DBXUNDO is too long.
	ACTION	Modify program.
-152	MESSAGE	DBCLOSE called while a transaction is in process; DBBEGIN called while a transaction is in process; DBXBEGIN called while a transaction is in progress
	MEANING	A transaction is in process.
	ACTION	Call DBEND before DBCLOSE. Call DBEND before the next DBBEGIN. Call DBXEND before the next DBXBEGIN.
-153	MESSAGE	DBEND called while no transaction in progress
	MEANING	No DBBEGIN precedes the call to DBEND.
	ACTION	Call DBBEGIN before DBEND.

-160	MESSAGE	ILR log file name conflict
	MEANING	A user file has the same name as the ILR log file name.
	ACTION	Replace the user file with a valid MPE ILR log file.
-161	MESSAGE	Cannot check for an ILR log file conflict: file system error *nn*
	MEANING	DBUTIL DISABLE command or call to DBOPEN failed to open ILR log file.
	ACTION	Check file system error number.
-166	MESSAGE	Cannot purge ILR log file: file system error *nn*
	MEANING	DBUTIL DISABLE, PURGE, or ERASE command could not purge an ILR log file because of a file system error.
	ACTION	Check file system error number.
-167	MESSAGE	Cannot begin MPE XL XM transaction: XM error *nn*
	MEANING	The logical beginning of an MPE transaction failed. *nn* represents the error number returned.
	ACTION	Notify HP support personnel.
-168	MESSAGE	Cannot attach *n* to MPE XL X<: file system error *nn*
	MEANING	Data set *n* could not be attached to the MPE transaction recovery mechanism. MPE intrinsic FILEINFO or FLABELINFO failed.
	ACTION	Notify HP support personnel.
-169	MESSAGE	Invalid mode for XM attach options
	MEANING	An internal error has occurred.
	ACTION	Notify HP support personnel.

-170	MESSAGE	Cannot open ILR log file: file system error *nn*
	MEANING	DBUTIL DISABLE command or call to DBOPEN was not able to open ILR log file due to file system error.
	ACTION	Check file system error number.
-172	MESSAGE	Cannot read ILR log file: file system error *nn*
	MEANING	DBUTIL DISABLE command or call to DBOPEN was not able to read ILR log file due to file system error.
	ACTION	Check file system error number.
-173	MESSAGE	Unable to obtain write access
	MEANING	DBOPEN has determined that it is necessary to perform ILR. The current user does not have write access to the database.
	ACTION	Have the database opened by a user with write access.
-174	MESSAGE	Bad mode for ILR
	MEANING	DBOPEN has determined that it is necessary to perform ILR. The database must be opened for writing.
	ACTION	Open the database in mode 1–4.
-175	MESSAGE	Cannot attach *n* to MPE XL XM: XM error *nn*
	MEANING	The data set *n* could not be attached to the transaction recovery mechanism. *nn* is the MPE error number returned.
	ACTION	Notify HP support personnel.
-176	MESSAGE	Cannot detach *n* from MPE XL XM: XM error *nn*
	MEANING	The data set *n* could not be detached from the transaction recovery mechanism. *nn* is the MPE error number returned.
	ACTION	Notify HP support personnel.

-177	MESSAGE	MPE log file is not in the same volume set as the database.
	MEANING	MPE transaction recovery requires that the user log file must reside in the same volume set as the database.
	ACTION	Build the MPE user log file in same volume set as database. **NOTE**: Because all files in a group are in same volume set, the MPE LISTGROUP command indicates the volume set where a database resides.
-178	MESSAGE	Cannot detach *n* from MPE XL XM: File system error *nn*
	MEANING	The data set *n* could not be detached from the MPE transaction recovery mechanism. The MPE intrinsic FLABELINFO failed. *nn* is the file system error number returned.
	ACTION	Notify HP support personnel.
-179	MESSAGE	Cannot begin MPE XL transaction for attach: XM error *nn*
	MEANING	Before attaching the entire database to the MPE transaction recovery mechanism, a logical beginning of a transaction is specified. The beginning of the transaction failed. *nn* is the error number returned.
	ACTION	Notify HP support personnel.
-180	MESSAGE	ILR log invalid - internal file name does not match root file
	MEANING	DBUTIL DISABLE command or call to DBOPEN attempted to open ILR log file, but the file names are inconsistent. This is unlikely to occur unless the disk is corrupted, or the TurboIMAGE/XL root file or ILR log file has been altered by a privileged mode user.
	ACTION	Use DBRESTOR to restore database.
-181	MESSAGE	ILR log file invalid - internal group name does not match root file
	MEANING	DBUTIL DISABLE command or call to DBOPEN attempted to open ILR log file, but group names do not match. (See error –180 for possible cause.)
	ACTION	With ILR on, use the MPE RESTORE command with the

		GROUP= option to restore the database. To avoid this error, disable ILR when moving a database from one group and account to another group and account.
-182	MESSAGE	ILR log invalid - internal account name does not match root file
	MEANING	DBUTIL DISABLE command or call to DBOPEN attempted to open ILR log file, but account names do not match. (See error –180 for possible cause.)
	ACTION	With ILR on, use the MPE RESTORE command with the ACC= option to restore the database. To avoid this error, disable ILR when moving a database from one group and account to another group and account.
-183	MESSAGE	ILR log invalid - internal creation date does not match root file
	MEANING	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/XL root file. Could be caused by storing the database followed by a partial restore with the MPE RESTORE command that excluded either the root file or the ILR log file. If there was an intervening ENABLE or DISABLE command, the creation dates will not match and only a partial DBSTORE/DBRESTOR occurred.
	ACTION	To avoid this problem always store database with DBSTORE and restore with DBRESTOR. Using these utilities assures that root file and ILR log file are consistent.
-184	MESSAGE	ILR log invalid - internal last access data access data does not match root file
	MEANING	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 root file. Possibly caused by a store and then a partial restore (using the MPE RESTORE command) of the database.
	ACTION	Always store database with DBSTORE and restore with DBRESTOR to assure consistent dates in root file and log file.

-187	MESSAGE	ILR is already enabled for this database
	MEANING	DBUTIL ENABLE command attempted to enable a database for ILR when ILR was already enabled.
	ACTION	Warning only.
-188	MESSAGE	ILR is already disabled for this database
	MEANING	DBUTIL DISABLE command attempted to disable a database for ILR when ILR was already disabled.
	ACTION	Warning only.
-189	MESSAGE	Cannot begin MPE XL transaction for detach: XM error *nn*
	MEANING	Before detaching the entire database from the MPE transaction recovery mechanism, a logical ending of a transaction is specified. The ending of the transaction failed. *nn* is the error number returned.
	ACTION	Notify HP support personnel.
-190	MESSAGE	Bad DBS
	MEANING	The Database System Control Block has been corrupted.
	ACTION	Notify HP support personnel.
-191	MESSAGE	The DBS is full
	MEANING	DBOPEN has detected that there is no more room in the DBS to hold any more DBG entries.
	ACTION	Notify HP support personnel.
-192	MESSAGE	Invalid DBU
	MEANING	The database user local control block has been corrupted.
	ACTION	Notify HP support personnel.
-193	MESSAGE	DBU control block is full
	MEANING	The database user local control block is full.
	ACTION	Notify HP support personnel.

-194	MESSAGE	Invalid DBB
	MEANING	The database buffer control block has been corrupted.
	ACTION	Notify HP support personnel.
-195	MESSAGE	Invalid DBG
	MEANING	The database global control block has been corrupted.
	ACTION	Notify HP support personnel.
-196	MESSAGE	DBB control block is full
	MEANING	The database buffer control block is full.
	ACTION	Notify HP support personnel.
-197	MESSAGE	DBG control block is full
	MEANING	The database global control block is full.
	ACTION	Notify HP support personnel.
-198	MESSAGE	Total DBOPEN count/user exceeds limit
	MEANING	The DBU index table, the DBUX, is full. This table is used to map the *baseid* to the user's DBU. The DBUX holds a maximum of 127 entries.
	ACTION	Notify HP support personnel.
-199	MESSAGE	Cannot end MPE XLSM transaction: XM error *nn*
	MEANING	The logical ending of an MPE transaction failed. *nn* is the error number returned. Any writes done on behalf of the intrinsic will be rolled out. **NOTE**: This error is only possible from DBOPEN, DBPUT, DBDELETE, DBUNLOCK, or DBUPDATE.
	ACTION	Notify system manager.
-200	MESSAGE	Database language not system supported
	MEANING	DBOPEN attempted to open the database and found that the language of the database is not currently configured. The collating sequence of the language is unavailable; DBOPEN cannot open the database.
	ACTION	Notify system manager.

-201	MESSAGE	Native Language Support not installed
	MEANING	NLS/3000 internal structures have not been built at system start-up. The collating sequence table of the language of the database is unavailable; DBOPEN cannot open the database.
	ACTION	Notify system manager.
-202	MESSAGE	MPE Native Language Support error *nn* returned by NLINFO
	MEANING	The error number given was returned by NLS/3000 on a NLINFO call in DBOPEN.
	ACTION	Notify system manager.
-204	MESSAGE	USER STACK IS TOO SMALL FOR RECOVERY IN DBOPEN.
	MEANING	There is not sufficient stack space to perform dynamic transaction recovery.
	ACTION	Increase the stack size for the user application.
-205	MESSAGE	WRONG VERSION OF DS SUBSYSTEM.
	MEANING	Remote database access is done, however, the DS subsystem does not support it.
	ACTION	Ensure that the DS subsystem is compatible with TurboIMAGE/XL. Notify your system administrator.
-206	MESSAGE	Database exceeds limits
	MEANING	DBOPEN has detected that the remote TurboIMAGE/XL database exceeds the local IMAGE limitations. Remote database access cannot be completed. A TurboIMAGE/XL database can only be accessed from a local node (which has IMAGE) if and only if the remote database has not exceeded the IMAGE/3000 data set and data item limitations.
	ACTION	Access the remote TurboIMAGE/XL database from another local node which also supports TurboIMAGE/XL.

-208	MESSAGE	MPE error *nn* returned by FLABELINFO for MPE XL XM
	MEANING	MPE file system error *nn* was returned by the intrinsic FLABELINFO.
	ACTION	Notify HP support personnel.
-209	MESSAGE	Invalid mode for MPE XL XM DETACH
	MEANING	An internal error has occurred.
	ACTION	Notify HP support personnel.
-210	MESSAGE	MPE error *nn* on log file name
	MEANING	MPE file system error *nn* was returned while trying to obtain the user log file name.
	ACTION	Notify HP support personnel.
-211	MESSAGE	INVALID OR NO USER LABEL.
	MEANING	The dataset file has no user label or a corrupted user label.
	ACTION	Notify your system administrator and HP support personnel.
-212	MESSAGE	Database corruption was detected.
	MEANING	Internally, the target record either converted into a wrong slot within a block or the record is beyond the dataset capacity.
	ACTION	Notify HP support personnel.
-213	MESSAGE	BXEND encountered XM error *nn* when ending a dynamic transaction
	MEANING	This is an internal error.
	ACTION	Notify your system manager and HP support personnel.
-214	MESSAGE	CANNOT CALL DBXUNDO WHEN A TRANSACTION IS STARTED BY DBBEGIN.
	MEANING	DBXUNDO can be called only when DBXBEGIN is called and not DBBEGIN. DBXBEGIN, DBXUNDO, and DBXEND are compatible intrinsics. DBBEGIN can only be matched with DBEND.

	ACTION	Correct the application.
-215	MESSAGE	M error *nn* encountered when rolling out a dynamic transaction.
	MEANING	This is an internal error.
	ACTION	Notify your system manager and HP support personnel.
-216	MESSAGE	Cannot end a dynamic transaction with DBEND
	MEANING	A transaction started by DBXBEGIN cannot use DBEND.
	ACTION	Replace the DBEND call with a call to DBXEND.
-217	MESSAGE	BOPEN mode *n* incompatible with Dynamic Rollback
	MEANING	Strong locking must be enforced for dynamic transactions.
	ACTION	Use DBOPEN mode 1, 3, or 4.
-218	MESSAGE	Output deferred is incompatible with Dynamic Rollback
	MEANING	A database cannot be enabled for AUTODEFER when dynamic roll-back intrinsics are used.
	ACTION	Either disable the database for deferred output or do not use dynamic roll-back intrinsics.
-219	MESSAGE	Remote database access is incompatible with Dynamic Rollback
	MEANING	Remote database access cannot be used in conjunction with dynamic roll-back.
	ACTION	Use either remote database access or dynamic roll-back intrinsics, but not both.
-220	MESSAGE	Database and user log not attached to the same XM log set
	MEANING	When a database is enabled for DBRECOV roll-back recovery, the database and user log must be attached to the same XM log set.
	ACTION	Move the database and user log to the same XM log set.

-221	MESSAGE	Cannot begin a transaction when a dynamic transaction is active
	MEANING	Each DBOPEN can have only one dynamic transaction active at one time.
	ACTION	Check the application to make sure that only one dynamic transaction is active at one time.
-222	MESSAGE	Only DBXUNDO allowed when dynamic transaction encounters an error
	MEANING	A write intrinsic or DBUNLOCK error occurred while in an active dynamic transaction. Only DBXUNDO can be called.
	ACTION	Change the application to call DBXUNDO when this type of error occurs.
-223	MESSAGE	Cannot DBXEND or DBXUNDO a transaction which was not active
	MEANING	DBXEND or DBXUNDO called and no dynamic transaction is active.
	ACTION	Do not call DBXEND or DBXUNDO unless the dynamic transaction has been started by a *DBXBEGIN*.
-224	MESSAGE	DBCONTROL mode 1 not allowed inside a dynamic transaction
	MEANING	DBCONTROL mode 1 enables a database for deferred output. Deferred output and dynamic roll-back are incompatible.
	ACTION	Do not use DBCONTROL mode 1 when inside a dynamic transaction.
-225	MESSAGE	Record table full for Dynamic Rollback
	MEANING	The transaction defined in the user application exceeds the maximum transaction size for dynamic transactions.
	ACTION	Shorten your dynamic transaction, or notify your system manager and HP support personnel.
-226	MESSAGE	Error occurred in when the 00 file was created
	MEANING	The transaction defined in the user application exceeds the maximum transaction size for dynamic transactions.

	ACTION	Notify your database administrator and HP support personnel.
-227	MESSAGE	Error occurred in 00 file recovery
	MEANING	An internal error has occurred.
	ACTION	Notify your database administrator and HP support personnel.
-228	MESSAGE	BXBEGIN encountered XM error *nn* when starting a dynamic transaction
	MEANING	An internal error has occurred.
	ACTION	Notify your database administrator and HP support personnel.
-229	MESSAGE	CANNOT DELETE MANUAL MASTER WITH EMPTY CHAINS.
	MEANING	Error occurred while deleting a manual master entry despite the chain being empty.
	ACTION	Notify HP support personnel.
-230	MESSAGE	A DBUNLOCK inside a dynamic transaction is not allowed
	MEANING	DBUNLOCK can only be performed when a dynamic transaction has been completed with a DBXEND or DBXUNDO.
	ACTION	Change the application so that it does not call DBUNLOCK after DBPUT, DBDELETE, or DBUPDATE is used inside a dynamic transaction.
-231	MESSAGE	During Dynamic Rollback recovery, internal procedure failed; error *nn*
	MEANING	An internal error has occurred.
	ACTION	Notify your database administrator and HP support personnel.
-232	MESSAGE	Illegal DBCLOSE mode 2 used during an active dynamic transaction.
	MEANING	Only DBCLOSE mode 3 is allowed during an active dynamic

		transaction.
	ACTION	Move the DBCLOSE call outside of the dynamic transaction, or, if appropriate, change the DBCLOSE mode to 3.
-233	MESSAGE	Key data found in database does not match that in the memo record
	MEANING	An internal error has occurred.
	ACTION	Notify your database administrator and Hewlett-Packard support personnel.
-234	MESSAGE	Cannot purge the 00 file
	MEANING	The 00 file cannot be purged due to MPE file system error.
	ACTION	Find out the file system error number in element 2 of the status array, and notify your database administrator and HP support personnel.
-235	MESSAGE	Dynamic transaction aborted due to DBCLOSE mode 1, database closed.
	MEANING	Only DBCLOSE mode 3 is allowed during an active dynamic transaction.
	ACTION	Move the DBCLOSE call outside of the dynamic transaction.
-236	MESSAGE	Internal error *nn* occurred when opening the AUX file.
	MEANING	An internal error has occurred.
	ACTION	Notify your database administrator and Hewlett-Packard support personnel.
-237	MESSAGE	Cannot DBXEND or DBXUNDO a DBBEGIN transaction
	MEANING	A transaction started by a DBBEGIN cannot use DBXEND or DBXUNDO.
	ACTION	Replace the DBXEND call with a call to DBEND or replace the DBBEGIN call with a call to DBXBEGIN.

-238	MESSAGE	MDBX, MODES OF DBXBEGIN/DBXEND DON'T MATCH.
	MEANING	DBXUNDO or DBXEND call, including the mode, does not match with the dynamic multi-database transaction started by DBXBEGIN.
	ACTION	Correct the application.
-240	MESSAGE	ERROR IN DYNAMIC ROLLBACK.
	MEANING	The dynamic transaction could not be rolled back successfully.
	ACTION	Notify HP support personnel.
-241	MESSAGE	BAD TAG FOR TURBOLKT TABLE.
	MEANING	The tag in the header record of the lock table, TURBOLKT.PUB.SYS, appears to be corrupted.
	ACTION	Notify HP support personnel.
-242	MESSAGE	Dynamic Rollback MDBX error.
	MEANING	Internal errors returned from various low level procedures which handle dynamic multiple database transactions.
	ACTION	Notify HP support personnel.
-243	MESSAGE	INVALID DYNAMIC ROLLBACK TRANSACTION ID.
	MEANING	The transaction id in the base parameter of the DBXUNDO call does not match with its corresponding DBXBEGIN.
	ACTION	Correct the application.
-244	MESSAGE	BASE COUNT OVER 15 FOR DMDBX.
	MEANING	The limit of the number of the databases to be included in a dynamic multi-database transaction is 15.
	ACTION	Correct the application.
-245	MESSAGE	OUT OF SPACE FOR TURBOGTX FILE.
	MEANING	The table, TURBOGTX.PUB.SYS, is full and additional information cannot be added to this table.
	ACTION	Notify HP support personnel.

-246	MESSAGE	ERROR IN TURBOGTX OPERATION RELATED TO ATC TRANSACTION.
	MEANING	Internal error encountered while updating the table, TURBOGTX.PUB.SYS, as there is still an active SQL transaction in the table.
	ACTION	Notify HP support personnel.
-250	MESSAGE	CBINIT FAILED ON *nn*
	MEANING	The initialization of a semaphore failed.
	ACTION	Notify HP support personnel.
-251	MESSAGE	DBS WAS OBTAINED BUT NOT RELEASED.
	MEANING	There is an active lock granted on the file, TURBODBS.PUB.SYS, and not released.
	ACTION	Notify HP support personnel.
-253	MESSAGE	Database enabled for indexing, but third-party indexing is not configured
	MEANING	This message is returned if you are using third-party indexing (TPI). The database has been enabled for indexing via DBUTIL, but the third-party product or database is not configured correctly.
	ACTION	Consult your third-party vendor documentation.
-254	MESSAGE	Rollback of third-party index failed; indexing disabled for data set
	MEANING	This message is returned if you are using third-party indexing (TPI). The third-party product did not perform the necessary roll-back function to maintain data set consistency.
	ACTION	Consult your third-party vendor documentation.
-255	MESSAGE	Third-party indexing disable failed; indexing disabled for database
	MEANING	This message is returned if you are using third-party indexing (TPI). The third-party product did not perform the necessary roll-back and single-index-disable functions to maintain database consistency (see error –254).
	ACTION	Consult your third-party vendor documentation.

-256	MESSAGE	Third-party index for path *nn* is full
	MEANING	This message is returned if you are using third-party indexing (TPI). The index file for the indicated path is full; *nn* represents the path number.
	ACTION	Consult your third-party vendor documentation.
-257	MESSAGE	Third-party index for path *nn* is damaged
	MEANING	This message is returned if you are using third-party indexing (TPI). The index file for the indicated path is damaged; *nn* represents the path number.
	ACTION	Consult your third-party vendor documentation.
-258	MESSAGE	Invalid argument for index
	MEANING	This message is returned if you are using third-party indexing (TPI). The argument parameter contained an inappropriate value for the specified index.
	ACTION	Consult your third-party vendor documentation.
-259	MESSAGE	Invalid mode for index
	MEANING	This message is returned if you are using third-party indexing (TPI). The mode is not applicable to this type of index.
	ACTION	Consult your third-party vendor documentation.
-260	MESSAGE	No previous list of qualified data entries
	MEANING	This message is returned if you are using third-party indexing (TPI). The DBFIND mode specified can only be used if a qualified set of entries exists.
	ACTION	Consult your third-party vendor documentation.
-261	MESSAGE	DYNAMIC PROCEDURE LOAD ERROR FOR INTRINSIC ROLLBACK.
	MEANING	There is a mismatch of the TurboIMAGE/XL version on the system and MPE/iX. The failed DBPUT, DBDELETE, or DBUPDATE could not be rolled back.
	ACTION	Notify your system administrator.

-262	MESSAGE	OLDER/INCOMPATIBLE VERSION OF IMAGE/SQL.
	MEANING	The version of IMAGE/SQL on the system is incompatible with TurboIMAGE/XL.
	ACTION	Notify your system administrator.
-263	MESSAGE	INVALID PCODE RETURNED BY TPI.
	MEANING	The third-party indexing product returned an unsupported or invalid pcode to IMAGE/SQL.
	ACTION	Notify your system administrator or third-party vendor.
-264	MESSAGE	WRITE ERROR FOR TPI FILES.
	MEANING	Third-party indexing product encountered error from storage management procedures. The transaction may be too big, causing the transaction to stall.
	ACTION	Notify HP support personnel.
-265	MESSAGE	ERROR IN THIRD-PARTY SHADOWING PACKAGE.
	MEANING	An internal error encountered while writing to third-party shadowing files.
	ACTION	Notify your HP personnel.
-266	MESSAGE	ERROR WHILE DISABLING THIRD-PARTY SHADOWING.
	MEANING	An internal error encountered while disabling third-party shadowing feature.
	ACTION	Notify your HP personnel.
-267	MESSAGE	DAMAGED FILE ERROR RETURNED BY THIRD-PARTY SHADOWING.
	MEANING	The files used for third-party shadowing appear to be corrupted.
	ACTION	Notify your HP personnel.

-268	MESSAGE	INVALID PCODE RETURNED BY TPS
	MEANING	The third-party shadowing product returned an unsupported or invalid pcode to IMAGE/SQL.
	ACTION	Notify your HP personnel.
-269	MESSAGE	WRITE ERROR FOR TPS FILES.
	MEANING	The transaction management encountered error while writing to third-party shadowing files.
	ACTION	Notify your HP personnel.
-3*nn*	MESSAGE	Internal error returned
	MEANING	Internal error *nn* occurred while in an intrinsic.
	ACTION	Notify HP support personnel.
-305	MESSAGE	INVALID DATA SET NUMBER.
	MEANING	Invalid set number for the master set. The internal error is returned when trying to find the primary address in the set using the key value.
	ACTION	Notify your HP personnel.
-306	MESSAGE	INVALID DATA SET TYPE.
	MEANING	Invalid set type, must be master. The internal error is returned when trying to find the primary address in the set using the key value.
	ACTION	Notify your HP personnel.
-307	MESSAGE	INVALID RECORD NUMBER FOUND.
	MEANING	The internal record pointer is bad.
	ACTION	Notify HP support personnel.
-308	MESSAGE	ERROR RELATED TO BEGINNING OF FILE.
	MEANING	An internal error encountered while trying to obtain the beginning of file information.
	ACTION	Notify your HP personnel.

-309	MESSAGE	BUFFER IO NOT YET COMPLETE.
	MEANING	An internal error encountered while performing internal buffer I/O of the data set.
	ACTION	Notify your HP personnel.
-310	MESSAGE	ERROR RELATED TO END OF FILE.
	MEANING	An internal error encountered while obtaining the end of file information.
	ACTION	Notify your HP personnel.
-312	MESSAGE	INTERNAL ERROR ENCOUNTERED WHILE READING DATABASE BLOCK.
	MEANING	An internal error encountered while reading the data set file.
	ACTION	Notify your HP personnel.
-314	MESSAGE	ERROR WHILE OBTAINING PATH INFORMATION FOR SET.
	MEANING	An internal error encountered while obtaining the path information for the set.
	ACTION	Notify your HP personnel.
-322	MESSAGE	INTERNAL TURBOIMAGE ERROR RETURNED *nn*
	MEANING	TurboIMAGE internal procedure which processed the item list encountered an unexpected error.
	ACTION	Notify HP support personnel.
-323	MESSAGE	UNEXPECTED EMPTY RECORD FOUND.
	MEANING	An unexpected empty record found when expecting a data record.
	ACTION	Notify your HP personnel.
-331	MESSAGE	DSET CAPACITY INFORMATION NOT CURRENT.
	MEANING	The data set capacity information appears to be incorrect in the user label of the file.
	ACTION	Notify your HP personnel.

-332	MESSAGE	ERROR IN QLOCK OPERATION.
	MEANING	An internal error encountered while reading from or writing to the QLOCK Table.
	ACTION	Notify your HP personnel.
-333	MESSAGE	ERROR IN QOPEN OPERATION.
	MEANING	An internal error encountered while reading from or writing to the QOPEN Table.
	ACTION	Notify your HP personnel.
-420	MESSAGE	FEATURE NOT IMPLEMENTED.
	MEANING	Use of a feature not supported for B-Tree indices is detected.
	ACTION	Correct your application.
-421	MESSAGE	BTE: unknown qualifier value for DBCONTROL mode 13.
	MEANING	The function code in the qualifier that passed to DBCONTROL is invalid.
	ACTION	Modify the program.
-422	MESSAGE	BTE: data set# not in valid range.
	MEANING	The data set number in the qualifier that passed to DBCONTROL is invalid.
	ACTION	Modify the program.
-423	MESSAGE	BTE: B-Tree already exists.
	MEANING	The data set you want to create a B-Tree for already has a B-Tree associated with it.
	ACTION	Check which data set you wish to create a B-Tree for and correct it.
-424	MESSAGE	BTE: Failed to create B-Tree.
	MEANING	The creation of the B-Tree failed.
	ACTION	Check the third and fourth halfword of the qualifier parameter passed to DBCONTROL for the status. This value should provide extra error information in HPERRMSG

format. Notify HP support personnel.

-425	MESSAGE	BTE: DB not opened exclusively.
	MEANING	The database is not opened exclusively while creating the B-Tree.
	ACTION	Modify the program. Creation of a B-Tree requires exclusive access to the database.
-426	MESSAGE	BTE: B-Tree doesn't exist.
	MEANING	A B-Tree does not appear to exist for a rebuildindex to work with.
	ACTION	Check the existence of the B-Tree or modify the program.
-427	MESSAGE	BTE: FCLOSE, purge failed.
	MEANING	While trying to purge the B-Tree file, FCLOSE failed.
	ACTION	Check the status of the B-Tree file. This may indicate a problem with the file security of the B-Tree file.
-428	MESSAGE	BTE: Rebuildindex failed.
	MEANING	The process of rebuilding the B-Tree failed.
	ACTION	Check the third and fourth halfword of the qualifier parameter you passed to DBCONTROL for the status. This value should provide extra error information in HPERRMSG format. Notify HP support personnel.
-429	MESSAGE	BTE: DBFIND argument version is bad.
	MEANING	The version number in the argument parameter passed to DBFIND is invalid. The version number tells DBFIND how to interpret the argument data.
	ACTION	Modify the program. Refer to DBFIND in chapter 5.
-430	MESSAGE	BTE: DBFIND (mode 4/24) argument type is bad.
	MEANING	The argument type in the argument parameter passed to DBFIND is invalid.
	ACTION	Modify the program. Refer to DBFIND in chapter 5.

-431	MESSAGE	BTE: DBFIND (mode 4/24) argument #1 length is bad.
	MEANING	The size (in bytes) for argument 1 in the argument parameter passed to DBFIND is less than 0 or greater than the key length.
	ACTION	Modify the program.
-432	MESSAGE	BTE: WILDCARD NOT ASCII
	MEANING	The wildcard is not a valid printable ASCII character (or a decimal value in the range 33-126) in the DBCONTROL mode 14 call.
	ACTION	Correct your application.
-433	MESSAGE	BTE: DBFIND (mode 4/24) argument #2 length is bad.
	MEANING	The size (in bytes) for argument 2 in the argument parameter passed to DBFIND is less than 0 or greater than the key length, or not valid for the search type.
	ACTION	Modify the program.
-434	MESSAGE	BTE: Data set is a detail and not a master.
	MEANING	The data set being accessed is a detail data set and not a master data set.
	ACTION	Correct application.
-436	MESSAGE	BTE: Failed to extract data from root file.
	MEANING	The procedure to get information of the specified data set failed.
	ACTION	Check the third and fourth halfword of the qualifier parameter you passed to DBCONTROL for the status. This value should provide extra error information in HPERRMSG format. Notify HP support personnel.
-437	MESSAGE	BTE: Failed to convert @c to [] dbfind.
	MEANING	Failed to convert wildcard or "greater than relation" into a "range relation".
	ACTION	Notify HP support personnel.

-438	MESSAGE	BTE: Bad item# in init btree.
	MEANING	The item number stored in the B-Tree control block is bad.
	ACTION	Notify HP support personnel.
-439	MESSAGE	BTE: Conversion of key from external to internal format failed.
	MEANING	An internal error occurred when converting a key from external format to internal format. This conversion is done prior to searching a B-Tree.
	ACTION	Notify HP support personnel.
-440	MESSAGE	BTE: XM Attach of index file failed.
	MEANING	Attaching a B-Tree to the Transaction Manager (XM) has failed.
	ACTION	Check the third and fourth halfword of the qualifier parameter you passed to DBCONTROL for the status. This value should provide extra error information in HPERRMSG format. Notify HP support personnel.
-441	MESSAGE	BTE: XM Detach of index file failed.
	MEANING	Detaching a B-Tree from the Transaction Manager (XM) has failed.
	ACTION	Check the third and fourth halfword of the qualifier parameter you passed to DBCONTROL for the status. This value should provide extra error information in HPERRMSG format. Notify HP support personnel.
-442	MESSAGE	BTE: RELEASE of index file failed.
	MEANING	Releasing the B-Tree file from GROUP/ACCOUNT security has failed in DBCONTROL mode 13 call.
	ACTION	Check the third and fourth halfword of the qualifier parameter you passed to DBCONTROL for the status. This value should provide extra error information in HPERRMSG format. Notify HP support personnel.
-443	MESSAGE	BTE: SECURE of index file failed.
	MEANING	Securing the B-Tree file (which makes normal GROUP/ACCOUNT security apply to it) has failed in a DBCONTROL mode 13 call.

	ACTION	Check the third and fourth halfword of the qualifier parameter you passed to DBCONTROL for the status. This value should provide extra error information in HPERRMSG format. Notify HP support personnel.
-444	MESSAGE	BTE: DBFIND on non-key field of master.
	MEANING	The item specified in the parameter is not a key field in the master data set. DBFIND on a master data set is allowed only if there is a B-Tree for the data set, and if the DBFIND item is the master data set's key item.
	ACTION	Modify the program, or add a B-Tree to the master data set.
-446	MESSAGE	BTE: Argument 2 specified for relop of (</<=/=/>=/>).
	MEANING	A relational operator (<, <=, >, >=, =) was found in the DBFIND argument, which implies that only one data argument field is expected. However, the argument 2 length field is non-zero, implying that a second argument data field was unnecessarily passed into DBFIND. For relational operators, DBFIND's argument 2 size must be zero.
	ACTION	Modify the program.
-447	MESSAGE	BTE: Failed to build record holding root data information.
	MEANING	A procedure that extracts information from a root file has failed. (Several B-Tree internal routines depend upon this information.)
	ACTION	Notify HP support personnel
-448	MESSAGE	BTE: Failed to setup information for userlabel 0 of data set.
	MEANING	The procedure that writes data to user label #0 of a data set has failed.
	ACTION	Notify HP support personnel.
-449	MESSAGE	BTE: Failed to position index at start of key range.
	MEANING	The procedure to position the B-Tree pointer to the first key that satisfies the current DBFIND has failed.

	ACTION	A REBUILDINDEX may solve the problem. If the problem occurs again, notify HP support personnel.
-451	MESSAGE	BTE: Root version less than "C4."
	MEANING	An attempt was made to add a B-Tree to a data set in an older format database. Normally, DBUTIL's ADDINDEX command, and DBCONTROL mode 13, will update an older format database to "C4" level prior to adding a B-Tree.
	ACTION	Notify HP support personnel.
-452	MESSAGE	BTE: Key length greater than 252 bytes (maximum index key size).
	MEANING	The key length of the data item you want to create a B-Tree index on is too long. The length can not exceed 252 bytes.
	ACTION	Consider reducing the size of the item used as the data set key. The 252 byte limitation is a result of a KSAM/iX limitation. B-Trees are currently implemented using KSAM/iX files.
-458	MESSAGE	DBOPEN failed, out of disk space.
	MEANING	The system does not have enough disk space to create run-time control blocks and global structures.
	ACTION	Notify your system administrator about the disk space and try again after disk space is made available.
-1000	MESSAGE	Switch failure, INFO *nn*, SUBSYS *nn*
	MEANING	The switch to Native Mode failed. SUBSYS *nn* is the subsystem that failed. INFO *nn* is the error number returned from that subsystem.
	ACTION	Notify HP support personnel.
-1001	MESSAGE	Switch to CM failed on CX'PCBXIMAGE
	MEANING	Internal error. This error is returned when DBOPEN calls CM CX'PCBXIMAGE and encounters a switch to CM failure.
	ACTION	Notify HP support personnel.

-1002	MESSAGE	HPLOADCMPROCEDURE failed on CX'PCBXIMAGE
	MEANING	Internal error. This error is returned when DBOPEN calls HPLOADCMPROCEDURE for CM CX'PCBXIMAGE and fails.
	ACTION	Notify HP support personnel.
-1003	MESSAGE	Switch to NM failed on DBxxx, INFO *nn* SUSSYS *nn*
	MEANING	The switch to Native Mode failed. DB*xxx* is the TurboIMAGE/XL intrinsic. SUBSYS *nn* is the subsystem that failed. INFO *nn* is the error number returned from that subsystem.
	ACTION	Notify HP support personnel.
-1004	MESSAGE	HPLOADNMPROC FAILED ON CM *intrinsic name*
	MEANING	An internal error has occurred.
	ACTION	Notify HP support personnel.

Library Procedure Exceptional Conditions

10	MESSAGE	Beginning of file
	MEANING	DBGET has encountered beginning of file during a backward serial read. (No entries exist before the one previously accessed.)
	ACTION	Appropriate action depends on program design.
11	MESSAGE	End of file
	MEANING	DBGET has encountered the end of file during a forward serial read. (No entries exist beyond the most recently accessed one.)
	ACTION	Appropriate action depends on program design.
12	MESSAGE	Directed beginning of file
	MEANING	DBGET has been called for a directed read with a record number less than 1.
	ACTION	Appropriate action depends on program design.
13	MESSAGE	Directed end of file
	MEANING	DBGET has been called for a directed read with a record number greater than the capacity of data set.
	ACTION	Appropriate action depends on program design.
14	MESSAGE	Beginning of chain
	MEANING	DBGET has encountered end of chain during a backward chained read.
	ACTION	Appropriate action depends on program design.
15	MESSAGE	End of chain
	MEANING	DBGET has encountered end of chain during a forward chained read.
	ACTION	Appropriate action depends on program design.

16	MESSAGE	Data set full
	MEANING	DBPUT has discovered that data set is full and cannot add a record as requested. (Message received when return status word 1 is 16, and word 3 is 0.)
	ACTION	Restructure database with larger capacity for this data set. (See chapter 8 for information on changing data set capacity in conjunction with DBUNLOAD and DBLOAD.)
16	MESSAGE	DBPUT cannot expand *dataset#*: Data set at maximum capacity
	MEANING	The current capacity is at the maximum allowed, the data set cannot be expanded. (Message received when return status word 1 is 16, and word 3 is 1.)
	ACTION	If necessary, use DBChange Plus or third-party software to increase the maximum capacity. (See *MPE/iX Release 5.0 Communicator* for information on using DBChange Plus. See chapter 3 for information on detail data set capacity parameters and chapter 5 for DBPUT.)
16	MESSAGE	DBPUT *dataset#* incomplete expansion: File system error *#*
	MEANING	DBPUT cannot expand to the incremental amount due to a file system error. (Message received when return status word 1 is 16, and word 3 is 2.)
	ACTION	Determine the file system error (see file system manual) and correct the problem identified. (See chapter 3 for information on detail data set capacity parameters and chapter 5 for DBPUT.)
16	MESSAGE	DBPUT cannot expand *dataset#*: Out of disc space (FSERR *#*)
	MEANING	There is no disc space for DBPUT to expand the data set. (Message received when return status word 1 is 16, and word 3 is 3.)
	ACTION	Increase disc space and rerun your application. (See chapter 3 for information on detail data set capacity parameters and chapter 5 for DBPUT.)

17	MESSAGE	`No master entry` (See below for additional status code 17 message.)
	MEANING	`DBFIND` is unable to locate master data set entry (chain head) for specified detail data set's search item value.
	ACTION	Appropriate action depends on program design.
17	MESSAGE	`No entry` (See above for additional status code 17 message.)
	MEANING	`DBGET` mode 1 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 key item value. `DBGET` has discovered that selected record is empty (does not contain an entry) when called with mode 4. `DBUPDATE` or `DBDELETE` was called when the "current record" was not established or was empty.
	ACTION	Appropriate action depends on program design.
18	MESSAGE	`Broken chain`
	MEANING	For `DBGET` with *mode* 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). For `DBGET` with *mode* 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). This error can occur in `DBOPEN` access modes 1, 5, and 6 because another user can make database 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/XL in the DBB 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`.
	ACTION	Begin reading chain again from first or last entry.

20	MESSAGE	Database locked or contains locks
	MEANING	DBLOCK Mode 2: The database cannot be locked. Refer to value of status element three: If 0, database already locked; if 1, database contains locked sets or entries. Mode 4, 6: The lock cannot be granted because the whole database is already locked.
	ACTION	Appropriate action depends on program design.
22	MESSAGE	Data set locked by another process
	MEANING	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.
	ACTION	Appropriate action depends on program design.
23	MESSAGE	Entries locked within set
	MEANING	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 or 6 only.
	ACTION	Appropriate action depends on program design.
24	MESSAGE	Item conflicts with current locks
	MEANING	Lock descriptors passed to DBLOCK specify a data item that is different than one used to set existing locks. TurboIMAGE/XL allows no more than one data item per data set to be used at one time for locking purposes. Returned in DBLOCK modes 5 and 6 only.
	ACTION	Appropriate action depends on program design.
25	MESSAGE	Entries already locked
	MEANING	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.
	ACTION	Appropriate action depends on program design.

26	MESSAGE	Lock not performed since deadlock would result.
	MEANING	If it is a self-deadlock, the database or data set is already locked by the same process but with a different open count. If it is locked by another process, it is because of the DBLOCK call from another user that the real deadlock is encountered.
	ACTION	To avoid the self-deadlock, modify the program. If it is a real deadlock caused by another process, call DBUNLOCK. Use the SHOW *dbname* LOCKS command of DBUTIL on all databases that the process has opened in order to determine the source of the deadlock.
41	MESSAGE	DBUPDATE attempted to modify value of critical item--key, search or sort (See below for additional status code 41 messages.)
	MEANING	DBUPDATE has been asked to change the value of a master data set key item or a detail data set search or sort item. For master data sets and regardless of the critical item update (CIUPDATE) option settings for the database and the current process, this message is always returned if you attempt to use DBUPDATE on key items. For detail data sets, this message is returned when you attempt to use DBUPDATE on a search or sort item and the CIUPDATE option setting for the database or a particular process does not permit this type of modification.
	ACTION	Correct call, or notify user that item cannot be updated. For detail data sets only, the application can be redesigned to use the CIUPDATE option.

41	MESSAGE	DBUPDATE: No chain head (master entry) for path *decimal integer: nn*
	MEANING	User has called DBUPDATE to update a detail data set entry with a search item value that does not match any existing key item value in the corresponding manual master data set. The digits *nn* identify the offending path number established by the sequence in which the detail data set search items appear in the set part of the schema. This message is associated with the critical item update (CIUPDATE) option; see chapter 4 for information about this option.
	ACTION	Notify user that the new data item value must already exist in the associated manual master data set. If the new value is to be allowed, add the manual master data set entry and try again.

41	MESSAGE	DBUPDATE: Full chain for path *decimal integer: nn* (contains 2,147,483,647 entries (See above and below for additional status code 41 messages.)
	MEANING	User has called DBUPDATE to update a detail data set entry associated with an automatic master data set chain that already contains 2,147,483,647 values, the maximum allowable entries. The digits *nn* identify the offending path number established by the sequence in which the detail data set search items appear in the set part of the schema. This message is associated with the critical item update (CIUPDATE) option; see chapter 4 for information about this option.
	ACTION	Consult with your database administrator to ensure the database has no structural damage. It is likely that the database contains an invalid chain count.

41	MESSAGE	DBUPDATE: Full automatic master for path *decimal integer: nn* (See above and below for additional status code 41 messages.)
	MEANING	Automatic master data set is full. User has called DBUPDATE to update a search item in a detail data set, and the corresponding key item value could not be added to the automatic master data set. The digits *nn* identify the offending path number established by the sequence in which the detail data set search items appear in the set part of the schema. This message is associated with the critical item update (CIUPDATE) option; see chapter 4 for information about this option.
	ACTION	Your database administrator needs to restructure the database, increasing the capacity of the automatic master data set. Refer to chapter 8 in this manual for information on database restructuring.
41	MESSAGE	DBUPDATE: Full automatic master synonym chain for path *decimal integer: nn* (See above for additional status code 41 messages.)
	MEANING	User has called DBUPDATE to update a detail data set search item associated with an automatic master data set that cannot accommodate another new value on the synonym chain for that path number. The digits *nn* identify the offending path number established by the sequence in which the detail data set search items appear in the set part of the schema. This message is associated with the critical item update (CIUPDATE) option; see chapter 4 for information about this option.
	ACTION	Ask your database administrator to verify that the synonym chain is indeed full and inspect your database for possible structural damage.
42	MESSAGE	Read only item
	MEANING	DBUPDATE has been asked to change the value of a data item for which the user does not have write access.
	ACTION	Notify user, cannot update item. Or change password in program.

43	MESSAGE	Duplicate key item value
	MEANING	DBPUT has been asked to insert a data entry into a master data set with a key item value which already exists in the data set.
	ACTION	Appropriate action depends on program design.
44	MESSAGE	Can't delete master entry with non-empty detail chains
	MEANING	DBDELETE has been asked to delete a master data set entry which still has one or more non-empty chains.
	ACTION	Appropriate action depends on program design.
49	MESSAGE	Illegal buffer address
	MEANING	Calling program's buffer (identified by buffer parameter) address of DBGET or DBINFO is an unsupported memory address.
	ACTION	Correct the buffer parameter with a valid address. CM user stack, NM user stack, and the NM heap are valid memory areas in which to create a buffer parameter.
50	MESSAGE	Buffer too small
	MEANING	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.
	ACTION	Correct procedure call, or change buffer name or size.
51	MESSAGE	STACK OVERFLOW FOR BASIC - IMAGE INTERFACE.
	MEANING	Insufficient stack space for the BASIC program to execute.
	ACTION	Increase the stack size using the NMSTACKSIZE option with the LINK command.

52	MESSAGE	INVALID PARAMETER FOR BASIC - IMAGE INTERFACE.
	MEANING	The call to TurboIMAGE/XL intrinsic has invalid parameters.
	ACTION	Correct the application.
53	MESSAGE	INVALID PARAMETER TYPE FOR BASIC - IMAGE INTERFACE.
	MEANING	The call to TurboIMAGE/XL intrinsic has invalid type for the parameter(s).
	ACTION	Correct the application.
60	MESSAGE	Database access disabled
	MEANING	DBOPEN has been called when the database has been disabled for access.
	ACTION	Notify database administrator.
61	MESSAGE	Database opened more than 63 times by same process.
	MEANING	DBOPEN has been called when the specified database has already been opened 63 times by the same process.
	ACTION	Correct your program so that DBOPEN does not open the same database more than 63 times.
62	MESSAGE	DBG Control Block is Full
	MEANING	Lock area within DBG is full.
	ACTION	Notify your database administrator.

63	MESSAGE	DBG disabled; potential damage; only DBCLOSE allowed
	MEANING	Another process sharing the database has aborted because of logical inconsistency or internal error in TurboIMAGE/XL, leaving DBG in potentially inconsistent state. All user accesses through existing DBG are disabled (except for DBCLOSE, mode 1). Returned by all intrinsics. If the database has been enabled for dumping, an I and a J file should exist. These files can help determine the cause of the error. The database should be recovered. This error has the same effect on a database as a system failure.
	ACTION	Issue a DBCLOSE, mode 1.
64	MESSAGE	No room for DBG entry in PCBX (MPE portion of stack)
	MEANING	The PCBX is full.
	ACTION	Try again when system resources are available.
65	MESSAGE	Can't grant buffer request
	MEANING	DBCONTROL cannot increase the number of buffers as requested by the user. A current count is returned.
	ACTION	Request a smaller number of buffers.
66	MESSAGE	DBG pointed to by root file does not match
	MEANING	Root file is not compatible with DBG.
	ACTION	Notify HP support personnel.
67	MESSAGE	DBU disabled; potential damage; only DBCLOSE allowed
	MEANING	The user's control block is in an inconsistent state.
	ACTION	Issue a DBCLOSE, mode 1.

68	MESSAGE	Bad DBB
	MEANING	Another process sharing the database has aborted because of logical inconsistency or internal error in TurboIMAGE/XL, leaving DBB in potentially inconsistent state. All user accesses through existing DBB are disabled (except for DBCLOSE, mode 1). Returned by all intrinsics. If the database has been enabled for dumping, an I and a J file should exist. These files can help determine the cause of the error. The database should be recovered. This error has the same effect on a database as a system failure.
	ACTION	Issue a DBCLOSE, mode 1.
69	MESSAGE	Bad database
	MEANING	AUTODEFER is active, and the database did not close normally. The database has been corrupted.
	ACTION	Notify your database administrator.
71	MESSAGE	LOGGING NOT ENABLED FOR USER
	MEANING	In order to perform the operation, the user requires an LG (user logging) capability.
	ACTION	Grant the user an LG capability.
72	MESSAGE	TURBOLKT TABLE FULL.
	MEANING	The lock table used to track locks on databases system-wide, TURBOLKT.PUB.SYS, is full.
	ACTION	Notify your HP personnel.
73	MESSAGE	ERROR IN TURBOLKT TABLE OPERATION.
	MEANING	An internal error occurred while reading from or writing to the lock table, TURBOLKT.PUB.SYS.
	ACTION	Notify your HP personnel.

1*nn*	MESSAGE	No chain head (master entry) for path *decimal integer: nn*
	MEANING	User has attempted to add detail data entry with a search item value that does not match any existing key item value in corresponding manual master data set. The digits represented by *nn* identify the offending path number established by the order in which the search items occur in the set part of the schema.
	ACTION	Notify user cannot add entry or add manual master entry and try again.
2*nn*	MESSAGE	Full chain for path *decimal integer: nn* (contains 2,147,483,647 entries)
	MEANING	User has attempted to add detail data entry to a chain which already contains the maximum allowable (2,147,483,647) entries. The digits represented by *nn* identify the offending path number established by the order in which the search items occur in the set part of the schema.
	ACTION	Consult with the database manager. The database could contain an invalid chain count.
3*nn*	MESSAGE	Full automatic master for path *decimal integer: nn*
	MEANING	Automatic master data set is full. Therefore, when the user attempted to add a search item to a detail data set, the corresponding key value could not be added to the automatic master data set. The digits represented by *nn* identify the offending path number established by the order in which the search items occur in the set part of the schema.
	ACTION	Restructure database, increasing capacity of automatic master. (See chapter 8).

4*nn*	MESSAGE	`Full automatic master synonym chain for path` *`decimal integer: nn`*
	MEANING	User has attempted to add a detail data set search item associated with an automatic master data set that cannot accommodate another new value on the synonym chain for that path number. The digits *nn* identify the offending path number established by the sequence in which the detail data set search items appear in the set part of the schema.
	ACTION	Ask your database administrator to verify that the synonym chain is indeed full and inspect your database for possible structural damage.
944	MESSAGE	`Warning: Assuming no message catalog`
	MEANING	No message catalog is available.
	ACTION	Notify your system administrator.
Others	MESSAGE	`Unrecognized return status:` *`nn`*
	MEANING	Return status could not be recognized.
	ACTION	Notify HP support personnel.

Library Procedure Abort Condition Messages in I File

MESSAGE	`BUFFER SUPPLY CRISIS`
MEANING	Internal software inconsistency caused TurboIMAGE/XL to mismanage its buffer space.
ACTION	Notify the database administrator and possibly HP support personnel. Save FID information if it is printed. You might need to perform database recovery procedures. (See chapter 7).
MESSAGE	`CRITICAL LABEL READ ERROR ON` *`data set`*
MEANING	Unable to read label of database file.
ACTION	Notify the database administrator and possibly HP support personnel. Save FID information if it is printed. You might need to perform database recovery procedures. (See chapter 7).

MESSAGE CRITICAL READ ERROR ON *data set*

MEANING While reading database file, MPE file read error was encountered.

ACTION Notify the database administrator and possibly HP support personnel. Save FID information if it is printed. You might need to perform database recovery procedures. (See chapter 7).

MESSAGE ERROR *intrinsic name* STATUS *return status*;
DBERROR/DBEXPLAIN FAILED I *info* S *subsys*

MEANING The specified intrinsic encountered an error. ERROR displays the intrinsic. STATUS is the return status from the intrinsic. The application called either DBERROR or DBEXPLAIN. The switch to Native Mode failed. I is the error number returned by the subsystem. S is the subsystem that failed.

ACTION Notify HP support personnel.

MESSAGE ESCAPE FROM TURBOIMAGE/XL; PC: *$pcode* CODE: *$escape code*

MEANING A hardware or software trap occurred. *pcode* and *escape code* are hex numbers. The first four hex digits represent the error code; the last four represent the subsystem number.

ACTION Notify the database administrator and HP support personnel if necessary. Save the *escape code* number.

MESSAGE HPUNLOADCMPROCEDURE FAILED ON *intrinsic name* INFO *info*
SUBSYS *subsystem*

MEANING A problem was encountered when switching to Compatibility Mode (CM). INFO is the error returned by the subsystem. SUBSYS is the subsystem which failed.

ACTION Notify HP support personnel.

MESSAGE LABEL WRITE ERROR ON *data set*

MEANING Unable to complete the writing of a user label on database file.

ACTION Notify the database administrator and HP support personnel if necessary.

MESSAGE LOST FREE SPACE IN *data set*

MEANING Internal software inconsistency caused unused record locations in data set to become lost or unavailable. File information display is printed for the data set.

ACTION Notify the database administrator and HP support personnel if necessary.

MESSAGE NEGATIVE MOVE ATTEMPT: *n*

MEANING An internal software inconsistency has been found while attempting to move data to or from user's stack.

ACTION Notify the database administrator and HP support personnel if necessary.

MESSAGE THE *intrinsic name parm* PARAMETER NOT ALIGNED ON A 16-BIT BOUNDARY

MEANING The designated parameter, *parm*, is 8-bit aligned.

ACTION Modify your application so that the *parm* parameter is 16-bit (halfword) aligned.

MESSAGE THE USER CANNOT BE IN SPLIT-STACK MODE

MEANING An application must be running on the user's stack when calling a TurboIMAGE/XL intrinsic. It is not permitted to run in split-stack mode. This message is only applicable if you are running in Compatibility Mode.

ACTION Modify your Compatibility Mode application so that it is not running in split-stack mode.

MESSAGE UNABLE TO CLOSE DATA SET

MEANING Unable to close a database file.

ACTION Modify your application so that the *parm* parameter is 16-bit (halfword) aligned.

MESSAGE UNABLE TO OPEN A *data set*

MEANING Unable to open database file.

ACTION Modify your application so that the *parm* parameter is 16-bit (halfword) aligned.

MESSAGE WRITE ERROR ON *data set*

MEANING While writing into database file, MPE file write error was found.

ACTION Modify your application so that the *parm* parameter is 16-bit (halfword) aligned.

MESSAGE WRONG NUMBER OF PARAMETERS OR BAD ADDRESS FOR PARAM *#n*

MEANING An address referenced by one of the parameters is not within the user's stack area in memory. 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.

ACTION Modify your application so that the *parm* parameter is 16-bit (halfword) aligned.

Utility Error Messages

Two types of error messages are generated by the utility programs. The first type is conditional errors, which occur when a utility cannot begin a function. Conditional errors do not cause utility program termination; this is discussed in more detail below. The second type is unconditional errors, which occur when a utility cannot complete a function. Unconditional errors usually cause program termination.

Conditional errors are associated with accessing the specified database. If you are in session mode, conditional errors can be corrected without terminating the run. After printing the error message, all utility programs, with the exception of DBUTIL, reprompt with the message "WHICH DATABASE?", allowing you to reenter the database reference. If you wish to terminate the utility program at this point, you can type a carriage return with or without leading blanks. DBUTIL, after printing the error message, reprompts with two greater than symbols (>>) at which point you can exit or enter another command. When DBUTIL encounters any errors while executing, a job control word (JCW) is set to 99. You can check the JCW in batch mode to determine whether or not a command was accepted. If you are in job mode and a conditional error occurs while a function is executing, that function is terminated.

The messages associated with conditional and unconditional errors are described along with their meanings in the sections entitled "Utility Program Conditional Messages" and "Utility Program Unconditional Messages."

Utility Program Conditional Messages

MESSAGE	ACCESS DISABLED UNTIL RECOVERY HAS COMPLETED
MEANING	Database is disabled for access, and recovery is enabled.
ACTION	Users will be able to access the database after the recovery process has completed.

MESSAGE	ACCT NAME MORE THAN 8 CHARACTERS
MEANING	The *acctname* part of DBNAME2 is too long.
ACTION	Use EDITOR to change the FILE command.

MESSAGE	APPARENT BACKUP/LOGFILE MISMATCH
MEANING	A call to DBPUT, DBUPDATE, or DBDELETE, that apparently succeeded when logging to the original database has failed during recovery.
ACTION	Make sure the proper backup database was restored and that the matching log file was used for recovery.

MESSAGE AUTODEFER MUST BE DISABLED BEFORE ILR CAN BE ENABLED

MEANING AUTODEFER option in DBUTIL has been enabled for the database. ROLLBACK or ILR cannot be enabled while deferred output is enabled.

ACTION Use DBUTIL to disable AUTODEFER and enable ILR or ROLLBACK for the database.

MESSAGE AUTODEFER MUST BE DISABLED BEFORE ROLLBACK CAN BE ENABLED

MEANING The rollback recovery option cannot be used when the database has been enabled for AUTODEFER.

ACTION Use DBUTIL to disable AUTODEFER and enable ROLLBACK for the database. (Refer to chapters 7 and 8 for more information on roll-back recovery.)

MESSAGE BACKUP DATABASE AND LOG FILE DO NOT MATCH

MEANING The *logid* of the log file and the *logid* in the database being recovered do not match; i.e., the correct log file is not being used for recovery.

ACTION Make sure that the database is using the correct log file.

MESSAGE BAD DATABASE REFERENCE

MEANING Database reference following the utility program :RUN command has a syntax error.

ACTION Correct the error in session mode or press return to terminate the program.

MESSAGE BAD MAINTENANCE WORD

MEANING User invoking the utility is not the creator of the referenced database and has not supplied the correct maintenance word.

ACTION Correct the error in session mode or press return to terminate the program.

MESSAGE CANNOT ENABLE BREAK KEY

MEANING Internal error.

ACTION Contact your database administrator.

MESSAGE CANNOT OPEN DATABASE

MEANING The database cannot be opened at this time. It could already be open in a mode that does not allow concurrent access.

ACTION Try the DBUTIL command again later.

MESSAGE CANNOT PARSE LOG FILE NAME

MEANING Internal error.

ACTION Contact your database administrator.

MESSAGE Cannot process data set file filename

MEANING Cannot read/write to the file due to a file system error. Another process may have exclusive access to the data set.

ACTION Check if the data set file exists or is locked. Retry.

MESSAGE CAN'T RECOVER DATA BASES IN STATISTICAL MODE

MEANING The CONTROL STATS command was entered subsequent to a RECOVER command.

ACTION Rerun DBRECOV without using statistical mode, or use statistical mode without specifying databases or recovery.

MESSAGE CAN'T OPEN DATABASE - *DBERROR message*

MEANING Database cannot be opened.

ACTION Check the DBERROR message to determine the problem.

MESSAGE CHECKSUM FAILURE ON LOG RECORD #*n* - EOF ASSUMED

MEANING Can occur if the system failed and a startup recovery is not performed. Also can occur if a statistical recovery is used against an open log file.

ACTION You may want to use a start recovery operation in the future, but this is not absolutely necessary.

MESSAGE COULD NOT FIND OPEN DATABASE IN DATABASE TABLE

MEANING Internal error.

ACTION Contact your database administrator.

MESSAGE CURRENT SINGLE TRANSACTION DOESN'T MATCH LOG TRANSACTION

MEANING Internal error.

ACTION Contact your database administrator.

MESSAGE CURRENT TRANSACTION DOESN'T MATCH LOG FILE TRANSACTION

MEANING Internal error.

ACTION Contact your database administrator.

MESSAGE DATABASE-ACCESS FILE DOES NOT EXIST

MEANING No such file exists in the logon user or account.

ACTION Check the files in group to determine the correct file name.

MESSAGE DATABASE-ACCESS FILE NAME MAY NOT BE QUALIFIED

MEANING The DBA file name cannot be qualified with group and account.

ACTION Log on in the same group and account as the database- access file.

MESSAGE DATABASE-ACCESS FILENAME TOO LONG

MEANING The file name has more than six characters.

ACTION Rename the file and try DBUTIL again.

MESSAGE DATABASE ALREADY CREATED

MEANING You have invoked DBUTIL in the CREATE mode and specified the name of a database which already exists.

ACTION In session mode, correct the error or press return to terminate program.

MESSAGE DATABASE ALREADY DISABLED FOR *option*

MEANING The *option* has already been disabled for this database.

ACTION Press return to terminate the program.

MESSAGE DATABASE CONFIGURED FOR INDEXING USING *third-party product name*

MEANING This message is returned if you are using third-party indexing (TPI). The message is generated from the DBUTIL ENABLE *database name* FOR INDEXING command if this database has already been enabled for indexing.

ACTION This message is generated from the DBUTIL ENABLE command, and you can disregard it if you want the database to remain enabled for indexing. You can use the DBUTIL SHOW command to verify whether or not indexing is enabled for a particular third-party product.

MESSAGE DATABASE ALREADY ENABLED FOR *option*

MEANING The *option* has already been enabled for this database.

ACTION Press return to terminate the program.

MESSAGE DATABASE CANNOT BE ERASED UNTIL RECOVERY HAS COMPLETED

MEANING Erasing database is attempted when recovery was started but not completed, or while recovery is in RESTART mode.

ACTION Press return to terminate program. After the recovery process has completed, the user can run DBUTIL again.

MESSAGE DATABASE CANNOT BE PURGED UNTIL RECOVERY HAS COMPLETED

MEANING Purging the database is attempted when recovery was started but not completed, or while recovery is in RESTART mode.

ACTION Press return to terminate program. After the recovery process has completed, the user can run DBUTIL again.

MESSAGE DATABASE CONFIGURED FOR INDEXING USING *third-party product name*, BUT NOT ENABLED FOR INDEXING

MEANING This message is returned if you are using third-party indexing (TPI). The message is generated from the DBUTIL DISABLE *database name* FOR INDEXING command if the database has not yet been enabled for indexing. Even if the database is configured for indexing, by default it remains disabled for indexing until the database is enabled for indexing either by the TPI configuration utility or the DBUTIL ENABLE command.

ACTION Disregard the message if you want the database to remain disabled for indexing.

MESSAGE DATABASE CONTAINS INVALID LOGGING IDENTIFIER

MEANING The log identifier in the database root file does not exist in the MPE logging identifier table.

ACTION Use DBUTIL to set a valid log identifier into the database.

MESSAGE DATABASE HAS BEEN MODIFIED SINCE LAST RESTORE

MEANING The DBSTORE flag has not been reset. This can mean a backup database has not been restored, or that there have been modifications since the last restoration.

ACTION Restore the backup database, or use the CONTROL NOSTORE option and reenter the RECOVER command.

MESSAGE DATABASE *n* HASN'T BEEN CREATED YET

MEANING The database creator must run DBUTIL and CREATE the database.

ACTION In session mode, correct the error or press return to terminate the program.

MESSAGE DATABASE IN USE

MEANING Utility program cannot get exclusive access to the referenced database due to other current users.

ACTION In session mode, press return to terminate the program.

MESSAGE DATABASE IS NOT ATTACHED TO ANY HP SQL DBEnvironment

MEANING This message is returned if you are *not* using ALLBASE/Turbo CONNECT (ATC) and specify an ATC-specific option. In this case, the DETACH option of the DBUTIL PURGE command is specified, but the database is not attached to any ALLBASE/SQL database environment (DBEnvironment).

ACTION Contact your database administrator.

MESSAGE DATABASE IS NOT EMPTY. LANGUAGE CANNOT BE CHANGED

MEANING The language can be changed only on a virgin root file or an empty database.

ACTION In session mode, press return to terminate the program.

MESSAGE DATABASE IS REMOTE

MEANING You must be logged on to the same group and account containing the root file to use DBUTIL.

ACTION Do a remote logon and run DBUTIL from your remote session.

MESSAGE DATABASE LANGUAGE NOT SYSTEM SUPPORTED

MEANING The language of the database is not currently configured on your system.

ACTION Notify the database administrator.

MESSAGE DATABASE NAME MAY NOT BE QUALIFIED

MEANING The database name cannot be qualified with group and account.

ACTION Log on to the same group and account as the database and reenter the command.

MESSAGE DATABASE NAME TOO LONG

MEANING The database name you specified has more than six characters.

ACTION Try the command again with the correct name.

MESSAGE DATABASE NOT ENABLED FOR RECOVERY

MEANING The recovery flag in the database root file is disabled.

ACTION Be sure you are running recovery against the appropriately restored database. You can override the disable flag with DBUTIL if necessary.

MESSAGE DATABASE OR ACCESS FILE DOES NOT EXIST

MEANING No such file exists in logon group or account.

ACTION Check the files in your group to find the correct file name.

MESSAGE DATABASE OR FILE ACCESS NAME REQUIRED

MEANING The command must include the database or access file name.

ACTION Reenter the command.

MESSAGE DATABASE OR FILE ACCESS NAME TOO LONG

MEANING File name has more than six characters.

ACTION Reenter the command with the correct name.

MESSAGE DATABASE REQUIRES CREATION

MEANING The database creator must run the DBUTIL program in CREATE mode prior to executing DBUNLOAD, DBLOAD, or DBUTIL in ERASE mode.

ACTION In session mode, correct the error or press return to terminate the program.

MESSAGE DATA SET *n* IS MISSING

MEANING The data set could have been purged, or was not restored.

ACTION Contact your database administrator or system manager.

MESSAGE DBLOAD TO DIFFERENT DATABASE NAME MAINTENANCE WORD IS NOT ALLOWED

MEANING The DBLOAD database name and maintenance word must be the same as those used for DBUNLOAD.

ACTION Use a database name and maintenance word which were used for the DBUNLOAD.

MESSAGE DBLOAD TO DIFFERENT GROUP OR ACCOUNT IS NOT ALLOWED.

MEANING DBLOAD must be done from the same group and account as the DBUNLOAD file.

ACTION Log on to the same group and account from which the DBUNLOAD was done. Then use DBLOAD again.

MESSAGE DBNAME DBSTORE OF TIME, DATE REQUIRED

MEANING DBSTORE timestamp in database does not match those in DBOPEN records.

ACTION Restore the proper database or use the proper log file. You can also use the NOSTAMP option in recovery.

MESSAGE DBNAME1 MORE THAN SIX CHARACTERS

MEANING *dbname*1 is too long.

ACTION Use EDITOR to change the FILE command.

MESSAGE DETACH FAILED FROM *DBEnvionment name*

MEANING This message is returned if you are using ALLBASE/Turbo CONNECT (ATC). The message is generated from the DBUTIL PURGE command used with the DETACH option. The database is attached to the indicated ALLBASE/SQL database environment (DBEnvironment), but the DETACH option is unsuccessful due to ATC error *nn*. The PURGE command will continue processing after the warning is issued.

ACTION Notify your database administrator.

MESSAGE 'DEV' MISSING

MEANING Either the keyword DEV or the equal sign (=) following it was not found.

ACTION Use EDITOR to change the FILE command.

MESSAGE DEVICE NOT 'DISC'

MEANING The string following the "#" sign was not "DISC."

ACTION Use EDITOR to change the FILE command.

MESSAGE DSDEVICE DOESN'T MATCH FILE COMMAND

MEANING The *dsdevice* name in the DSLINE command was not the same as the *dsdevice* name in the FILE command.

ACTION Use EDITOR to change the command which is in error.

MESSAGE DSDEVICE NAME MORE THAN 8 CHARACTERS

MEANING The *dsdevice* name is too long.

ACTION Use EDITOR to change the command which is in error.

MESSAGE DUPLICATE FILE NAME

MEANING The required file name is already assigned to another file. Example: If the ORDERS database requires six data sets (ORDERS01–ORDERS06), then a previously defined file ORDERS03 will trigger this message. This can occur if a database is not purged before running DBRESTOR.

ACTION In session mode, correct the error or press return to terminate the program.

MESSAGE EMBEDDED BLANK IN DBNAME2

MEANING One of the periods separating the user name from the group name or the group name from the account name was preceded or followed by a blank.

ACTION Use EDITOR to change the MPE FILE command.

MESSAGE ERROR READING ROOT FILE LABEL

MEANING DBUTIL is unable to read the root file.

ACTION Contact your HP systems engineer.

MESSAGE ERROR READING ROOT FILE RECORD

MEANING DBUTIL is unable to read a root file record.

ACTION Contact your HP systems engineer.

MESSAGE ERROR WRITING ROOT FILE LABEL

MEANING DBUTIL found an error while writing the root file label.

ACTION Contact your HP systems engineer.

MESSAGE ERROR WRITING ROOT FILE RECORD

MEANING DBUTIL found an error while writing to a root file record.

ACTION Contact your HP systems engineer.

MESSAGE EXCEEDS ACCOUNT DISC SPACE

MEANING Amount of space required by the database plus the space already assigned to other files exceeds the disk space available to the account.

ACTION Request the system manager to increase account's disk space.

MESSAGE EXCEEDS GROUP DISC SPACE

MEANING The amount of disk space required by the database plus the space assigned to other files in the group exceeds the amount of disk space available to the account.

ACTION Request the system manager to increase the group's disk space.

MESSAGE EXPECTED DATABASE STATE TO BE "CLOSED"

MEANING Internal error.

ACTION Contact your database administrator.

MESSAGE	FCHECK FAILURE FCLOSE FAILURE FCONTROL FAILURE FENTRY FAILURE FGETINFO FAILURE
MEANING	These are exceptional errors indicating a hardware or software failure.
ACTION	Notify your database administrator of the failure.

MESSAGE	FFILEINFO INTRINSIC FAILED
MEANING	Internal error.
ACTION	Contact your database administrator.

MESSAGE	FGETINFO CANNOT VERIFY THAT FILE IS TYPE "LOG"
MEANING	Internal error.
ACTION	Contact your database administrator.

MESSAGE	FGETINFO INTRINSIC FAILED
MEANING	Internal error.
ACTION	Contact your database administrator.

MESSAGE	FILE CODE IS NOT 0.
MEANING	The database-access file to be activated has a file code of other than 0.
ACTION	Be sure you have the correct EDITOR file.

MESSAGE	FILE EQUATES ARE ILLEGAL FOR DATABASE AND DATABASE ACCESS FILE
MEANING	DBUTIL does not allow you to equate the name of the database or database-access file to another file with the MPE FILE command.
ACTION	Use the MPE RESET command to cancel the FILE command. You can either break and resume execution or exit DBUTIL and run it again.

MESSAGE	FILE *file name* RECSIZE TOO SMALL: LOG(S) TRUNCATED.
MEANING	User recovery file size is too small to hold the largest log record.
ACTION	Increase record size or use variable length records.

MESSAGE	FILE NAME MAY NOT BE QUALIFIED
MEANING	The file name cannot be qualified with group and account.
ACTION	Log on to the same group and account and reenter the command.

MESSAGE FILE NAME MORE THAN SIX CHARACTERS

MEANING The file name part of *dbname*2 is too long.

ACTION Use EDITOR to change the FILE command.

MESSAGE FOPEN FAILURE ON logfilename:ferrmsg

MEANING You cannot open the log file.

ACTION Examine ferrmsg to determine the cause.

MESSAGE FREAD ERROR ON ASCII ACCESS FILE

MEANING Exceptional errors indicate a hardware or software failure.

ACTION Notify the database administrator of the error.

MESSAGE FWRITE ERROR ON filename:ferrmsg

MEANING FWRITE failure on user recovery file.

ACTION Check ferrmsg to determine the cause.

MESSAGE GROUP NAME MORE THAN 8 CHARACTERS

MEANING The group name part of *dbname*2 is too long.

ACTION Use EDITOR to change the FILE command.

MESSAGE ILR MUST BE DISABLED BEFORE AUTODEFER CAN BE ENABLED

MEANING Deferred output option (AUTODEFER) cannot be used when the database has been enabled for ILR or ROLLBACK.

ACTION Use DBUTIL to disable ILR or ROLLBACK and enable AUTODEFER. (Refer to chapters 7 and 8 for more information on AUTODEFER.)

MESSAGE INCOMPLETE

PURGE

RELEASE

SECURE

MEANING The operation was interrupted while in process.

ACTION Contact your database administrator.

MESSAGE INCOMPLETE DBNAME2

MEANING The file name or group name is followed by a period and the rest of the record is empty. Or the next character is a semicolon (;).

ACTION Use EDITOR to change the FILE command.

MESSAGE INCOMPLETE LOCAL PART

MEANING 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.

ACTION Use EDITOR to change the HELLO command in the RDBA access file.

MESSAGE INCOMPLETE REMOTE PART

MEANING The period following the *rusername* was missing, or the *raccountname* was missing, or the *raccountname* was followed by a comma and the *rgroupname* was missing.

ACTION Use EDITOR to change the HELLO command in the RDBA access file.

MESSAGE INSUFFICIENT DISC SPACE

MEANING The amount of disk space required for the database is not available on the system.

ACTION Consult with the database administrator about disk space requirements.

MESSAGE INSUFFICIENT VIRTUAL MEMORY

MEANING Not enough virtual memory is available to open and access the database.

ACTION Try running the utility later when the system is not as busy.

MESSAGE INTERNAL ERROR--DATABASE CONTAINS INVALID LOGGING IDENTIFIER

MEANING The logging ID configured in the database through DBUTIL is not configured through the MPE GETLOG command.

ACTION Contact your database administrator.

MESSAGE INVALID ACCESS CLASS NUMBER

MEANING The access class should be a number between 1 and 63.

ACTION Reenter with a valid access class number.

MESSAGE INVALID CHARACTER IN PASSWORD

MEANING Password does not consist of allowed characters.

ACTION Check the password specified and reenter the command.

MESSAGE INVALID COMMAND

MEANING Command is unknown to the utility.

ACTION Check spelling or see manual for legal commands.

MESSAGE INVALID CONTENTS OF ASCII ACCESS FILE ACTIVATION / DEACTIVATION FAILED

MEANING The file specified is either

- Not a database-access file, or
- It does not contain the valid format and parameters.

ACTION Either

- Verify the correct database-access file, or
- Use EDITOR to make the required changes and try accessing again.

MESSAGE INVALID DATABASE CONTROL BLOCK

MEANING TurboIMAGE/XL encountered an inconsistency in the control blocks.

ACTION Contact your system engineer.

MESSAGE INVALID DATABASE ACCESS FILE NAME

MEANING The file name or database name must be 1 to 6 characters starting with a letter.

ACTION Reenter the command with the correct name.

MESSAGE INVALID DATABASE NAME

MEANING The file name or database name must be 1 to 6 characters starting with a letter.

ACTION Reenter the command with the correct name.

MESSAGE INVALID DATABASE NAME OR ACCESS FILE NAME

MEANING The file name or database name must be 1 to 6 characters starting with a letter.

ACTION Reenter the command with the correct name.

MESSAGE INVALID DB INTRINSIC CODE IN LOG RECORD CODE FIELD

MEANING Internal error.

ACTION Contact your database administrator.

MESSAGE INVALID DELIMITER

MEANING The command or a space is incorrectly positioned.

ACTION Use HELP to check command syntax and reenter the command.

MESSAGE INVALID FIRST LOG FILE NAME

MEANING Internal error.

ACTION Contact your database administrator.

MESSAGE INVALID FIRST LOG FILE SEQUENCE NUMBER

MEANING Internal error.

ACTION Contact your database administrator.

MESSAGE INVALID FIRST LOG FILE TIMESTAMP

MEANING Internal error.

ACTION Contact your database administrator.

MESSAGE INVALID IMAGE LOG RECORD DETECTED

MEANING DBRECOV encountered an undefined log record code.

ACTION Notify the database administrator and HP support personnel.

MESSAGE INVALID LANGUAGE

MEANING Language name or number has invalid characters.

ACTION Retype the correct language name or number.

MESSAGE INVALID LOG CODE IN LOG RECORD

MEANING Internal error.

ACTION Contact your database administrator.

MESSAGE INVALID LOG DEVICE TYPE IN LOG RECORD

MEANING Internal error.

ACTION Contact your database administrator.

MESSAGE INVALID LOG FILE DEVICE TYPE

MEANING Internal error.

ACTION Contact your database administrator.

MESSAGE INVALID MAINTENANCE WORD

MEANING An incorrect maintenance word was used.

ACTION Check the maintenance word specified and reenter the command.

MESSAGE INVALID NUMBER OF BUFFERS SPECIFIED

MEANING The number of buffers must be between 4 and 255.

ACTION Reenter the DBUTIL SET command with a valid number.

MESSAGE INVALID NUMBER OF USERS SPECIFIED

MEANING The minimum number of users allowed is 1 and the maximum is 120. The ranges of users specified must be in ascending order.

ACTION Reenter the DBUTIL SET command with a valid number.

MESSAGE INVALID PARAMETER

MEANING You specified an incorrect parameter.

ACTION Use HELP to check the command format and reenter the command.

MESSAGE INVALID PASSWORD FOR DATABASE LOGGING IDENTIFIER

MEANING Log password does not correspond with the log identifier used with the MPE GETLOG command.

ACTION Use DBUTIL to set the valid log identifier password into the database.

MESSAGE INVALID SUBSYSTEM ACCESS FLAG IN ROOT FILE

MEANING TurboIMAGE/XL detected an invalid value for the subsystem access in the root file. Valid accesses are READ, R/W, and NONE.

ACTION Reset the subsystem access flag using the DBUTIL SET command.

MESSAGE INVALID (*x*) IN COLUMN *y*

MEANING The *x* represents the invalid character found in column *y*.

ACTION Use EDITOR to change the database-access file record which returns the error message.

MESSAGE LANGUAGE MUST NOT BE LONGER THAN 16 CHARACTERS

MEANING The language name is too long.

ACTION Retype the correct language name.

MESSAGE LANGUAGE NOT SUPPORTED

MEANING The language is not supported on your system or is not a valid language name.

ACTION Check with your database administrator for the configuration of the language, or enter a valid language name or number.

MESSAGE LESS THAN 3 RECORDS IN FILE!

MEANING The database-access file does not contain a FILE, DSLINE, or HELLO command.

ACTION Use EDITOR to create the missing records.

MESSAGE LOCAL ACCT NAME IS TOO LONG

MEANING *lacctname* is more than eight characters.

ACTION Use EDITOR to change the *lacctname*.

MESSAGE LOCAL GROUP NAME IS TOO LONG

MEANING *lgroupname* has more than eight characters.

ACTION Use EDITOR to change the *lgroupname*.

MESSAGE LOCAL USER NAME TOO LONG

MEANING *lusername* has more than eight characters.

ACTION Use EDITOR to change the *lusername*.

MESSAGE LOG BUFFER OVERFLOW

MEANING Buffer space is insufficient to build the log record.

ACTION Notify the database administrator and HP support personnel.

MESSAGE LOG BUFFER OVERFLOW DURING GET NEXT LOGICAL LOG RECORD

MEANING Internal error.

ACTION Contact your database administrator.

MESSAGE LOG FILE AND DATABASE IDs DO NOT MATCH

MEANING Internal error. Database is not using the correct log file.

ACTION Contact your database administrator.

MESSAGE LOGFILE AND DATABASE LOGID'S DO NOT MATCH

MEANING The restored database does not match the log file.

ACTION Be sure the log file and the backup database are properly attached.

MESSAGE LOG FILE HEADER RECORD IS NOT FIRST RECORD

MEANING Internal error.

ACTION Contact your database administrator.

MESSAGE LOGFILE IS EMPTY

MEANING The log file has no records.

ACTION Be sure you have correctly identified the log file.

MESSAGE LOG FILE OPEN FAILED

MEANING Internal error.

ACTION Contact your database administrator.

MESSAGE LOG FILE RECORD SIZE IS IMPROPER

MEANING Internal error. The records are not 256 bytes with a fixed length.

ACTION Be sure this file is actually a log file. Contact your database administrator.

MESSAGE LOGID:*logid* IS INVALID

MEANING The database *logid* has been removed from the MPE table.

ACTION Set the database to an existing *logid*, or use the MPE GETLOG command to establish the database *logid* on the MPE table.

MESSAGE LOGID MUST NOT BE LONGER THAN 8 CHARACTERS

MEANING The *logid* name is too long.

ACTION Retype the correct *logid* to set into the database.

MESSAGE LOGID PASSWORD IS INCORRECT

MEANING The password has been altered, or you are using the wrong password.

ACTION Set the *logid* with the correct password into the database.

MESSAGE LOGGING IS ENABLED, CAN NOT CLEAR LOGID

MEANING The *logid* cannot be changed while logging is in process in order to ensure validity of recovery and logging.

ACTION Change the *logid* after the recovery process has completed.

MESSAGE LOG PERIOD DOES NOT EQUAL ONE (1)

MEANING Internal error.

ACTION Contact your database administrator.

MESSAGE MAINTENANCE WORD CANNOT BE CHANGED UNTIL RECOVERY HAS COMPLETED

MEANING Changing maintenance word was attempted when recovery was enabled.

ACTION After the recovery process has completed use the DBUTIL SET command to change the maintenance word.

MESSAGE MAINTENANCE WORD REQUIRED

MEANING The user invoking the utility is not the creator of the referenced database and has not supplied a maintenance word.

ACTION If DBUTIL, reenter the command with the maintenance word. Otherwise correct the error or press return to terminate the program.

MESSAGE MAINTENANCE WORD TOO LONG

MEANING The specified maintenance word has more than eight characters.

ACTION Reenter the command with the correct maintenance word.

MESSAGE MAXIMUM ERROR COUNT HAS BEEN EXCEEDED

MEANING DBRECOV keeps track of all the errors that were encountered. If the number goes past a certain threshold (default is 0, but can be configured up to a maximum of 3000), DBRECOV will abort.

ACTION Correct the problems causing the errors. If the user specifies an error count, this count could also be increased.

MESSAGE MAXIMUM WARNING COUNT HAS BEEN EXCEEDED

MEANING DBRECOV keeps track of all the warnings that were encountered. If the number goes past a certain threshold, DBRECOV will abort.

ACTION If the user specifies a warning count, this count could be increased.

MESSAGE MISSING!!

MEANING The keyword 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.

ACTION Use EDITOR to modify the record in the database-access file that returned the error message.

MESSAGE MISSING DBNAME1

MEANING An equal sign (=), semicolon (;), or end of line followed the keyword FILE.

ACTION Use EDITOR to change the FILE command.

MESSAGE MISSING DBNAME2

MEANING *dbname*1= was followed by a semicolon (;) or end of line.

ACTION Use EDITOR to change the FILE command.

MESSAGE MISSING DSDEVICE

MEANING DEV= was followed by a semicolon (;), #, or the rest of the record was blank.

ACTION Use EDITOR to change the FILE command.

MESSAGE MISSING LOCAL PART

MEANING No characters preceded the =HELLO command.

ACTION Use EDITOR to change the HELLO command.

MESSAGE MISSING REMOTE PART

MEANING The remote-ID-sequence of the database-access file is missing.

ACTION Use EDITOR to add the remote part.

MESSAGE MISSING REMOTE PASWD

MEANING HELLO was not followed by text, or was followed by a semicolon (;).

ACTION Using the EDITOR, change the HELLO command.

MESSAGE MISSING SEMI-COLON

MEANING DEV= was not preceded by a semicolon (;)

ACTION Use EDITOR to change the FILE command.

MESSAGE MISSING # SIGN

MEANING *dsdevice* was not followed by a #.

ACTION Use EDITOR to change the FILE command.

MESSAGE MORE THAN ONE (1) LOG FILE HEADER RECORD IS FOUND

MEANING Internal error.

ACTION Contact your database administrator.

MESSAGE MULTIPLE LOG IDENTIFIERS NOT ALLOWED

MEANING You tried to simultaneously recover databases that logged to different log files.

ACTION Run DBRECOV once for each log file.

MESSAGE MUSTRECOVER MUST BE DISABLED BEFORE LOGGING CAN BE DISABLED

MEANING Logging cannot be disabled while MUSTRECOVER is active.

ACTION Disable MUSTRECOVER first and then disable logging.

MESSAGE MUSTRECOVER & ROLLBACK MUST BE DISABLED BEFORE DISABLING LOGGING

MEANING Logging cannot be disabled if either MUSTRECOVER or roll-back recovery is active.

ACTION Disable both MUSTRECOVER and roll-back recovery before disabling Logging.

MESSAGE NEGATIVE CURRENT RECORD NUMBER IN BACKSPACE REQUEST

MEANING Internal error.

ACTION Contact your database administrator.

MESSAGE NEGATIVE NUMBER OF BACKSPACES REQUESTED

MEANING Internal error.

ACTION Contact your database administrator.

MESSAGE NLS RELATED ERROR

MEANING An error was returned by NLS/3000 on a DBOPEN on the database.

ACTION Notify the database administrator.

MESSAGE NLINFO FAILURE

MEANING An error was returned by NLS/3000.

ACTION Notify database administrator.

MESSAGE NO DATABASE HAS BEEN SPECIFIED FOR RECOVERY

MEANING RUN command has been entered and no databases have been specified for recovery.

ACTION Use the RECOVER command to specify database(s) or EXIT to terminate.

MESSAGE NO INDEXING PRODUCT CONFIGURED

MEANING This message is returned if you are *not* using third-party indexing (TPI) and you attempted to disable the database for indexing. The message is generated by the DBUTIL DISABLE *database name* FOR INDEXING command when the database is not configured for third-party indexing.

ACTION Disregard the message if your database is not configured for third- party indexing.

MESSAGE NO INDEXING PRODUCT CONFIGURED; CANNOT ENABLE INDEXING

MEANING This message is returned if you are using third-party indexing (TPI). The message is generated by the DBUTIL ENABLE *database name* FOR INDEXING command when the database is not yet configured for third-party indexing using the third-party utility. The database must be configured for indexing before it can be enabled for indexing.

ACTION Refer to your third-party vendor documentation for instructions on configuring the database for indexing. Once you have configured the database for indexing, retry enabling the database for indexing.

MESSAGE NON-CREATOR ACCESS NOT PERMITTED

MEANING You must be the database creator to perform this function, or you must use the maintenance word.

ACTION Log on as the database creator, or use the maintenance word with the command. If the maintenance word is not set, have the creator set one.

MESSAGE NONEXISTENT LOGID

MEANING You are trying to set an unrecognized *logid* into the database, or the database *logid* has been removed from MPE.

ACTION Use the MPE GETLOG command to put the database *logid* into the MPE table, or set an existing *logid* into the database.

MESSAGE NO SUCH DATABASE

MEANING Specified database does not exist in users logon group.

ACTION Check base name and logon account and group. Press return to terminate the program.

MESSAGE NOT A DATABASE ACCESS FILE

MEANING The specified file is not a database-access file.

ACTION Check the file name and try the command again.

MESSAGE NOT A DATABASE OR DATABASE ACCESS FILE

MEANING The specified database is invalid, or the specified file is not a database-access file.

ACTION Check the database or file name and try the command again.

MESSAGE NOT A DATABASE ROOT FILE

MEANING The specified file is not a database root file.

ACTION Check the specified database name and try the command again.

MESSAGE NOT A PRIVILEGED DATABASE ACCESS FILE

MEANING The database-access file has not been activated.

ACTION Check the command and reenter it correctly.

MESSAGE NOT ALLOWED; MUST BE CREATOR

MEANING The user invoking the utility is not the creator of the database and the database has no maintenance word.

ACTION Log on with correct user name, account and group.

MESSAGE NOT AN UNPRIVILEGED DATABASE ACCESS FILE

MEANING The database-access file is already activated.

ACTION Check the command and reenter it if necessary.

MESSAGE OUTMODED ROOT

MEANING The root file of the specified database corresponds to a different version of IMAGE software and is not compatible with the utility program currently executing.

ACTION Consult with your database administrator to verify the correct version of the software. Ask HP support personnel about conversion process.

MESSAGE PARAMETER EXPECTED

MEANING The command you specified calls for another parameter.

ACTION Use HELP to find the correct command syntax.

MESSAGE PARAMETER MUST BE A COMMAND

MEANING Only command names can be specified with the HELP command.

ACTION Reenter the command.

MESSAGE PARAMETER SPECIFIED TWICE

MEANING The command specified requires only one parameter entry.

ACTION Reenter the command.

MESSAGE PARSE OF FILE REFERENCE FAILED

MEANING Internal error.

ACTION Contact your database administrator.

MESSAGE PASSWORD IS INCORRECT

MEANING The *logid* password in the database is not the same as the password in the MPE table.

ACTION Set the *logid* and correct password into the database. Or, use the MPE ALTLOG command to alter the password in the MPE table so it matches the one in the database.

MESSAGE PASSWORD MUST NOT BE LONGER THAN 8 CHARACTERS

MEANING The password is too long so it is incorrect.

ACTION Retype the correct *logid* and password.

MESSAGE PREMATURE EOF ON ASCII ACCESS FILE

MEANING DBUTIL encountered an end of file mark before at least one =HELLO record in the database-access file.

ACTION Use EDITOR to change the ASCII file.

MESSAGE RECORD SIZE EXCEEDS 128 CHARACTERS!

MEANING The longest record in the database-access file exceeds the allowable length.

ACTION Use EDITOR to modify the database-access file.

MESSAGE RECORDS ARE NUMBERED!

MEANING The database-access file is numbered.

ACTION Use EDITOR to keep the file unnumbered.

MESSAGE RECORD TABLE OVERFLOW

MEANING More than 1000 detail records have been added and they have been given relative record numbers different from those originally assigned. Wrong database could have been restored, or not all modifications to the database were logged.

ACTION Run DBRECOV without the NOABORTS option or recover to a point before the occurrence of the error. Or restore the correct database. Or restore database and perform DBUNLOAD/DBLOAD.

MESSAGE RELEASE OF ENTRY FROM RECOVERY TABLE FAILED
MEANING Internal error.
ACTION Contact your database administrator.

MESSAGE REMOTE ACCOUNT *racctname* TOO LONG
MEANING *racctname* is more than eight characters long.
ACTION Use EDITOR to change the HELLO command.

MESSAGE REMOTE ACCT PASSWORD TOO LONG
MEANING *rupasw* is more than eight characters.
ACTION Use EDITOR to change the HELLO command.

MESSAGE REMOTE GROUP *rgroupname* TOO LONG
MEANING *rgroupname* is more than eight characters.
ACTION Use EDITOR to change the HELLO command.

MESSAGE REMOTE GROUP PASSWORD TOO LONG
MEANING *rgpasw* is more than eight characters.
ACTION Use EDITOR to change the HELLO command.

MESSAGE REMOTE USER *rusername* TOO LONG
MEANING *rusername* is more than eight characters.
ACTION Use EDITOR to change the HELLO command.

MESSAGE REMOTE USER PASSWORD TOO LONG
MEANING *rupasw* is more than eight characters.
ACTION Use EDITOR to change the HELLO command.

MESSAGE RESTART FILE OPEN FAILED
MEANING Internal error.
ACTION Contact your database administrator.

MESSAGE ROLLBACK MUST BE DISABLED BEFORE AUTODEFER CAN BE ENABLED

MEANING Deferred output option (AUTODEFER) cannot be used when the database has been enabled for ROLLBACK.

ACTION Disable ROLLBACK and enable AUTODEFER using DBUTIL. (Refer to chapters 7 and 8 for more information on AUTODEFER.)

MESSAGE ROLLBACK MUST BE DISABLED BEFORE LOGGING CAN BE DISABLED

MEANING An attempt was made to disable logging while rollback was still enabled.

ACTION Use the DBUTIL DISABLE command to disable the database for ROLLBACK, then logging.

MESSAGE ROOT FILE DOES NOT EXIST

MEANING A root file with the name of the specified database does not exist in the logon group and account.

ACTION Check the database name you specified and reenter the command.

MESSAGE SEQUENCE ERROR ON LOG RECORD #*n* - EOF ASSUMED

MEANING Log record numbers are out of sequence.

ACTION Notify the database administrator and HP support personnel. One likely cause is that a log file has been altered, or that a start recovery operation was not performed after a system failure.

MESSAGE TIMESTAMP OUT OF SEQUENCE ON LOG RECORD #*n* - EOF ASSUMED.

MEANING Log record timestamps are out of sequence.

ACTION Notify the database administrator and HP support personnel. One likely cause is that a log file has been altered, or that a start recovery operation was not performed after a system failure.

MESSAGE TOO MANY PARAMETERS

MEANING Too many parameters were entered with the command.

ACTION Reenter the command with the correct number of parameters.

MESSAGE UNABLE TO COPY GLOBAL AND TABLE INFORMATION TO RESTART FILE

MEANING Internal error.

ACTION Contact your database administrator.

MESSAGE UNABLE TO GET NEXT LOGICAL LOG RECORD

MEANING Internal error.

ACTION Contact your database administrator.

MESSAGE UNABLE TO GET PRIOR LOGICAL LOG RECORD

MEANING Internal error.

ACTION Contact your database administrator.

MESSAGE UNKNOWN COMMAND, TRY HELP

MEANING DBUTIL does not recognize the command you specified.

ACTION Enter the HELP command to get the list of DBUTIL commands.

MESSAGE USER IS NOT CREATOR OF LOGGING IDENTIFIER

MEANING A function or activity was attempted by someone other than the creator of the log identifier.

ACTION Log on to the system as the user who created the log identifier or as a user with database administrator (SM) or operator (OP) capability.

MESSAGE WARNING: DATABASE MODIFIED AND NOT DBSTORED

MEANING The database has been enabled for logging but a backup copy has not been made by executing DBSTORE.

ACTION Execute DBSTORE to make a backup copy of the database.

MESSAGE WARNING: MUSTRECOVER DISABLED BUT DATABASE NEEDS RECOVERY CANNOT GUARANTEE A CONSISTENT DATABASE

MEANING The system failed while the MUSTRECOVER option was enabled. The database needs to be recovered to ensure database consistency.

ACTION To ensure database consistency, respond with N at the DBUTIL prompt to discontinue the DISABLE command, then recover the database with DBRECOV. To allow access to the database without ensuring database consistency by recovering the database, respond with Y.

MESSAGE WARNING: MUSTRECOVER IS ENABLED AND DATABASE NEEDS RECOVERY

MEANING The database is enabled for MUSTRECOVER and the database needs to be recovered.

ACTION Use DBRECOV to recover the database.

MESSAGE WARNING: ROOT FILE ATC FLAG IS ON, BUT *dbname*TC FILE IS MISSING

MEANING This message is returned if you are using ALLBASE/Turbo CONNECT. The message is generated from the DBSTORE and the DBUTIL SHOW *database name* ALL commands. The ALLBASE/Turbo CONNECT (ATC) flag in the root file indicates that the database is attached to at least one ALLBASE/SQL database environment (DBEnvironment), but its corresponding ATC file does not exist. Therefore, DBSTORE will not store the TC file and DBUTIL will not show all the DBEnvironments the database is attached to.

ACTION Inform your database administrator. The ALLBASE/Turbo CONNECT flag and the file can be synchronized by using the IMAGE/SQL utility DETACH and ATTACH commands.

MESSAGE WARNING: ROOT FILE ATC FLAG IS ON, BUT *dbname*TC FILE IS MISSING, CANNOT DETACH

MEANING This message is returned if you are using ALLBASE/Turbo CONNECT (ATC). The message is generated from the DBUTIL PURGE command used with the DETACH option. The ATC flag in the root file indicates that the database is attached to at least one ALLBASE/SQL database environment (DBEnvironment), but its corresponding ATC file does not exist. Therefore, DBUTIL cannot detach the specified database. The PURGE command will continue processing after the warning is issued.

ACTION Inform your database administrator. The ALLBASE/Turbo CONNECT flag and the file can be synchronized by using the IMAGE/SQL utility DETACH and ATTACH commands.

MESSAGE WARNING: ROOT FILE ATC FLAG IS OFF, BUT *dbname*TC FILE EXISTS

MEANING This message is returned if you are using ALLBASE/Turbo CONNECT (ATC). The message is generated from the DBSTORE and the DBUTIL SHOW *database name* ALL commands. The ATC flag in the root file indicates that the database is not attached to an ALLBASE/SQL database environment (DBEnvironment), but its corresponding ATC file exists. Therefore, DBSTORE will not store the TC file and DBUTIL will not show all the DBEnvironments the database is attached to.

ACTION Inform your database administrator. The problem could be caused by restore activity in the group. The ATC flag and the file can be synchronized by using the IMAGE/SQL utility DETACH and ATTACH commands.

MESSAGE WARNING: THE LANGUAGE OF THE DATABASE IS DIFFERENT FROM THE LANGUAGE FOUND ON THE DBLOAD MEDIA. CONTINUE DBLOAD OPERATION / (Y/N):

MEANING The user has changed language of database between DBUNLOAD and DBLOAD. DBLOAD wants user to be aware of potential differences in sorted chains of the collating sequence of the two languages. (The language of database on disk and database on tape are different).

In session mode you are asked if you want to continue operation. In job mode, DBLOAD will terminate execution.

ACTION After looking at the information, DBLOAD returns with the result on the sorted chains in the database. Continue the operation by answering Y (yes).

MESSAGE 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.

MEANING You have added or deleted data sets from the database between a DBUNLOAD and a DBLOAD. DBLOAD will continue with the load, but data can be lost or put into the wrong sets due to an invalid set change.

ACTION Stop the DBLOAD, correct the schema, and perform DBLOAD again. Or, if you are sure the database is not corrupt, allow DBLOAD to continue.

Utility Unconditional Error Messages

MESSAGE AUTOMATIC MASTER IS FULL ON PATH *#n*

MEANING DBLOAD is unable to load a detail entry because the automatic master set associated with path *n* of the detail set is full.

ACTION Re-create the root file with a larger capacity for automatic master. Rerun the necessary utilities.

MESSAGE ***BAD DATABASE***

MEANING This message is issued by DBSTORE, DBRESTOR, and DBUNLOAD. It means the database is flagged "bad" because of a known structural error due to an abnormal termination or to a system failure during DBLOAD or some other "deferred output" 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 database on disk retains its "bad" flag and cannot be accessed through DBOPEN.

ACTION The database is not usable in its current state. Purge it and restore a backup copy, or erase it and then load a tape written by DBUNLOAD or another external copy of the data.

MESSAGE BROKEN FILE EQUATION CHAIN FOR TAPE FILE

MEANING Issued by DBSTORE if, in a chain of file equations, the actual device designator cannot be found.

ACTION Check your MPE FILE commands and reenter them correctly before running DBSTORE.

MESSAGE CANNOT GET EXTRA DATA SEGMENT NECESSARY FOR RESTORE OPERATION

MEANING DBRESTOR was unable to get the extra data segment it needed for the buffers used in the restore operation.

ACTION Wait until system resources are available and then try again.

MESSAGE CANNOT OPEN TERMINAL, TERMINATING

MEANING DBUTIL is unable to access the terminal.

ACTION Call HP systems engineer.

MESSAGE CAN'T CREATE NEW COPY OF FILE *x* (FS ERROR #)

MEANING MPE intrinsic FOPEN failed while creating a temporary file.

ACTION Check the file system error. Possible reasons: The specified device does not exist or has insufficient space.

MESSAGE CAN'T GET SPECIFICATIONS VIA FGETINFO FOR THIS FILE *x* (FS error #)

MEANING MPE intrinsic FGETINFO failed. DBUTIL cannot get information for the specified file.

ACTION Check the file system error to determine cause.

MESSAGE CAN'T OPEN FILE *x* (FS ERROR #)

MEANING MPE intrinsic FOPEN failed.

ACTION Check the file system error to determine cause.

MESSAGE CAN'T PURGE OLD COPY OF FILE *x* (FS ERROR #)

MEANING MPE intrinsic FCLOSE failed while purging the "old" file.

ACTION Check the file system error to determine cause.

MESSAGE CAN'T RESET MOVE FLAG FOR FILE *x* IN THE ROOT FILE

MEANING Error occurred while writing root file label to reset the database return status.

ACTION Contact your database administrator.

MESSAGE CAN'T SAVE NEW COPY OF FILE *x* (FS ERROR #)

MEANING MPE intrinsic FCLOSE failed while saving the temporary file.

ACTION Check file system error to determine the cause.

MESSAGE CAN'T SET MOVE FLAG FOR FILE *x* IN ROOT FILE

MEANING DBUTIL set the root file flag, and an error occurred while writing the root file label.

ACTION Contact your database administrator.

MESSAGE CHAIN IS FULL ON PATH *#n*

MEANING DBLOAD is unable to load a detail entry because the chain count for path number *n* of detail set exceeds 2^{31} –1 (or 2,147,483,647 entries).

ACTION Delete some of the entries and reload or change the database design.

MESSAGE ***COPY FAILED***

MEANING An error occurred while copying the file. DBUTIL has encountered a problem reading or writing the files. (It is possible the disk pack is bad.)

ACTION Restore the database and try the move again. Contact your database administrator if the copy fails again.

MESSAGE COULD NOT ATTACH DATA SET *n* TO XM LOG FILE: FILE SYSTEM ERROR *nn*

MEANING The data set could not be attached to the MPE transaction recovery mechanism. *nn* is the file system error number returned.

ACTION Notify HP support personnel.

MESSAGE COULD NOT ATTACH DATA SET *n* TO XM LOG FILE: XM ERROR *nn*

MEANING The data set *n* could not be attached to the MPE transaction recovery mechanism. *nn* is the error number returned.

ACTION Notify HP support personnel.

MESSAGE COULD NOT DETACH DATA SET *n* FROM XM LOG FILE: FILE SYSTEM ERROR *nn*

MEANING The data set *n* could not be detached from the MPE transaction recovery mechanism. The MPE intrinsic FLABELOPEN failed. *nn* is the file system error number returned.

ACTION Notify HP support personnel.

MESSAGE COULD NOT DETACH DATA SET *n* FROM XM LOG FILE: XM ERROR *nn*

MEANING The data set *n* could not be detached from the MPE transaction recovery mechanism. *nn* is the error number returned.

ACTION Notify HP support personnel.

MESSAGE COULDN'T OPEN THE DATABASE

MEANING No root file exists for the specified file. The file specified might not be part of the database.

ACTION Check the file or database. Try the operation again with the correct file name.

MESSAGE DATABASE STATE DOES NOT ALLOW MOVE TO BE DONE

MEANING The database is in a state in which a move cannot be performed.

ACTION Check the database. If the database has not been created, use the DBUTIL CREATE command to create it. If the database is damaged, run recovery.

MESSAGE DATA SET FULL

MEANING Data set currently being loaded is full.

ACTION Re-create root file, increase the data set's capacity and run the utilities again.

MESSAGE DATABASE TABLE OVERFLOWED

MEANING Internal error.

ACTION Contact your database administrator.

MESSAGE DATABASE TABLE TRANSACTION COUNT IS NEGATIVE

MEANING Internal error.

ACTION Contact your database administrator.

MESSAGE DBBEGIN FOUND AN ALREADY ACTIVE TRANSACTION

MEANING Internal error.

ACTION Contact your database administrator.

MESSAGE DBBEGIN FOUND PROCESS ALIVE, BUT NOT ACTIVE

MEANING Internal error.

ACTION Contact your database administrator.

MESSAGE DBEND FOUND TRANSACTION STILL ALIVE AND ACTIVE

MEANING Internal error.

ACTION Contact your database administrator.

MESSAGE DBEND FOUND TRANSACTION WHICH ALREADY ENDED

MEANING Internal error.

ACTION Contact your database administrator.

MESSAGE ***DBSTORE FAILED - NO DATABASE STORED***

MEANING A file error or other message follows explaining the problem.

ACTION Contact your database administrator.

MESSAGE Dynamic capacity expansion in progress flag is on

MEANING The data set is being expanded by DBPUT in another process. The number of entries and the current capacity may not be accurate since the data set is being expanded at this time. If asterisks (***) are in the "no. of entries" and "%max cap" fields, then a previous DBPUT may have aborted while DBPUT was expanding this data set.

ACTION If there are positive numbers in the "no. of entries" and "% max cap" fields, nothing needs to be done. This message is for information only. If the asterisks are in these two fields, then a DBOPEN must be performed to correct the capacity information for the data set. DBOPEN can be done via QUERY/3000 or any application. This DBOPEN should be the *first* open of the database in order to correctly adjust the data set free count and the root file data set capacity fields using the actual data set file size. (See chapter 3 for more information on capacity expansion and chapter 5 for DBOPEN.)

MESSAGE EOF SEEN, PROGRAM TERMINATING

MEANING An end-of-file (EOF) has been entered.

ACTION Run DBUTIL again, and do not type :EOF interactively. Or correct job stream which contains error.

MESSAGE FGETINFO FAILURE

MEANING DBUTIL received an error when calling MPE FGETINFO.

ACTION Contact your database administrator.

MESSAGE FILE EQUATE FOR

DBSTORE

DBRESTOR

DBLOAD

DBUNLOAD

ONLY MAY USE DEV

MEANING If you specify an input/output file with an MPE FILE command for any of these utility programs, only the file designators and DEV= parameters are allowed.

ACTION Enter the MPE FILE command again and rerun program.

MESSAGE FILE NOT ON TAPE

MEANING Issued by DBRESTOR if the database to be restored is not on the tape.

ACTION Check your MPE FILE command and/or the tape you mounted.

MESSAGE FILE *x* ALREADY RESIDES ON DEVICE *x*

MEANING The file specified already resides on the device.

ACTION Continue with the next desired command, or exit from DBUTIL.

MESSAGE FREADDIR FAILURE

MEANING This is an MPE exceptional failure and is returned when DBUTIL calls FREADDIR.

ACTION Contact your database administrator.

MESSAGE HARD TERMINAL READ ERROR, TERMINATING

MEANING DBUTIL cannot read input from terminal.

ACTION Notify HP systems engineer.

MESSAGE INTERNAL ERROR--CURRENT TRANSACTION COUNT IS NEGATIVE

MEANING Internal error.

ACTION Contact your database administrator.

MESSAGE INTERNAL ERROR--DATABASE SHOULD NOT BE OPEN AT THIS TIME

MEANING Internal error.

ACTION Contact your database administrator.

MESSAGE INTERNAL ERROR--DBBEGIN FOUND PROCESS ALIVE, BUT TRAN STATE INACTIVE

MEANING Internal error.

ACTION Contact your database administrator.

MESSAGE INTERNAL ERROR--EXPECTED DATABASE TO BE OPEN FOR THIS PROCESS

MEANING Internal error.

ACTION Contact your database administrator.

MESSAGE INTERNAL ERROR--FCLOSE INTRINSIC FAILED

MEANING Internal error.

ACTION Contact your database administrator.

MESSAGE INTERNAL ERROR--FOPEN FAILS IN F'HEADLOG

MEANING Internal error.

ACTION Contact your database administrator.

MESSAGE INTERNAL ERROR--FOPEN INTRINSIC FAILED

MEANING Internal error.

ACTION Contact your database administrator.

MESSAGE INTERNAL ERROR--FREADLABEL FAILS IN F'HEADLOG

MEANING Internal error.

ACTION Contact your database administrator.

MESSAGE INTERNAL ERROR--FREADLABEL INTRINSIC FAILED

MEANING Internal error.

ACTION Contact your database administrator.

MESSAGE INTERNAL ERROR--FWRITELABEL INTRINSIC FAILED

MEANING Internal error.

ACTION Contact your database administrator.

MESSAGE INTERNAL ERROR--HEAP STAGING AREA OVERFLOWED

MEANING Internal error.

ACTION Contact your database administrator.

MESSAGE INTERNAL ERROR--INVALID DATABASE TABLE ADDRESS

MEANING Internal error.

ACTION Contact your database administrator.

MESSAGE INTERNAL ERROR--INVALID DBTAB'PROC'TAIL

MEANING Internal error.

ACTION Contact your database administrator.

MESSAGE INTERNAL ERROR--INVALID PASSWORD FOR DATABASE LOGGING IDENTIFIER

MEANING Log password does not correspond with the log identifier in the log file.

ACTION Contact your database administrator.

MESSAGE INTERNAL ERROR--INVALID PROCTABLE INDEX FOUND IN PROC'ACTIVELINK

MEANING Internal error.

ACTION Contact your database administrator.

MESSAGE INTERNAL ERROR--INVALID RECOVERY TABLE ADDRESS

MEANING Internal error.

ACTION Contact your database administrator.

MESSAGE INTERNAL ERROR--LOGID'TO'FNAME FAILURE

MEANING Internal error.

ACTION Contact your database administrator.

MESSAGE INTERNAL ERROR--NO PROCESS TABLE ENTRY EXISTS
MEANING Internal error.
ACTION Contact your database administrator.

MESSAGE INTERNAL ERROR--PROBLEM RETURNING HEAP
MEANING Internal error.
ACTION Contact your database administrator.

MESSAGE INTERNAL ERROR--PROCESS NOT IN PROCESS TABLE
MEANING Internal error.
ACTION Contact your database administrator.

MESSAGE INTERNAL ERROR--PROCESS TABLE IS FULL
MEANING Internal error.
ACTION Contact your database administrator.

MESSAGE INTERNAL ERROR--RECORD TABLE IS FULL
MEANING Internal error.
ACTION Contact your database administrator.

MESSAGE INTERNAL ERROR--UNABLE TO ALLOCATE THE HEAP
MEANING Internal error.
ACTION Contact your database administrator.

MESSAGE ***INVALID SET COUNT***
MEANING TurboIMAGE/XL found an inconsistency in the data set count.
ACTION Notify HP systems engineer.

MESSAGE INVALID TRANSACTION SEQUENCE DETECTED
MEANING Internal error. The transaction numbers are not consecutive.
ACTION Notify your database administrator and HP support personnel.

MESSAGE MOVE OF FILE *x* NOT ALLOWED: FILE IS NOT CORRECT TYPE (FILE CODE #)

MEANING The file specified is not a database file.

ACTION Try the MOVE operation again with the correct file name.

MESSAGE NO MANUAL ENTRY FOR DETAIL ON PATH #*n*

MEANING DBLOAD is attempting to load detail data set entry *n*, which is the number of the detail data set path referencing the manual master in question.

ACTION Add entry to manual master with application program or QUERY. Run DBLOAD again.

MESSAGE PROCESS TABLE OVERFLOW

MEANING At some point more than 180 processes were logging at the same time.

ACTION Notify your database administrator and HP support personnel.

MESSAGE RECOVERY TABLE IS FULL

MEANING Internal error.

ACTION Contact your database administrator.

MESSAGE RECOVERY TABLE OVERFLOW

MEANING You tried to specify more than 100 user recovery files.

ACTION Reduce the number of recovery files needed.

MESSAGE RECSIZE MUST BE <=*xxxx* AND EVENLY DIVISIBLE BY 256.

MEANING Issued if the REC=*recsize* parameter for DBSTORE specified a record size not modulo 256 halfwords, or greater than the configured record size of the device.

ACTION Check the REC=*recsize* parameter and correct it before running DBSTORE again.

MESSAGE ROLLBACK LOG FILE MUST BE IN THE SAME VOLUME SET AS THE DATABASE

MEANING MPE transaction recovery requires that the MPE user log file must reside on the same volume set as the database.

ACTION Build the MPE user log file in the same volume set as the database. **NOTE**: Because all files in a group are in the same volume set, the MPE command LISTGROUP indicates the volume set where the database resides.

MESSAGE STAGING FILE OVERFLOW

MEANING The staging file is not large enough. This message is generated by the DBRECOV utility.

ACTION Use a file equation to increase the TEMPLOG file size. For example:

```
FILE TEMPLOG;DISC=7000000
```

If you still get this error after increasing the file size, consult with your database administrator and HP support personnel.

MESSAGE SWITCH TO NM TO ATTACH DATABASE TO XM LOG FILE FAILED: XM ERROR *nn*

MEANING It is necessary to switch to NM to attach the database to the MPE recovery system. The switch failed. *nn* is the switch to NM error number returned.

ACTION Notify HP support personnel.

MESSAGE SWITCH TO NM TO DETACH DATABASE FROM XM LOG FAILED: SWITCH ERROR *nn*

MEANING It is necessary to switch to NM to detach the database from the MPE transaction recovery mechanism. The switch failed. *nn* is the switch to NM error number returned.

ACTION Notify HP support personnel.

MESSAGE UNABLE TO CONTINUE

MEANING 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.

ACTION Save the file information. Consult with database administrator and HP support personnel.

MESSAGE UNABLE TO OBTAIN FILE LABEL INFORMATION FOR FILE *x*

MEANING MPE internal procedure failed. DBUTIL cannot obtain information for the specified file.

ACTION Contact your database administrator.

MESSAGE UNEXPECTED ROOT FILE STATE

MEANING The MOVE process was unable to reset the database flag. The database could be in an inconsistent state.

ACTION Contact your database administrator.

MESSAGE WARNING: USER LOG FILE IS NOT IN THE SAME VOLUME SET AS THE DATABASE

MEANING MPE transaction recovery requires that the MPE user log file must reside on the same volume set as the database.

ACTION Build the MPE user log file in the same volume set as the database.

NOTE: Because all files in a group are in the same volume set, the MPE command LISTGROUP indicates the volume set where the database resides.

Extended Utility Program Unconditional Messages

A set of messages can be returned when an unusual condition causes TurboIMAGE/XL to fail while a TurboIMAGE/XL utility is executing. In each of the following messages, the value *xxxxx* is the location where TurboIMAGE/XL failed.

Error messages 401 through 414 are all TurboIMAGE/XL utility failures caused by failure of the TurboIMAGE/XL intrinsic specified in the message.

-3 MESSAGE TURBOIMAGE FAILURE (FREADDIR FAILURE ON ROOT FILE AT *xxxxx*)

MEANING Root file was not readable. Problem could be in hardware.

ACTION Notify the database administrator of the error.

-4 MESSAGE TURBOIMAGE FAILURE (FREADLABEL FAILURE ON ROOT FILE AT *xxxxx*)

MEANING User's label was not readable. Problem could be in hardware.

ACTION Notify the database administrator of the error.

-64	MESSAGE	SET NUMBERS ARE OUT OF SEQUENCE. TURBOIMAGE FAILED AT *xxxxx*)
	MEANING	The next set number on the backup volume was not what DBLOAD expected. Status element 0 has the expected set number; status element 1 has the next set number on the backup volume.
	ACTION	An error for a known value that was written to tape with FWRITE was found. The tape or tape drive could be defective.
-66	MESSAGE	BLOCK NUMBERS ARE OUT OF SEQUENCE. TURBOIMAGE FAILED AT *xxxxx*)
	MEANING	The next block on the backup volume was not what DBLOAD expected. Status elements 0 and 1 have the expected block number; elements 2 and 3 have the actual next block number; element 4 has the set number.
	ACTION	An error for a known value that was written to tape with FWRITE was found. The tape or tape drive could be defective.
-70	MESSAGE	ENTRY NUMBERS ARE OUT OF SEQUENCE.TURBOIMAGE FAILED AT *xxxxx*)
	MEANING	The next block on the backup volume was not what DBLOAD expected. Status elements 0,1 have the expected block number; elements 2,3 have the actual next block number; element 4 has the set number.
	ACTION	An error for a known value that was written to tape with FWRITE was found. The tape or tape drive could be defective.

-74	MESSAGE	THE RECORD JUST READ IS UNRECOGNIZABLE TURBOIMAGE FAILED AT *xxxxx*)
	MEANING	The record just read from the backup volume was not what DBLOAD expected. The record could be an EOF, EOT, etc. The status element 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
	ACTION	An error for a known value that was written to tape with FWRITE was found. The tape or tape drive could be defective.
97	MESSAGE	SIZE ERROR WHILE CONTRACTING AT *xxxxx*)
	MEANING	A TurboIMAGE utility called ZSIZE to contract the Z to DB area of the stack. *xxxxx* is the octal location where TurboIMAGE failed.
	ACTION	Notify HP support personnel.
98	MESSAGE	SIZE ERROR WHILE EXPANDING AT *xxxxx*)
	MEANING	A TurboIMAGE utility called ZSIZE to expand the Z to DB area of the stack. *xxxxx* is the octal location where TurboIMAGE failed.
	ACTION	Notify HP support personnel.

101	MESSAGE	DBRESTOR FAILURE IN DBRESTOR AT *xxxxx*)
	MEANING	The DBRESTOR utility called the MPE RESTORE command, which encountered problems while restoring the database. A detailed error message should follow this message.
	ACTION	Take appropriate action depending on the detail message.
401	MESSAGE	DBOPEN FAILURE AT *xxxxx*)
	MEANING	A detailed error message can be retrieved by calling DBEXPLAIN.
	ACTION	For more information, see "Library Procedure Error Messages" earlier in this appendix.
402	MESSAGE	DBINFO FAILURE AT *xxxxx*)
	MEANING	A detailed error message can be retrieved by calling DBEXPLAIN.
	ACTION	For more information, see "Library Procedure Error Messages" earlier in this appendix.
403	MESSAGE	DBCLOSE FAILURE AT *xxxxx*)
	MEANING	A detailed error message can be retrieved by calling DBEXPLAIN.
	ACTION	For more information, see "Library Procedure Error Messages" earlier in this appendix.
404	MESSAGE	DBFIND FAILURE AT *xxxxx*)
	MEANING	A detailed error message can be retrieved by calling DBEXPLAIN.
	ACTION	For more information, see "Library Procedure Error Messages" earlier in this appendix.
405	MESSAGE	DBGET FAILURE AT *xxxxx*)
	MEANING	A detailed error message can be retrieved by calling DBEXPLAIN.
	ACTION	For more information, see "Library Procedure Error Messages" earlier in this appendix.

406	MESSAGE	DBUPDATE FAILURE AT *xxxxx*)
	MEANING	A detailed error message can be retrieved by calling DBEXPLAIN.
	ACTION	For more information, see "Library Procedure Error Messages" earlier in this appendix.
407	MESSAGE	DBPUT FAILURE AT *xxxxx*)
	MEANING	A detailed error message can be retrieved by calling DBEXPLAIN.
	ACTION	For more information, see "Library Procedure Error Messages" earlier in this appendix.
408	MESSAGE	DBDELETE FAILURE AT *xxxxx*)
	MEANING	A detailed error message can be retrieved by calling DBEXPLAIN.
	ACTION	For more information, see "Library Procedure Error Messages" earlier in this appendix.
409	MESSAGE	DBLOCK FAILURE AT *xxxxx*)
	MEANING	A detailed error message can be retrieved by calling DBEXPLAIN.
	ACTION	For more information, see "Library Procedure Error Messages" earlier in this appendix.
410	MESSAGE	DBUNLOCK FAILURE AT *xxxxx*)
	MEANING	A detailed error message can be retrieved by calling DBEXPLAIN.
	ACTION	For more information, see "Library Procedure Error Messages" earlier in this appendix.
411	MESSAGE	TDBCONTROL FAILURE AT *xxxxx*)
	MEANING	A detailed error message can be retrieved by calling DBEXPLAIN.
	ACTION	For more information, see "Library Procedure Error Messages" earlier in this appendix.

412	MESSAGE	DBBEGIN FAILURE AT *xxxxx*)
	MEANING	A detailed error message can be retrieved by calling DBEXPLAIN.
	ACTION	For more information, see "Library Procedure Error Messages" earlier in this appendix.
413	MESSAGE	DBEND FAILURE AT *xxxxx*)
	MEANING	A detailed error message can be retrieved by calling DBEXPLAIN.
	ACTION	For more information, see "Library Procedure Error Messages" earlier in this appendix.
414	MESSAGE	DBMEMO FAILURE AT *xxxxx*)
	MEANING	A detail error message can be retrieved by calling DBEXPLAIN.
	ACTION	For more information, see "Library Procedure Error Messages" earlier in this appendix.

B Results of Multiple Access

When opening a database with DBOPEN, TurboIMAGE/XL returns information in the status array describing the results of the procedure call. Figure B-1., "Actions Resulting from Multiple Access of Databases." can be used to interpret these results when multiple processes are using the database.

Each box in Figure B-1. is associated with a requested mode or TurboIMAGE/XL 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 element 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 database activity, the results are like those shown in the other boxes. Two types of situations can occur:

- If the first two elements of the status array contain –1 and 0, respectively, the third element of status will contain a single number. This number is the MPE/iX failure code returned from the FCHECK intrinsic. See the *MPE/iX Intrinsic Reference Manual* for MPE/iX failure code meanings. If that number is 48, 90, or 91, the failure occurred because current access to the database 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 element. The possible modes and utility routines which other processes could be using are the ones which label the columns containing these boxes. For example, if the third status element 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 database with that mode. In the example above, alternate modes would be 1 or 5 because these rows contain G in columns 1 and 5.

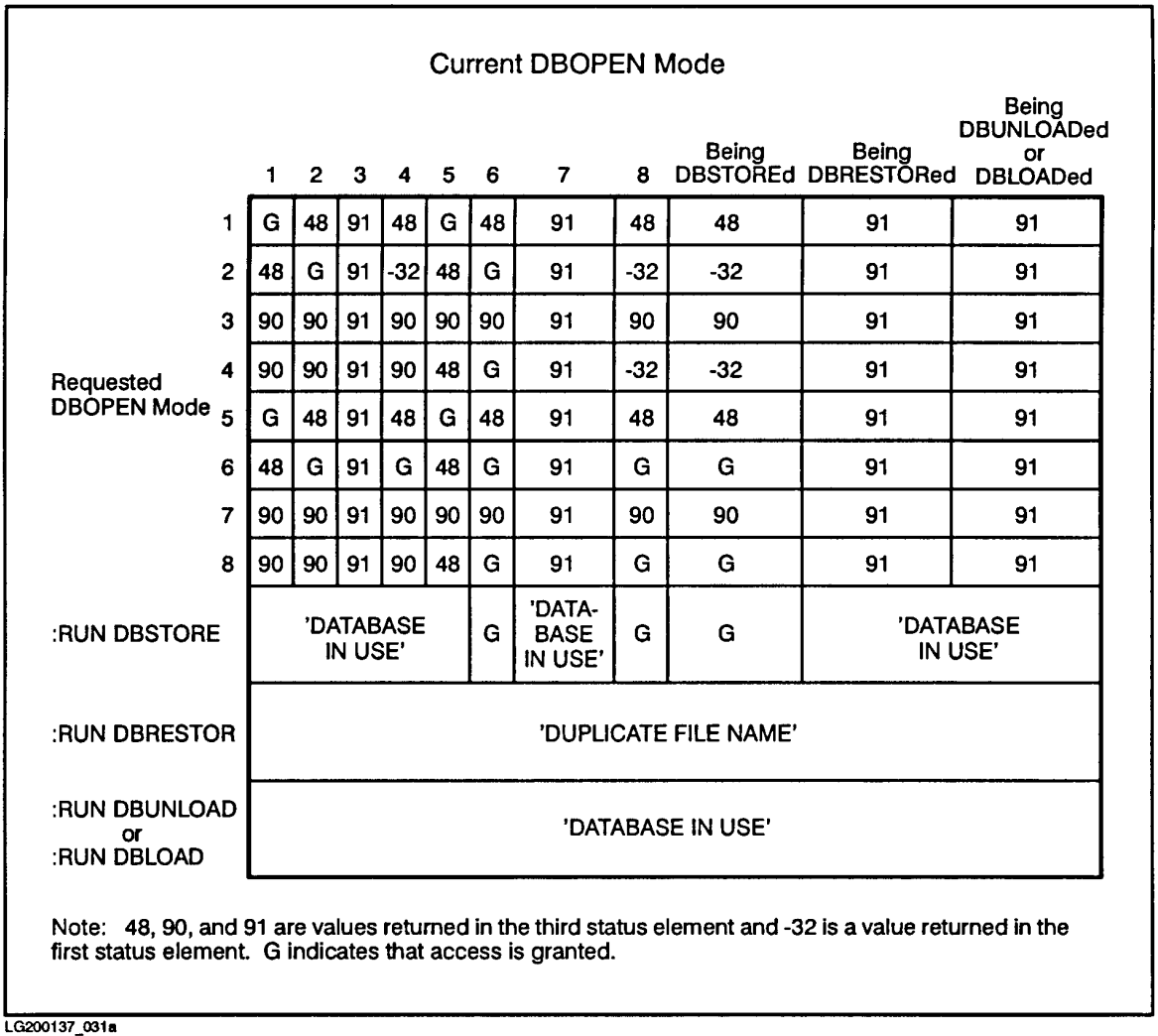
  If the box with contents matching the third status element is in a column associated with a utility, usually the only choice is to wait until execution terminates. When DBSTORE is being run, it is possible to open the database with mode 6 or 8.

- If the first element 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/iX 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 element of status equals –32, possible current modes are 4 and 8, and the DBSTORE utility could be executing.

The messages enclosed in quotes are printed when the situation represented by the row

and column headings occurs.

Figure B-1. Actions Resulting from Multiple Access of Databases



Note: 48, 90, and 91 are values returned in the third status element and -32 is a value returned in the first status element. G indicates that access is granted.

LG200137_031a

C Database Design Considerations

- Keep one-of-a-kind information, such as unique identifiers, in master data sets. Keep duplicate information, such as records of events (sales, purchases, shipments), in detail data sets.
- Define a search item in a detail data set if you want to retrieve all entries with a common value for that data item.
- 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.
- Limit the use of sort items to paths with relatively short chains in order to reduce the time required to add and delete entries.
- Select the path most frequently accessed in chained order as the primary path.
- Remember that data items must be an integral number of halfwords in length.
- When selecting the maximum block size, consider the environment in which the database will be used. (Refer to the `$CONTROL` command in chapter 3 for more information.)
- If you intend to use QUERY/3000 with your database, refer to the *QUERY/V Reference Manual* for the data types that QUERY/3000 supports.
- 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 chapter 4 to decide on appropriate access modes to use for your application programs.
- Analyze the time required to maintain the database, for example, the time required to unload and load the database. A database restructuring tool can help keep the time spent on maintenance to a minimum.
- The capacity of each data set should be defined as realistically as possible because a capacity that is too large wastes disk space. Data set can be expanded dynamically if capacity expansion parameters are set. Otherwise, the capacity can be increased when necessary by restructuring the database as described in chapter 8.
- A master data set capacity equal to a prime number or to the product of two or three primes can yield fewer synonyms than a master data set capacity of many prime factors. See Figure C-1. for a partial list of prime numbers.
- The account and group in which the database resides must have enough file space available to contain all the database files.
- If your application uses sorted paths, plan to add or delete entries (`DBPUT, DBDELETE`) to sorted chains when the system is not very busy. If it is very busy, limit the database

activity on sorted chains to reading and updating (DBUPDATE).

- Do all or most of your locking at one level (database, data set, or data entry).
- 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.
- Avoid holding locks around a terminal read.
- Avoid beginning and ending logical transactions around a terminal read.
- All databases involved in a multiple database transaction must reside on the same system.
- If transaction logging is used, the database administrator must ensure that *all* databases involved in a multiple database transaction adhere to the following criteria:

  — Are enabled for logging.

  — Log to the same log file.

  — Are either enabled or disabled for roll-back recovery.

  — Are either enabled or disabled for MUSTRECOVER.
- For remote database access (applications and databases are not on the same system), the application's node, the database node, and all intermediate nodes where a multiple database transaction is routed must be on TurboIMAGE/XL version C.04.00 or later.

Figure 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	980,017	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	

D Multiple Calls to DBLOCK

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 can arise if you prepare your application programs with MR capability (CAP=MR).

Some typical situations in which CAP=MR could be required are the following:

- A program has two or more databases open and wishes to lock part or all of each database simultaneously. (One or more of the databases may be on a remote HP 3000.)
- A program wishes to lock an MPE file and a database simultaneously.
- A program wishes to lock data entries in a database and, after reading their contents, to apply further locks. This is very dangerous and is not recommended, because deadlocks can occur very easily.

The danger in all cases is that a deadlock could 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.

To avoid restarting the operating system, use DBCONTROL mode 7 to activate deadlock detection for all open databases. If a deadlock occurs, an error 26 is returned to the process instead of causing a system hang. The process can subsequently call DBUNLOCK to release all locks on the database placed by the access path.

TurboIMAGE/XL 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 of the following lockable entities:

- Databases, data sets, and data entries
- Remote databases, 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 database, extreme caution should be exercised because the deadlock situations can be very subtle. For example, if a process locks a data set and then attempts to lock the database, the process will wait for itself forever.

If it is absolutely necessary to make multiple DBLOCK calls, the following information about how TurboIMAGE/XL performs locking may be useful.

Sort Sequence for Lock Descriptors

TurboIMAGE/XL 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 relative operator (relop) field contains <=, it collates before any other lock descriptors for the data set because it has the lowest possible lower bound for its value. For example, a lock descriptor of SALES:QUANTITY <= 10 collates before a lock descriptor of SALES:QUANTITY = 5, because the lower bound of the former is the lowest possible integer for an I-type data item.

Conditional Locks

During a DBLOCK, if TurboIMAGE/XL 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/XL indicates how many locks have been applied when it returns (status element 1) to the calling process. It does not release the successful locks even though all the requested locks have not been applied. Because TurboIMAGE/XL 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 element 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.

Remote Databases

Locking remote database entities is the same as locking database entities with the following exception. If the local system has a user-created process structure, and each process is locking a remote database independently, the programs must have Multiple RIN capability because 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 databases, 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 being suspended permanently which requires a remote system restart to recover.

CAUTION Hewlett-Packard does not accept responsibility for system lockouts and deadlocks when Multiple RIN capability is in use with TurboIMAGE/XL.

The following DBUTIL command can be useful in tracing deadlocks that occur when CAP=MR is used.

```
>>SHOW database name LOCKS
```

E TurboIMAGE/XL Log Record Formats

This appendix shows the TurboIMAGE/XL log record formats for the DBBEGIN, DBCLOSE, DBDELETE, DBEND, DBMEMO, DBOPEN, DBPUT, DBUPDATE, DBXBEGIN, DBXEND, and DBXUNDO intrinsics. Note that the recovery flag will always be zero in the log file records. This flag is used during recovery if user recovery files are created.

NOTE All TurboIMAGE/XL records are contained within MPE/iX "WRITELOG" records. Consequently, all information contained in the header portion of each WRITELOG record is available, in addition to the information provided by TurboIMAGE/XL.

DBBEGIN

```
HALFWORD(0-8) -   MPE WRITELOG RECORD
HALFWORD(9)   -   TURBOIMAGE/XL LOG RECORD LENGTH
HALFWORD(10)  -   DBBEGIN LOG RECORD CODE ("BE")
HALFWORD(11)  -   DATA SEGMENT NUMBER
HALFWORD(12)  -   RECOVERY FLAG ("NO"-FAILED,"OK"-RECOVERED)
                  OR MULTIPLE DATABASE TRANSACTION SEQUENCE ID
HALFWORD(13)  -   TRANSACTION NUMBER (1 WORD)
HALFWORD(15)  -   LENGTH OF USER BUFFER
HALFWORD(16)  -   START OF USER BUFFER
```

DBBEGIN Mode 4 (MDBXBEGIN)

```
HALFWORD(0-8)   - MPE WRITELOG RECORD
HALFWORD(9)     - TURBOIMAGE/XL LOG RECORD LENGTH
HALFWORD(10)    - MDBXBEGIN LOG RECORD CODE ("TB")
HALFWORD(11)    - NOT USED
HALFWORD(12)    - SEQUENCE ID
                      BITS 0-3   = RESERVED
                      BITS 4-9   = SEQUENCE NUMBER
                      BITS 10-15 = TOTAL NUMBER OF DBBEGIN/DBEND
                                   LOG RECORDS IN THE SEQUENCE
HALFWORD(13-14) - TRANSACTION ID
HALFWORD(15)    - LENGTH OF USER BUFFER
HALFWORD(16)    - START OF USER BUFFER
HALFWORD(17)    - NUMBER OF BASE IDs INVOLVED IN MDBX
HALFWORD(18)    - DATA SEGMENT NUMBER OF FIRST BASE ID
HALFWORD(19-20) - TRANSACTION NUMBER FOR 1ST BASE ID
HALFWORD(21)    - DATA SEGMENT NUMBER OF SECOND BASE ID
HALFWORD(22-23) - TRANSACTION NUMBER FOR 2ND BASE ID
```

DBCLOSE

```
HALFWORD(0-8) -   MPE WRITELOG RECORD
HALFWORD(9)   -   TURBOIMAGE/XL LOG RECORD LENGTH
HALFWORD(10)  -   DBCLOSE LOG RECORD CODE ("CL")
HALFWORD(11)  -   BASE LOGGING ID
                       BIT 0 TO 5  = USER DBOPEN COUNT
                       BIT 6 TO 15 = INDEX TO THE DBS FOR THE DBG
HALFWORD(12)  -   USER PROCESS ABORT INDICATOR
HALFWORD(13)  -   RESERVED FOR DBRECOV RUN TIME USE
HALFWORD(14)  -   RESERVED FOR DBRECOV RUN TIME USE
```

DBDELETE

```
HALFWORD(0-8) -   MPE WRITELOG RECORD
HALFWORD(9)   -   TURBOIMAGE/XL LOG RECORD LENGTH
HALFWORD(10)  -   DBDELETE LOG RECORD CODE ("DE")
HALFWORD(11)  -   BASE LOGGING ID
                       BIT 0 TO 5  = USER DBOPEN COUNT
                       BIT 6 TO 15 = INDEX TO THE DBS FOR THE DBG
HALFWORD(12)  -   RECOVERY FLAG ("NO"-FAILED,"OK"-RECOVERED)
HALFWORD(13)  -   TRANSACTION NUMBER (1 WORD)
HALFWORD(15)  -   DATA SET NUMBER
HALFWORD(16)  -   DATA SET TYPE ("MA"-MASTER,"DE"-DETAIL)
HALFWORD(17)  -   RECORD NUMBER (1 WORD)
HALFWORD(19)  -   MODE PARAMETER
HALFWORD(20)  -   OFFSET TO KEY ITEM VALUE (IF MASTER TYPE)
HALFWORD(21)  -   OFFSET TO DELETED DATA
HALFWORD(22)  -   START OF KEY AND DATA BUFFER
```

DBEND

```
HALFWORD(0-8) -   MPE WRITELOG RECORD
HALFWORD(9)   -   TURBOIMAGE/XL LOG RECORD LENGTH
HALFWORD(10)  -   DBEND LOG RECORD CODE ("EN"), OR
                  ("AE") IF ABORTED
HALFWORD(11)  -   BASE LOGGING ID
                       BIT 0 TO 5  = USER DBOPEN COUNT
                       BIT 6 TO 15 = INDEX TO THE DBS FOR THE DBG
HALFWORD(12)  -   RECOVERY FLAG ("NO"-FAILED,"OK"-RECOVERED)
                  OR MULTIPLE DATABASE TRANSACTION SEQUENCE ID
HALFWORD(13)  -   TRANSACTION NUMBER (1 WORD)
HALFWORD(15)  -   LENGTH OF USER BUFFER
HALFWORD(16)  -   START OF USER BUFFER
```

DBEND Mode 4 (MDBXEND)

```
HALFWORD(0-8)   - MPE WRITELOG RECORD
HALFWORD(9)     - TURBOIMAGE/XL LOG RECORD LENGTH
HALFWORD(10)    - MDBXEND LOG RECORD CODE ("TE")
HALFWORD(11)    - NOT USED
HALFWORD(12)    - SEQUENCE ID
                      BITS 0-3   = RESERVED
                      BITS 4-9   = SEQUENCE NUMBER
                      BITS 10-15 = TOTAL NUMBER OF DBBEGIN/DBEND
                                   LOG RECORDS IN THE SEQUENCE
HALFWORD(13-14) - TRANSACTION ID
HALFWORD(15)    - LENGTH OF USER BUFFER
HALFWORD(16)    - START OF USER BUFFER (pointer to beginning of user text)
HALFWORD(17)    - NUMBER OF BASE IDs INVOLVED IN MDBX
HALFWORD(18)    - DATA SEGMENT NUMBER OF FIRST BASE ID
HALFWORD(19-20) - TRANSACTION NUMBER FOR 1ST BASE ID
HALFWORD(21)    - DATA SEGMENT NUMBER OF SECOND BASE ID
HALFWORD(22-23) - TRANSACTION NUMBER FOR 2ND BASE ID
```

DBMEMO

```
HALFWORD(0-8) -   MPE WRITELOG RECORD
HALFWORD(9)   -   TURBOIMAGE/XL LOG RECORD LENGTH
HALFWORD(10)  -   DBMEMO LOG RECORD CODE ("ME")
HALFWORD(11)  -   BASE LOGGING ID
                       BIT 0 TO 5  = USER DBOPEN COUNT
                       BIT 6 TO 15 = INDEX TO THE DBS FOR THE DBG
HALFWORD(12)  -   RECOVERY FLAG ("NO"-FAILED,"OK"-RECOVERED)
HALFWORD(13)  -   TRANSACTION NUMBER (1 WORD)
HALFWORD(15)  -   LENGTH OF USER BUFFER
HALFWORD(16)  -   START OF USER BUFFER
```

DBOPEN

```
HALFWORD(0-8) -   MPE WRITELOG RECORD
HALFWORD(9)   -   TURBOIMAGE/XL LOG RECORD LENGTH
HALFWORD(10)  -   DBOPEN LOG RECORD CODE ("OP")
HALFWORD(11)  -   BASE LOGGING ID
                       BIT 0 TO 5  = USER DBOPEN COUNT
                       BIT 6 TO 15 = INDEX TO THE DBS FOR THE DBG
HALFWORD(12)  -   USER NAME
HALFWORD(16)  -   USER GROUP
HALFWORD(20)  -   USER ACCOUNT
HALFWORD(24)  -   USER IDENTIFIER
HALFWORD(28)  -   DATABASE NAME
HALFWORD(31)  -   DATABASE GROUP
HALFWORD(35)  -   DATABASE ACCOUNT
HALFWORD(39)  -   SECURITY CLASS
HALFWORD(40)  -   DBOPEN MODE PARAMETER
HALFWORD(41)  -   LOGGING IDENTIFIER
HALFWORD(45)  -   DBSTORE TIME STAMP (3 HALFWORDS)
HALFWORD(48)  -   USER PROGRAM NAME
HALFWORD(52)  -   USER PROGRAM GROUP
```

```
HALFWORD(56)  -   USER PROGRAM ACCOUNT
HALFWORD(60)  -   MODE FROM WHO INTRINSIC
HALFWORD(61)  -   CAPABILITY FROM WHO INTRINSIC
HALFWORD(63)  -   LOCAL ATTRIBUTE FROM WHO INTRINSIC
HALFWORD(65)  -   LOGICAL DEVICE OF JOB/SESSION INPUT
HALFWORD(66)  -   PREVIOUS ROLLBACK TIME STAMP (3 HALFWORDS)
HALFWORD(69)  -   CURRENT ROLLBACK TIME STAMP (3 HALFWORDS)
HALFWORD(72)  -   RESERVED FOR DBRECOV RUN TIME USE
HALFWORD(73)  -   RESERVED FOR DBRECOV RUN TIME USE
```

DBPUT

```
HALFWORD(0-8) -   MPE WRITELOG RECORD
HALFWORD(9)   -   TURBOIMAGE/XL LOG RECORD LENGTH
HALFWORD(10)  -   DBPUT LOG RECORD CODE ("PU")
HALFWORD(11)  -   BASE LOGGING ID
                       BIT 0 TO 5  = USER DBOPEN COUNT
                       BIT 6 TO 15 = INDEX TO THE DBS FOR THE DBG
HALFWORD(12)  -   RECOVERY FLAG ("NO"-FAILED,"OK"-RECOVERED)
HALFWORD(13)  -   TRANSACTION NUMBER (1 WORD)
HALFWORD(15)  -   DATA SET NUMBER
HALFWORD(16)  -   DATA SET TYPE ("MA"-MASTER,"DE"-DETAIL)
HALFWORD(17)  -   RECORD NUMBER (1 WORD)
HALFWORD(19)  -   MODE PARAMETER
HALFWORD(20)  -   OFFSET TO KEY ITEM VALUE (IF MASTER TYPE)
HALFWORD(21)  -   OFFSET TO ITEM LIST
HALFWORD(22)  -   OFFSET TO DATA
HALFWORD(23)  -   BEGIN OF KEY,ITEM LIST,AND DATA BUFFER
```

DBQUIESCE

```
HALFWORD(0-8) -   MPE WRITELOG RECORD
HALFWORD(9)   -   TURBOIMAGE/XL LOG RECORD LENGTH
HALFWORD(10)  -   QUIESCE CODE
                       QB - QUIESCE BEGIN
                       QE - QUIESCE END
                       UQ - UNQUIESCE
HALFWORD(11)  -   FOR INTERNAL USE
HALFWORD(12)  -   QUIESCE FLAG FROM QLOCK TABLE
HALFWORD(13)  -   USER NAME
HALFWORD(17)  -   USER GROUP
HALFWORD(21)  -   USER ACCOUNT
HALFWORD(25)  -   DATABASE NAME
HALFWORD(29)  -   DATABASE GROUP
HALFWORD(33)  -   DATABASE ACCOUNT
HALFWORD(37)  -   USER PROGRAM NAME
HALFWORD(41)  -   USER PROGRAM GROUP
HALFWORD(45)  -   USER PROGRAM ACCOUNT
HALFWORD(49)  -   TRUE ONLINE-BACKUP TIME
HALFWORD(51)  -   TRUE ONLINE-BACKUP DATE
```

DBUPDATE

```
   HALFWORD(0-8) -   MPE WRITELOG RECORD
   HALFWORD(9)   -   TURBOIMAGE/XL LOG RECORD LENGTH
   HALFWORD(10)  -   DBUPDATE LOG RECORD CODE ("UP")
   HALFWORD(11)  -   BASE LOGGING ID
                          BIT 0 TO 5  = USER DBOPEN COUNT
                          BIT 6 TO 15 = INDEX TO THE DBS FOR THE DBG
   HALFWORD(12)  -   RECOVERY FLAG ("NO"-FAILED,"OK"-RECOVERED)
   HALFWORD(13)  -   TRANSACTION NUMBER (1 WORD)
   HALFWORD(15)  -   DATA SET NUMBER
   HALFWORD(16)  -   DATA SET TYPE ("MA"-MASTER,"DE"-DETAIL)
   HALFWORD(17)  -   RECORD NUMBER (1 WORD)
   HALFWORD(19)  -   MODE PARAMETER
   HALFWORD(20)  -   OFFSET TO KEY ITEM VALUE (IF MASTER TYPE)
   HALFWORD(21)  -   OFFSET TO ITEM LIST
   HALFWORD(22)  -   OFFSET TO NEW DATA
   HALFWORD(23)  -   OFFSET TO OLD DATA
   HALFWORD(24)  -   BEGIN OF KEY,ITEM LIST,AND DATA BUFFER
```

DBXBEGIN

```
   HALFWORD(0-8) -   MPE WRITELOG RECORD
   HALFWORD(9)   -   TURBOIMAGE/XL LOG RECORD LENGTH
   HALFWORD(10)  -   DBXBEGIN LOG RECORD CODE ("XB")
   HALFWORD(11)  -   BASE LOGGING ID
                          BIT 0 TO 5  = USER DBOPEN COUNT
                        BIT 6 TO 10 = INDEX TO THE DBS FOR THE DBGHALFWORD(12)  -
RECOVERY FLAG ("NO"-FAILED,"OK"-RECOVERED)
   HALFWORD(13)  -   TRANSACTION NUMBER (1 WORD)
   HALFWORD(15)  -   LENGTH OF USER BUFFER
   HALFWORD(16)  -   START OF USER BUFFER
```

DBXEND

```
   HALFWORD(0-8) -   MPE WRITELOG RECORD
   HALFWORD(9)   -   TURBOIMAGE/XL LOG RECORD LENGTH
   HALFWORD(10)  -   DBXEND LOG RECORD CODE ("XE"), OR
                     ("XA") IF ABORTED
   HALFWORD(11)  -   BASE LOGGING ID
                          BIT 0 TO 5  = USER DBOPEN COUNT
                          BIT 6 TO 10 = INDEX TO THE DBS FOR THE DBG
   HALFWORD(12)  -   RECOVERY FLAG ("NO"-FAILED,"OK"-RECOVERED)
   HALFWORD(13)  -   TRANSACTION NUMBER (1 WORD)
   HALFWORD(15)  -   LENGTH OF USER BUFFER
   HALFWORD(16)  -   START OF USER BUFFER
```

DBXUNDO

```
HALFWORD(0-8) -   MPE WRITELOG RECORD
HALFWORD(9)   -   TURBOIMAGE/XL LOG RECORD LENGTH
HALFWORD(10)  -   DBXEND LOG RECORD CODE ("XU")
HALFWORD(11)  -   BASE LOGGING ID
                       BIT 0 TO 5  = USER DBOPEN COUNT
                       BIT 6 TO 10 = INDEX TO THE DBS FOR THE DBG
HALFWORD(12)  -   RECOVERY FLAG ("NO"-FAILED,"OK"-RECOVERED)
HALFWORD(13)  -   TRANSACTION NUMBER (1 WORD)
HALFWORD(15)  -   LENGTH OF USER BUFFER
HALFWORD(16)  -   START OF USER BUFFER
```

QOPEN

```
HALFWORD(0-8) -   MPE WRITELOG RECORD
HALFWORD(9)   -   TURBOIMAGE/XL LOG RECORD LENGTH
HALFWORD(10)  -   QOPEN LOG RECORD CODE ("QO")
HALFWORD(11)  -   DBOPEN ACTIVE OR INACTIVE
HALFWORD(12-74)-  DBOPEN INFORMATION AS IN DBOPEN LOG RECORD
HALFWORD(75)  -   UNIQUE USER LOGGING PROCESS NUMBER
```

F MPE/iX Log Record Formats

This appendix lists the MPE/iX Log Record Formats for log files and user recovery files.

HEADER

```
HALFWORD(0)   -   RECORD NUMBER (1 WORD)
HALFWORD(2)   -   CHECKSUM
HALFWORD(3)   -   SUBSYSTEM IDENTIFIER (1ST BYTE)
HALFWORD(3)   -   LOG RECORD CODE (2ND BYTE - 4)
HALFWORD(4)   -   TIME
HALFWORD(6)   -   DATE
HALFWORD(7)   -   LOGGING IDENTIFIER
```

OPENLOG

```
HALFWORD(0)   -   RECORD NUMBER (1 WORD)
HALFWORD(2)   -   CHECKSUM
HALFWORD(3)   -   SUBSYSTEM IDENTIFIER (1ST BYTE)
HALFWORD(3)   -   LOG RECORD CODE (2ND BYTE - 1)
HALFWORD(4)   -   TIME
HALFWORD(6)   -   DATE
HALFWORD(7)   -   LOGGING IDENTIFIER
HALFWORD(11)  -   LOG NUMBER
HALFWORD(12)  -   USER NAME,GROUP,ACCOUNT
HALFWORD(24)  -   PIN#
```

WRITELOG

```
HALFWORD(0)   -   RECORD NUMBER (1 WORD)
HALFWORD(2)   -   CHECKSUM
HALFWORD(3)   -   SUBSYSTEM IDENTIFIER (lST BYTE)
HALFWORD(3)   -   LOG RECORD CODE (2ND BYTE - 2)
HALFWORD(4)   -   TIME
HALFWORD(6)   -   DATE
HALFWORD(7)   -   LOG NUMBER
HALFWORD(8)   -   USER BUFFER LENGTH
HALFWORD(9)   -   USER BUFFER AREA
```

WRITELOG CONTINUATION

```
HALFWORD(0)   -   RECORD NUMBER (1 WORD)
HALFWORD(2)   -   CHECKSUM
HALFWORD(3)   -   SUBSYSTEM IDENTIFIER (lST BYTE)
HALFWORD(3)   -   LOG RECORD CODE (2ND BYTE - 7)
HALFWORD(4)   -   TIME
HALFWORD(6)   -   DATE
HALFWORD(7)   -   LOG NUMBER
HALFWORD(8)   -   USER BUFFER LENGTH
HALFWORD(9)   -   USER BUFFER AREA
```

CLOSELOG

```
HALFWORD(0)   -   RECORD NUMBER (1 WORD)
HALFWORD(2)   -   CHECKSUM
HALFWORD(3)   -   SUBSYSTEM IDENTIFIER (1ST BYTE)
HALFWORD(3)   -   LOG RECORD CODE (2ND BYTE - 3)
HALFWORD(4)   -   TIME
HALFWORD(6)   -   DATE
HALFWORD(7)   -   LOGGING IDENTIFIER
HALFWORD(11)  -   LOG NUMBER
HALFWORD(12)  -   USER NAME, GROUP, ACCOUNT
HALFWORD(24)  -   PIN#
```

CHANGELOG

```
HALFWORD(0)   -   RECORD NUMBER (1 WORD)
HALFWORD(2)   -   CHECKSUM
HALFWORD(3)   -   12 RECORD CONTAINS THE previous FILE IN THE SET
HALFWORD(3)   -   13 RECORD CONTAINS THE next FILE IN THE SET
HALFWORD(4)   -   TIME
HALFWORD(6)   -   DATE
HALFWORD(7)   -   LOGID
HALFWORD(11)  -   SEQUENCE NUMBER OF THE CURRENT FILE
HALFWORD(12)  -   CREATION TIME OF THE FIRST FILE
HALFWORD(14)  -   CREATION DATE OF THE FIRST FILE
HALFWORD(15)  -   NAME OF THE FIRST FILE IN THE SET
HALFWORD(33)  -   LOG RECORD TYPE OF THE FIRST FILE IN THE SET
HALFWORD(34)  -   NAME OF THE next FILE IN THE SET
HALFWORD(34)  -   NAME OF THE previous FILE IN THE SET
HALFWORD(52)  -   LOG RECORD TYPE OF THE next FILE IN THE SET
HALFWORD(52)  -   LOG RECORD TYPE OF THE previous FILE IN THE SET
HALFWORD(53)  -   NAME OF THE CURRENT FILE IN THE SET
HALFWORD(71)  -   LOG RECORD TYPE OF THE CURRENT FILE IN THE SET
```

RESTART

```
HALFWORD(0)   -   RECORD NUMBER (1 WORD)
HALFWORD(2)   -   CHECKSUM
HALFWORD(3)   -   SUBSYSTEM IDENTIFIER (1ST BYTE)
HALFWORD(3)   -   LOG RECORD CODE (2ND BYTE - 6)
HALFWORD(4)   -   TIME
HALFWORD(6)   -   DATE
HALFWORD(7)   -   LOGGING IDENTIFIER
```

CRASH

```
HALFWORD(0)   -   RECORD NUMBER (1 WORD)
HALFWORD(2)   -   CHECKSUM
HALFWORD(3)   -   SUBSYSTEM IDENTIFIER (1ST BYTE)
HALFWORD(3)   -   LOG RECORD CODE (2ND BYTE - 9)
HALFWORD(4)   -   TIME
HALFWORD(6)   -   DATE
```

NULL

```
HALFWORD(0)   -   RECORD NUMBER (1 WORD)
HALFWORD(2)   -   CHECKSUM
HALFWORD(3)   -   RECORD CODE (BLANK BLANK)
```

TRAILER

```
HALFWORD(0)   -   RECORD NUMBER (1 WORD)
HALFWORD(2)   -   CHECKSUM
HALFWORD(3)   -   SUBSYSTEM IDENTIFIER (1ST BYTE)
HALFWORD(3)   -   LOG RECORD CODE (2ND BYTE - 5)
HALFWORD(4)   -   TIME
HALFWORD(6)   -   DATE
HALFWORD(7)   -   LOGGING IDENTIFIER
```

G Recovery and Logging Quick Reference

Recovery Quick Reference

The following pages offer a very brief outline of the recovery options in TurboIMAGE/XL. For more information regarding dynamic roll-back recovery, Intrinsic Level Recovery (ILR), roll-forward recovery and roll-back recovery refer to chapter 7, "Logging and Recovery," and chapter 8, "Using the Database Utilities."

You should determine which type of recovery to use based on the size of the database, frequency of system failures, equipment availability, and other considerations.

Dynamic Roll-Back Recovery

- Allows dynamic transactions to be rolled back on-line while other database activity is occurring. If a system failure occurs, incomplete dynamic transactions will be rolled back at the first call to DBOPEN for the database after the system is restarted.
- Uses DBXBEGIN, DBXEND, and DBXUNDO intrinsics.
- Requires strong locking. Calling DBUNLOCK after a call to DBPUT, DBDELETE or DBUPDATE within a dynamic transaction will return an error because the call to DBUNLOCK must occur after the call to DBXEND. If necessary locks are not acquired before calling DBXBEGIN, covering locks must be used on the intrinsic calls within the transaction, or the intrinsic will return an error.
- Requires checking of the status condition before proceeding further after a modification using DBPUT, DBDELETE, DBUPDATE, or after DBUNLOCK, or DBCLOSE in mode 2. If an error is returned indicating the failure of the intrinsic, the choices are:
  - — Call DBXEND. The successful modifications completed within this dynamic transaction will not be rolled back.
  - — Call DBXUNDO to roll back the entire transaction. Even successful modifications completed within this dynamic transaction will be rolled back.
  - — Continue with the remainder of the dynamic transaction even though this intrinsic failed. Be very cautious in your design when taking this option as the final result can be unpredictable.
- Cannot use deferred output (AUTODEFER). This ensures the structural integrity of the database.
- Can use user logging for roll-forward recovery purposes in the event of a disk media failure.
- Can be used with remote databases.

Intrinsic Level Recovery (ILR)

- Should only be used to force flushing to disk after a call to DBPUT or DBDELETE.
- Not required with TurboIMAGE/XL, because physical integrity is automatic and transparent to the user. The same physical integrity is available through Transaction Management (XM) on MPE/iX without enabling ILR. Using XM instead of ILR requires less overhead.
- Guarantees that at most one DBPUT and DBDELETE call per process will be lost. The database is recovered automatically if recovery is needed the next time the system is rebooted.
- Cannot use deferred output (AUTODEFER). This ensures the structural integrity of the database.
- Cannot defer writing modifications to the database because deferred output cannot be used.
- Is intrinsic driven; transaction locking is not necessary.
- Incurs some overhead on DBPUTs and DBDELETEs due to the request to flush Transaction Management (XM) log file pages to disk at the end of each completed DBPUT and DBDELETE.
- Can be used with user logging.

Roll-Forward Recovery

- Provides recovery of a database, both structurally and logically, to a likeness of its state at the time of a hard system failure.
- Uses DBSTORE and DBRESTOR or TurboSTORE/iX 7x24 True-Online Backup (with ONLINE=START or ONLINE=END option) to backup the database and MPE/iX RESTORE command to restore the database to a consistent state.
- If True-Online Backup was not used to store the database, and the AUTO option is in use, all logfiles in the logging cycle must be present for recovery. All logfiles starting with the first one, *XXXXX*001, are required regardless of some intermediate recovery or use of the STOP/RESTART feature.
- Requires logging be enabled. During recovery all log files since the last database backup copy must be applied.
- TurboIMAGE/XL logging depends upon exact correspondence between the stored backup database copy and the working database on disk at the time logging was interrupted. The DBSTORE flag and log file time stamp will enforce this condition.
- Logging provides recovery of both intrinsics and transactions following a system failure.
- Is initiated with the >RECOVER command of DBRECOV. The database must be purged and restored before recovery is initiated.
- During a start recovery operation all transactions in the memory buffer will be lost.
- Does not require logical transaction locking; however, it is recommended.

- When the roll-forward process finishes, the RESTART option in user logging can be used or a new logging cycle can be started. Remember to purge the log files before starting the new logging cycle.
- May not recover all transactions. If using DBEND mode 2, transactions will be flushed to the log file at DBEND time; therefore, transactions will not be lost.
- You should consider how often to store the database. The more frequent the backup, the smaller the log files will be, thus cutting the time required for recovery.

Roll-Forward Flag Settings

The recommended settings for TurboIMAGE/XL flags used for roll-forward recovery are shown in the following table. Note that the flags change depending on whether you are storing a database, running in production, or initiating a database recovery.

Table G-1. Roll-Forward Flag Settings

Flags	True-Online Backup	DBSTORE	Production	DBRECOV
Access	Disabled/Enabled	Disabled	Enabled	Disabled
AUTODEFER	Disabled/Enabled	Disabled	Disabled/Enabled	Disabled
Dumping	Disabled	Disabled	Disabled	Disabled
ILR	Disabled	Disabled	Disabled	Disabled
Logging	Enabled	Enabled	Enabled	Enabled
Recovery	Disabled/Enabled	Enabled	Disabled	Enabled
Roll-back	Disabled	Disabled	Disabled	Disabled

Roll-Back Recovery

- Provides rapid recovery of database integrity following a soft system crash and restores the database to a consistent state, physically and logically, by backing out any incomplete transactions.
- Requires logging. Storing the database is recommended but not required. (Logging is automatically enabled when roll-back is enabled.)
- If logging to disk, requires that the database and the user log file be on the same volume set.
- If logging to tape, requires that the database be on the system volume set.
- Is initiated with the >ROLLBACK command of DBRECOV.
- When disabled, logging must be manually disabled using DBUTIL.
- Requires all multiple-intrinsic database transactions to execute independently, using logical transaction locking.
- Uses the time stamp during recovery to verify the correct log file for each database being recovered. The time stamp is updated when the database 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 database will be considered correct and cannot be rolled back.
- Transactions are not lost during a start recovery operation because they are not held in the memory buffer. (A start recovery operation must be performed after a system failure.)
- When the roll-back 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.

Roll-Back Flag Settings

The recommended settings for TurboIMAGE/XL flags used for roll-back recovery are shown in the following table. Note that the flags change depending on whether you are storing a database, running in production, or initiating a database recovery.

Table G-2. Roll-Back Flag Settings

Flags	True-Online Backup	DBSTORE	Production	DBRECOV
Access	Disabled/Enabled	Disabled	Enabled	Disabled
AUTODEFER	Disabled	Disabled	Disabled	Disabled
Dumping	Disabled	Disabled	Disabled	Disabled
ILR	Disabled	Disabled	Disabled	Disabled
Logging	Enabled	Enabled	Enabled	Enabled
Recovery	Disabled/Enabled	Enabled	Disabled	Enabled
Roll-back	Enabled	Enabled	Enabled	Enabled

Recovery

- Without a database backup copy, recovery can be performed using DBUNLOAD (chained) and DBLOAD to salvage all or most of a database.

Logging Device Quick Reference

The lists below outline both the benefits and disadvantages of logging to disk and logging to tape. Refer to chapter 7, "Logging and Recovery" for more information on user logging.

You must determine which type of logging device to use based on equipment, operations staff, number of users and size of the database, and other considerations listed here.

Logging to Tape

- For roll-back recovery, requires the database be on the system volume set.
- Does not take up disk 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. The 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 suspend. Applications requiring logging will get a `WRITELOG` error and will terminate.
- Must do a start recovery operation after a soft system crash to write a crash log record to the log file.
- During a start recovery operation, the console operator can respond with an option to override or cancel the cleanup procedure on log files. Fewer log records written just prior to system failure are lost.
- 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. Read the labels on the MPE/iX tapes for overwrite security information.
- Has overhead similar to logging to disk.

Logging to Disk

- For roll-back recovery, requires the database and the user log file to be on the same volume set.
- Files are susceptible to a hard crash.
- The integrity of the log file may be no better than the current database state. The log file may contain inconsistencies, bad characters, or other invalid data.
- If you are not using the `GETLOG` command `AUTO` option or the `CHANGELOG` command, you must make sure disk file space of the current log file is sufficient so that end of file is not reached. If the end of file is reached, logging will stop. Applications requiring logging will get a `WRITELOG` error and will terminate.

- Must do a start recovery operation after a soft system crash to write a crash log record to the log file.
- Has overhead comparable to logging to tape.

NOTE In the event of a system failure and subsequent start recovery operation and when using private volumes, logging will not resume until these private volumes have been mounted. Enter the MPE `VMOUNT` command into the `SYSSTART` file to resume logging.

Sample Job Streams

This section shows a sample job stream which can be used to initiate a logging cycle. Prior to using this job stream, you should check the MPE/iX logging configuration for adequate capacity. Also shown in this section are sample job streams which can be used to recover a database using either roll-forward recovery or roll-back recovery. You must decide which recovery method to use and how often the database should be stored.

NOTE You must have logging (LG) or operator (OP) capability to use the following MPE/iX commands: `GETLOG`, `RELLOG`, `ALTLOG`, `CHANGELOG`, `LISTLOG`, and `SHOWLOGSTATUS`. You must have LG or OP capability also if you intend to open a database with logging enabled. Logging capability is acquired through the MPE system manager and account manager commands.

In the following example, roll-forward recovery is being used and `DBSTORE` is used to backup the database. If using roll-back recovery, replace the line `>>ENABLE ORDERS FOR LOGGING` with `>>ENABLE ORDERS FOR ROLLBACK`. This enables the database for logging and for roll-back recovery.

Figure G-1. Sample Job Stream for Starting Logging Cycle

```
:JOB MGR.DATAMGT
:GETLOG ORDERLOG;LOG=ORDER001,DISC          Acquire log identifier.

:BUILD ORDER001;DISC=200000,20,7;CODE=LOG   Build new log file.

:RUN DBUTIL.PUB.SYS
SET ORDERS LOGID=ORDERLOG
                                            Response to logid password prompt.

ENABLE ORDERS FOR LOGGING                   Set the database flags in
DISABLE ORDERS FOR ACCESS                   the root file.
ENABLE ORDERS FOR RECOVERY
ENABLE ORDERS FOR MUSTRECOVER
EXIT
:RUN DBSTORE.PUB.SYS                        Store the database.
ORDERS

:LOG ORDERLOG,START                         Start the logging process;
                                            logid is ORDERLOG.

:RUN DBUTIL.PUB.SYS
ENABLE ORDERS FOR ACCESS                    Set the database flags
DISABLE ORDERS FOR RECOVERY                  in the root file.
ENABLE ORDERS FOR MUSTRECOVER
EXIT
:EOJ
```

The job stream above builds a new log file. In this case, the log file resides on disk and sets the database flags. Note that because no logid password was specified in the GETLOG command, a blank line is left as a response to the password prompt. A backup copy of the database is made (this sets the date and time the copy was made in the root file), logging is initiated with START, the database is enabled for access, and recovery is disabled.

At the end of the logging cycle, stop logging, store the current log file on tape for back-up, purge the current log file, build a new log file, and store a database backup copy. To start a logging cycle, the steps (except GETLOG command) shown in the previous job stream in Figure G-1. are completed after you do the following:

```
:LOG ORDERLOG,STOP    Stop logging.
                      Use STORE command to store the log file to tape.
:PURGE ORDER001       Purge the current log file.
```

The next example in Figure G-2. uses roll-forward recovery and DBSTORE/DBRESTORE. If all recommended procedures have been followed, the database backup copy will have flags set for enabling recovery and disabling access, so the step to set these flags would be unnecessary. If this process is being done interactively, the following command in DBUTIL will show if the flags for recovery and access are correctly set:

```
>>SHOW database name FLAGS
```

Figure G-2. Sample Job Stream for Roll-Forward Recovery

```
:JOB MGR.DATAMGT
:RUN DBUTIL.PUB.SYS              If the database is to be stored prior
DISABLE ORDERS FOR ACCESS        to recovery, set flags in the database
ENABLE ORDERS FOR RECOVERY       and run DBSTORE.
EXIT
:RUN DBSTORE.PUB.SYS             Current database on the system.
ORDERS                           Optional step.

:RUN DBUTIL.PUB.SYS
PURGE ORDERS                     Purge the current database.
EXIT
:RUN DBRESTOR.PUB.SYS            Restore backup copy of the database for recovery.
ORDERS


:RUN DBUTIL.PUB.SYS
DISABLE ORDERS FOR ACCESS         Set the flags in the database.
ENABLE ORDERS FOR RECOVERY
ENABLE ORDERS FOR MUSTRECOVER
EXIT
:RUN DBRECOV.PUB.SYS              Use roll-forward recovery on
RECOVER ORDERS                    database ORDERS.
FILE PART1,SYS/P1D1.MKTG,0,3      The FILE command is used to
FILE PART2,SYS/P1D1.MKTG,0,3      route log records to individual
FILE PART3,SYS/P1D1.MKTG,0,3       user log files.
RUN
EXIT
:LOG ORDERLOG,RESTART             Restart current log file and set the
:RUN DBUTIL.PUB.SYS               database flags.
ENABLE ORDERS FOR ACCESS
DISABLE ORDERS FOR RECOVERY
ENABLE ORDERS FOR MUSTRECOVER
EXIT
:EOJ
```

Storing and purging the damaged database prior to restoring it 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.

The next example in Figure G-3. uses roll-back recovery. If all recommended procedures have been followed, the database will have flags set for enabling recovery and disabling access, so the step to set these flags would be unnecessary. A backup of the database using DBSTORE or TurboSTORE/iX 7x24 True-Online Backup (with ONLINE=START or ONLINE=END option) can be done at this time; however, this is optional. If this process is being done interactively, the following command in DBUTIL will show if the flags for recovery and access are correctly set:

```
>>SHOW database name FLAGS
```

Figure G-3. Sample Job Stream for Roll-Back Recovery

```
:JOB MGR.DATAMGT
:RUN DBUTIL.PUB.SYS
DISABLE ORDERS FOR ACCESS       Set the flags in the database
ENABLE ORDERS FOR RECOVERY      root file.
EXIT

  A DBSTORE of the database at this time is recommended.

:RUN DBRECOV.PUB.SYS
CONTROL NOSTORE                 Allows recovery to proceed whether or not the
DBSTORE flag is set.
ROLLBACK ORDERS                 Use roll-back recovery on database ORDERS.
FILE PART1,SYS/P1D1.MKTG,0,3    The FILE command is used to route log
FILE PART2,SYS/P1D1.MKTG,0,3    records to individual user files.
FILE PART3,SYS/P1D1.MKTG,0,3
RUN
EXIT
:LOG ORDERLOG,RESTART           Restart the current log file and set
:RUN DBUTIL.PUB.SYS             the database flags.
ENABLE ORDERS FOR ACCESS
DISABLE ORDERS FOR RECOVERY
EXIT
:EOJ
```

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.

The following job stream shows how you can start a new log cycle for roll-forward recovery using TurboSTORE/iX 7x24 True-Online Backup (with `ONLINE=START` or `ONLINE=END` option) to backup the database. If using roll-back recovery, replace the line `>>ENABLE ORDERS FOR LOGGING` with `>>ENABLE ORDERS FOR ROLLBACK`. This enables the database for logging and for roll-back recovery.

Figure G-4. Sample Job Stream for Starting Logging Cycle

```
:JOB MGR.DATAMGT
:GETLOG ORDERLOG;LOG=ORDER001,DISC;AUTO      Acquire log identifier.  AUTO is
optional.

:BUILD ORDER001;DISC=200000,20,7;CODE=LOG    Build new log file.

:RUN DBUTIL.PUB.SYS
SET ORDERS LOGID=ORDERLOG                    Response to logid password prompt.

ENABLE ORDERS FOR LOGGING                   Set the database flags in the root file
DISABLE ORDERS FOR ACCESS
ENABLE ORDERS FOR RECOVERY
ENABLE ORDERS FOR MUSTRECOVER
EXIT
:FILE ORDERBK1;DEV=TAPE                      Mount the tape.
:STORE ORDERS;*ORDERBK1;ONLINE=START         Store the database.


:LOG ORDERLOG,START                          Start the logging process; logid is
ORDERLOG.


:RUN DBUTIL.PUB.SYS
ENABLE ORDERS FOR ACCESS                     Set the database flags
DISABLE ORDERS FOR RECOVERY                   in the root file.
ENABLE ORDERS FOR MUSTRECOVER
EXIT
:EOJ
```

The job stream above builds a new log file. In this case, the log file resides on disk and sets the database flags. Note that because no logid password was specified in the GETLOG command, a blank line is left as a response to the password prompt. A backup copy of the database is made (this sets the date and time the copy was made in the root file), logging is initiated with START, the database is enabled for access, and recovery is disabled. The database is ready for access.

If you want to make another backup to keep recovery time short, you do not need to stop logging. Also the database could be open for access. You can backup the database using TurboSTORE/iX 7x24 True-Online Backup (with ONLINE=START or ONLINE=END option) and optionally store and purge the log files (if AUTO option was used with GETLOG) preceding the log file which was in use at the time of the backup. If you do not purge the preceding log files, you can continue until log file ORDER999. After that you will get an error when attempting to write to LOG0001.

Figure G-5. Sample Job Stream for Backup with Database Open for Access

```
:JOB MGR.DATAMGT
:FILE ORDERBK2;DEV=TAPE                    Mount tape.
:STORE ORDERS;*ORDERBK2;ONLINE=START        Store database while it is open for
access.
:EOJ                                       Assume that you will start a new log
cycle after ORDER999.
```

The next example in Figure G-6. uses roll-forward recovery and TurboSTORE/iX 7x24 True-Online Backup (with ONLINE=START or ONLINE=END option). If all recommended procedures have been followed, the database backup copy will have flags set for enabling recovery and disabling access, so the step to set these flags would be unnecessary. If this process is being done interactively, the following command in DBUTIL will show if the flags for recovery and access are correctly set:

```
>>SHOW database name FLAGS
```

Figure G-6. Sample Job Stream for Roll-Forward Recovery

```
:JOB MGR.DATAMGT
:FILE ORDERBAD;DEV=TAPE                   Current damaged database on the system
needing recovery.
:STORE ORDERS;*ORDERBAD;ONLINE=START        Store damaged database; optional step.

:RUN DBUTIL.PUB.SYS
PURGE ORDERS                               Purge the current database.
EXIT
:FILE ORDERBK2;DEV=TAPE                    Mount the backup tape.
:RESTORE *ORDERBK2;ORDERS                 Restore the backup copy of the database
for recovery.

:RUN DBUTIL.PUB.SYS
DISABLE ORDERS FOR ACCESS                  Set the flags in the database.
ENABLE ORDERS FOR RECOVERY
ENABLE ORDERS FOR MUSTRECOVER
EXIT
:RUN DBRECOV.PUB.SYS                       Use roll-forward recovery on
RECOVER ORDERS                             database ORDERS.
FILE PART1,SYS/P1D1.MKTG,0,3               Use the FILE command to route log
records
FILE PART2,SYS/P1D1.MKTG,0,3               to individual user log files.
FILE PART3,SYS/P1D1.MKTG,0,3
RUN
EXIT
:LOG ORDERLOG,RESTART                      Restart current log file and set the
:RUN DBUTIL.PUB.SYS                        database flags.
ENABLE ORDERS FOR ACCESS
DISABLE ORDERS FOR RECOVERY
ENABLE ORDERS FOR MUSTRECOVER
EXIT
:EOJ
```

Storing and purging the damaged database prior to restoring it is optional. After recovery has completed, logging can either be restarted (with the current log file) or the log file can be purged and a new log file built.

The next example in Figure G-7. uses roll-back recovery. A backup of the database using DBSTORE or TurboSTORE/iX 7x24 True-Online Backup (with ONLINE=START or ONLINE=END option) can be done at this time; however, this is optional. The following command in DBUTIL will show if the flags for recovery and access are correctly set:

```
>>SHOW database name FLAGS
```

Figure G-7. Sample Job Stream for Roll-Back Recovery

```
:JOB MGR.DATAMGT
:RUN DBUTIL.PUB.SYS
DISABLE ORDERS FOR ACCESS       Set the flags in the database
ENABLE ORDERS FOR RECOVERY      root file.
EXIT

 A backup of the database at this time is recommended.

:RUN DBRECOV.PUB.SYS
CONTROL NOSTORE                 Allows recovery to proceed whether or not the
DBSTORE flag is set.
ROLLBACK ORDERS                 Use roll-back recovery on database ORDERS.
RUN
EXIT
:LOG ORDERLOG,RESTART           Restart the current log file and set
:RUN DBUTIL.PUB.SYS             the database flags.
ENABLE ORDERS FOR ACCESS
DISABLE ORDERS FOR RECOVERY
EXIT
:EOJ
```

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.

H TurboIMAGE/XL versus TurboIMAGE/V

TurboIMAGE/XL is available on the HP 3000 Series 900. This database management system is very similar to TurboIMAGE/V. However, some differences exist due to the architecture of the 900 series and the MPE/iX features of which TurboIMAGE/XL takes advantage.

Overview

Specific TurboIMAGE/XL differences are listed here:

- Transaction Management (XM), an internal MPE/iX service, is used to do the following:
  - — Ensure intrinsic level physical consistency of the database as a default unless the AUTODEFER option is enabled. Incomplete intrinsics will be backed out and not reapplied. This performs the same functionality as Intrinsic Level Recovery (ILR) on MPE V.
  - — Guarantee via Intrinsic Level Recovery (ILR) that not more than one DBDELETE or DBPUT per process will be lost. ILR should be disabled on TurboIMAGE/XL.
  - — Recover transactions via dynamic roll-back recovery while other database activity is occurring.
- Mapped files are used instead of extra data segments for run-time control blocks. Another control block, the Database User Local Index (DBUX), has been added.
- The maximum number of DBOPENs per process is 127 (63 per database) depending on the availability of system resources.
- Some status area information returned by TurboIMAGE/XL library procedures is different for MPE/iX applications because of the change from 16-bit to 32-bit architecture.
- Some information returned by DBINFO mode 402 is different because of the change in how ILR is implemented.
- For roll-back recovery: If logging to disk, the user log file and the database must reside in the same volume set. If logging to tape, the database must be in the system volume set.
- DBSTORE has a TRANSPORT option in the INFO=string for moving TurboIMAGE/XL databases to MPE V.
- Better integrated third-party indexing software that works in conjunction with TurboIMAGE/XL provides the capability to do generic key searches, multiple keyword

retrievals, and sorted sequential searches on any database that is enabled for third-party indexing.

- A logical transaction bracketed by DBXBEGIN and DBXEND can be dynamically rolled back. This dynamic transaction can be of 1 to 15 databases.
- A data set can be a jumbo set exceeding 4 gigabytes.
- B-tree indices on key items of master data sets allow generic and range searches on key items of master sets as well as corresponding detail data set search items.
- Databases can be stored, even when they are open for read/write access, using TurboSTORE/iX 7x24 True Online Backup (with ONLINE=START or ONLINE=END option). Their related files, such as TC file and third-party index files, are stored along with the databases.
- DBRECOV can perform a roll-forward recovery for a database stored using TurboSTORE/iX 7x24 True Online Backup (with ONLINE=START or ONLINE=END option).
- Data sets, master as well as detail, can be dynamically expanded during DBPUT.

The rest of this appendix discusses the major differences in detail. They include the following:

- Intrinsic Level Recovery
- Control Blocks
- Status Area
- Moving from MPE/iX to MPE V

TurboIMAGE/XL differences are summarized in Table H-1.

Moving to TurboIMAGE/XL

To move a database from MPE V to MPE/iX, you need to perform the following tasks:

1. Disable both ILR and roll-back recovery (use the >>DISABLE command of DBUTIL).
2. On MPE V, store the database with DBSTORE (or use the MPE V STORE command).
3. On MPE/iX, restore the database with DBRESTOR (or use the MPE/iX RESTORE command).

For more detailed migration information, refer to the *Migration Process Guide*.

Table H-1. TurboIMAGE/XL Differences

Area Affected	Changes/Additions
Data Set Size	– The maximum data set size is 4 gigabytes unless it is a jumbo data set.
DBEXPLAIN	– Information for `DBEXPLAIN` is stored in the Database User Local Control Block (DBU). – Calls to `DBEXPLAIN` must be made immediately after receiving an error status.
DBOPEN	– A maximum of 127 `DBOPEN`s is allowed per process (63 per database).
DBSTORE Command	– A TRANSPORT option has been added for use when moving a TurboIMAGE/XL database to MPE V. – The move from MPE/iX to MPE V may not be possible if data sets are larger than the MPE V file size limit.
Dynamic Roll-Back Recovery	– Physical integrity of the database is provided by MPE/iX Transaction Management (XM) logging, by default. – Dynamic roll-back works in conjunction with XM and ensures the logical integrity of the database after a program abort or system failure. – Transactions are rolled back online while other database activity is occurring. – Dynamic transaction can span single or multiple databases.
Intrinsic Level Recovery	– Physical consistency of the database is provided by MPE/iX Transaction Management as a default. – The ILR log file is replaced by the MPE/iX Transaction Management log file. – Incomplete intrinsics are backed out and will not be reapplied. – `DBUPDATE` is included in the recovery. – `DBINFO` mode 402 no longer contains ILR recovery information. – To move between MPE V and MPE/iX, disable ILR and roll-back recovery. – ILR is not required for roll-back recovery.
Remote Database Access	– Only NS/3000 is supported on MPE/iX.
Roll-Back Recovery	– ILR is not required.

Table H-1. TurboIMAGE/XL Differences

Area Affected	Changes/Additions
	– If logging to disk, the user log file and the database must be in the same volume set. If logging to tape, the database must be on the system volume set.
Run-Time Control Blocks	– Mapped files are used for control blocks. – A control block called the Database User Local Index (DBUX), which is unique to a process, has been added. It contains dynamic roll-back information, as well as the addresses and file numbers of all the DBUs and DBRs belonging to a specific process. – The database ID number returned by `DBOPEN` now serves as an index into the DBUX.
Status Area	– Status elements 5, 7, 8, and 10 are modified for Native Mode (NM) applications. – For condition code -9, elements 2, 3, and 4 have a new format for both Compatibility Mode (CM) and NM applications.
TURBO Trace and Profiler	– TURBO Trace and Profiler are not available.
Third-Party Indexing	– If your database is configured for indexing using a supported third-party product, you have the capability to do generic key searches, multiple keyword retrievals, and sorted sequential searches on that database using the `DBFIND` and `DBGET` intrinsics. Refer to your third-party vendor documentation for details.
B-Tree Indices	– If a B-tree index exists on the key item of the master, allows you to do generic and range searches using this key item, or search items of its corresponding detail data sets. `DBFIND` can be used both for master and detail. New feature for TurboIMAGE/XL; not in TurboIMAGE/V.
Master and Detail Data Set Expansion	– Data set can be expanded dynamically according to capacity expansion parameters. New feature for TurboIMAGE/XL; not in TurboIMAGE/V.
Jumbo Data Sets	– Allows you to create data sets greater than 4 gigabytes. New feature for TurboIMAGE/XL; not in TurboIMAGE/V.
Dynamic Multiple Database Transaction (DMDBX)	– This is an extension of the dynamic transaction for a single database. The dynamic transaction can span 1-15 databases. New feature for TurboIMAGE/XL; not in TurboIMAGE/V.
Deadlock Detection	– Database can be activated for deadlock detection to avoid system hangs. New feature for TurboIMAGE/XL; not in TurboIMAGE/V.

Intrinsic Level Recovery

Intrinsic Level Recovery (ILR) is not required to protect the structural integrity of the database, because in default recovery mode TurboIMAGE/XL takes advantage of an internal MPE/iX file system service, called Transaction Management (XM), to provide intrinsic level backup and recovery. Unless AUTODEFER is enabled, all intrinsics that modify the database (DBPUTs, DBDELETEs, and DBUPDATEs) are written to XM log file pages by MPE/iX. In default recovery mode, XM log file pages are only written to disk when one of the following occurs:

- A system-specified time has elapsed.
- A request is made by a subsystem, such as TurboIMAGE/XL, to flush the XM log file to disk (for example, ILR is enabled, or DBEND mode 2 is specified and logging is enabled).
- The XM buffer is full.

Thus, intrinsics that have completed may not yet be written to disk. If a system failure occurs, only intrinsics that have been written to disk are recovered. *When recovery is necessary, it must be performed at system startup time before anyone modifies the database.*

ILR also uses the MPE/iX Transaction Management service. When ILR is enabled, XM log file pages are written to disk at the end of each completed DBDELETE and DBPUT. If a system failure occurs, at most only one DBPUT or DBDELETE is not recovered. (Note that a completed DBUPDATE does not force a XM log write to disk.) Because of these differences in implementation, ILR on TurboIMAGE/V is not completely compatible with ILR on TurboIMAGE/XL.

ILR does have performance implications. Therefore, you may wish to reevaluate your use of ILR with TurboIMAGE/XL. ILR is no longer necessary to ensure the structural integrity of the database. However, if your data is not easily recoverable (information taken over the telephone, for example) you may wish to use ILR because at most only one intrinsic can be lost.

NOTE Before moving TurboIMAGE/V databases to MPE/iX, it is recommended that ILR be disabled. If it is not, TurboIMAGE/XL assumes you want ILR to remain enabled.

Major Differences

- ILR is performed by MPE/iX Transaction Management. As a result, the ILR file *DatabaseName00* does not exist. All transactions are logged to the MPE/iX Transaction Management log file.
- ILR is not required to protect the structural integrity of the database.
- ILR is not required for roll-back recovery.
- DBUPDATE is included with DBPUT and DBDELETE in Intrinsic Level Recovery. However, completed DBUPDATEs are not guaranteed to be committed to the log file for recovery.

- On TurboIMAGE/XL, only completed DBPUTs and DBDELETEs are recovered by ILR if the database was enabled for ILR. If these intrinsics are interrupted by a system failure or other abnormal termination, they are not recovered. On TurboIMAGE/V, if the database is enabled for ILR, the last DBPUT or DBDELETE, which was interrupted by a system failure or other abnormal termination, will be completed.
- *When recovery is necessary, it must be performed at system startup time before anyone modifies the database.* This can cause a slightly longer startup time. (If a program accessing a TurboIMAGE/XL database aborts during the execution of an intrinsic, the incomplete intrinsic will be undone.)
- On TurboIMAGE/XL, DBINFO mode 402 does not return information about whether or not ILR recovery has been done on the last DBPUT or DBDELETE. (Because database recovery is now performed at system startup time, this information is not available.)

  Table H-2. compares DBINFO mode 402 on TurboIMAGE/V with DBINFO mode 402 on TurboIMAGE/XL.

Table H-2. DBINFO Mode 402 Changes

Element	TurboIMAGE/V	TurboIMAGE/XL
1	ILR log flag: 1 if enabled; 0 if not enabled.	No change.
2	Calendar date ILR was enabled.	No change.
3, 4	Clock time ILR was enabled.	No change.
5	1 if ILR used; 0 if ILR not used.	Always 0.
6	P if DBPUT; D if DBDELETE; otherwise blank.	Always blank.
7–14	Data set name, when ILR used; otherwise blank.	Always blank.
15–16	Reserved.	No change.

Control Blocks

Run-time control blocks are created differently on MPE/iX. A new control block called the Database User Local Index (DBUX), which is unique to each process, has been added. Also, four new global tables to monitor system-wide database activity are added. See the descriptions below for these four tables that were added:

- TURBOLKT
- TURBOGTX
- QOPEN
- QLOCK

Major Differences

Run-time control blocks are no longer created as privileged extra data segments. Instead, on MPE/iX privileged mapped files are used. If an error occurs when MPE/iX opens a mapped file, the following error is returned for status code –9:

```
CANNOT CREATE controlblockname: FILE SYSTEM ERROR nn
```

where *controlblockname* is one of the following:

```
DBS
DBG
DBU
DBR
DBUX
```

and *nn* is the number of the file system error. Refer to "Local Database Access" in Chapter 10.

Specifically, control blocks are created as follows:

- The Database System Control Block (DBS) is stored in a permanent mapped file called TURBODBS.PUB.SYS. It is created as a permanent file and is unique to a system. DBOPEN creates it if it does not exist and resets it after a system abort.
- The Database Globals Control Block (DBG) and the Database Buffer Area Control Block (DBB) are stored in a permanent mapped file called *DatabaseName*GB which resides in the same group and account as the database. *DatabaseName*GB is created when the first user opens the database and purged when the last user exits.
- Each Database User Local Control Block (DBU) is stored in an unnamed/new mapped file. A DBU is created each time a user does a DBOPEN for local database access.
- Each Database Remote Control Block (DBR) is stored in an unnamed/new mapped file. A DBR is created each time a user does a DBOPEN for remote database access.
- The Database User Local Index Control Block (DBUX) is a new run-time control block

stored in an unnamed/new mapped file. One DBUX is created per process. The database ID number serves as an index into the DBUX and points to the virtual addresses of all current DBU/DBRs belonging to that process. 127 entries are allowed, meaning that each user (process) is allowed 127 DBOPENs (63 per database). Refer to the discussion of DBOPEN in Chapter 5 for additional information. If the DBUX is full and the user attempts to open another database, the following error is displayed for status code –198:

```
TOTAL DBOPEN COUNT PER USER EXCEEDS LIMIT OF 127
```

- The Database Lock Table (TURBOLKT) contains information pertaining to locks on the database and is used to avoid deadlocks.
- The Multi-Database Transactions (TURBOGTX) contains information pertaining to dynamic transaction constituting multiple databases.
- The QOPEN Table (QOPEN) contains information about user logging process for active DBOPENS of modes 1-4.
- The QLOCK Table (QLOCK) contains information about all writers to databases and a pointer to QOPEN.

Status Area

All addresses on MPE/iX are 32 bits. This has necessitated a change in some of the information returned in the status area by TurboIMAGE/XL library procedures.

Major Differences

Because TurboIMAGE/XL uses MPE/iX mapped files, DBOPEN calls HPFOPEN to open these mapped files. If an error occurs during this process, the condition code –9, formerly used to indicate an MPE/iX GETDSEG failure, is now used to indicate an MPE/iX HPFOPEN failure. DBOPEN returns the HPFOPEN status in the status array as shown in Table H-3.

Table H-3. Condition Code -9 Status Array

Element	Content
1	–9
2	Control Block Code DBG = 1 DBU = 2 DBR = 3 DBS = 4 DBUX = 5
3	HPFOPEN File System Error (16 bits)
4	File System Intrinsic Code

When DBOPEN is successful, elements 3 and 4 still contain the size, in halfwords, of the DBG and DBU respectively. However, because the maximum value that can be reported is 32,768 halfwords (65,536 bytes), if the DBG or the DBU/DBR is larger than that, only 32,768 halfwords are reported in the status area. Note that although on MPE/iX words are 32-bit words, TurboIMAGE/XL still returns all lengths in the status area as the number of 16-bit halfwords.

For applications in Compatibility Mode (CM):

- Switch stubs translate the information returned to the status area by TurboIMAGE/XL. After translation, each element of the status area contains the information expected by TurboIMAGE/XL applications with one exception. Element 10 contains the relative address of the switch stub rather than that of the actual library procedure (intrinsic).

For applications in Native Mode (NM):

- On TurboIMAGE/XL, elements 5 and 10 of the status area do not contain code offsets because they are now 32 bits. Halfwords 5 and 10 return 0.
- Halfwords 7 and 8 contain the 32-bit address of the database parameter.

- DBEXPLAIN cannot find complete information in the status area for its explanation. To solve this problem, TurboIMAGE/XL places the missing information in the DBU, which DBEXPLAIN now references. However, the application must call DBEXPLAIN immediately after the status information is received. If any other library procedure is called between the time the error status is returned and DBEXPLAIN is called, DBEXPLAIN displays the last information stored in the DBU, which may or may not belong to the library procedure that encountered the error. This could affect applications that use two or more different status arrays.

Table H-4. compares elements 5 through 10 of the TurboIMAGE/V status area with elements 5 through 10 of the TurboIMAGE/XL status area. If an error occurs, this information is returned for all library procedures. If the procedure executes successfully, this information is returned for DBBEGIN, DBCLOSE, DBCONTROL, DBEND, DBINFO, DBLOCK, DBMEMO, and DBOPEN.

Table H-4. Status Area Changes for MPE/iX Applications

Element	TurboIMAGE/V and Compatibility Mode	TurboIMAGE/XL
5	PB-relative address of the caller.	0 (moved to DBU)
6	Bits 7-15: Intrinsic number of called library procedure. Bits 0-3: Zero or access mode in which database is opened.	No change.
7	16-bit address of the database.	First 16 bits of the database address.
8	16-bit address of the data set name or qualifier.	Second 16 bits of database address (data set or qualifier address moved to DBU).
9	Value of the mode parameter.	No change.
10	PB-relative address of the library procedure or the Compatibility Mode switch stub.	0 (moved to DBU)

Moving from MPE/iX to MPE V

If you need to move a TurboIMAGE database from MPE/iX to MPE V, you need to perform the following tasks:

1. If the TurboIMAGE/XL database you stored has taken advantage of dynamic roll-back recovery, check for the presence of a *dbname*00 file. If the *dbname*00 file exists, the database needs to be recovered. Open the database and close it again. The recovery occurs automatically with the call to DBOPEN.
2. Disable ILR, roll-back recovery, and, if applicable, third-party indexing. Set the critical item update (CIUPDATE) option to DISALLOWED.
3. Drop B-tree indices, if any. Use DROPINDEX command of DBUTIL.
4. Check the buffer specifications to ensure that they are compatible with MPE V. TurboIMAGE/XL buffer specifications are set at a large fixed number. Refer to the following discussion of the *BUFFSPECS* parameter of the DBUTIL >>SET command for setting the buffer specifications for a database to be moved to MPE V.
5. If your database is attached to any SQL database environment, detach it from all DBEnvironments. Use the DETACH command of DBUTIL.
6. Store the database with DBSTORE and be sure to specify the TRANSPORT option (see "Major Differences" below).
7. If necessary, re-enable both ILR and roll-back recovery on MPE V after the database is restored.

Major Differences

- A TRANSPORT option has been added to the DBSTORE command. When you are transporting databases from MPE/iX to MPE V, you must use this TRANSPORT option. To do this, supply an INFO parameter as follows:

```
RUN DBSTORE;INFO="TRANSPORT"
```

- Disable ILR before transporting TurboIMAGE/XL databases to MPE V.

NOTE Your TurboIMAGE database may be too large to move to the MPE V system. This is because the expanded file size available on MPE/iX (4 gigabytes as well as availability of jumbo data sets) means data sets can exceed the MPE V file size limit (0.5 gigabytes). If you store a database on MPE/iX using DBSTORE with the TRANSPORT option, you will receive an MPE/iX error when a data set larger than the MPE V limit is encountered.

Buffer Specifications

For databases that will be moved to MPE V, the DBUTIL >>SET command can be used to specify the number of input/output buffers to be allocated by TurboIMAGE/V in the Database Buffer Area Control Block (DBB) depending on the number of users concurrently accessing the database. The syntax of this command follows:

```
>>SET database name [/maint word] { BUFFSPECS=num buffers
(from-users/to-users) [,num buffers(from-users/to-users)] ... }
```

For a complete syntax listing of the DBUTIL >>SET command, refer to Chapter 8.

Parameters

BUFFSPECS is for MPE V compatibility only, because the TurboIMAGE/XL buffer specifications are fixed at a large default value. For databases that will be moved to MPE V, it sets the number of buffers to be allocated by TurboIMAGE/V in the Database Buffer Area Control Block (DBB). The following parameters are used with the *BUFFSPECS* parameter:

num buffers is the number of buffers for the range of users specified between the parentheses that follow. The minimum number of buffers allowed is 5 and the maximum is 255. 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 database, the amount of main memory available, and many other factors.

from-users is the minimum number of concurrent users (access paths) for which the preceding *num buffers* should be allocated. The minimum *from-users* value allowed is 1 and the maximum is 120. The value must be less than the immediately following *to-users* value.

to-users is the maximum number of concurrent users for which the preceding *num buffers* should be allocated. The minimum *to-users* value allowed is 1 and the maximum is 120. The value must be greater than the immediately preceding *from-users* value.

Discussion

The following BUFFSPECS discussion is for databases that will be moved to MPE V. The buffer specifications are fixed at 1280 on TurboIMAGE/XL.

The *from-users* and *to-users* counts can be the same. For example:

```
SET ORDERS/SELL BUFFSPECS=10(4/4)
```

is valid. The value of *from-users* must be less than or equal to the value of *to-users*. In addition, 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 on MPE V:

```
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 the largest number of search items in any detail data set in the database plus 3, or it is equal to 8, whichever is larger.

If p is the maximum number of search items (the path count), the value of b can be represented as follows:

b = max (p+3,8)

For example, the largest path count for a detail data set in the ORDERS database is 4. (This is the path count for the CUST data set.) Therefore, the value of b for the ORDERS database is shown here:

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 1

```
:RUN DBUTIL.PUB.SYS                 Initiate DBUTIL execution.
     .
     .
     .
>>SET ORDERS BUFFSPECS=5(1/120)     Specify 5 buffers to be allocated for from 1
For database ORDERS                 to 120 users (access paths).
BUFFER SPECIFICATIONS:              DBUTIL confirms the specifications
                                    by listing them.
5(1/120)
>>
```

Example 2

```
:RUN DBUTIL.PUB.SYS
     .
     .
     .
>>SHOW ORDERS BUFFSPECS                    Check the current buffer
                                           specifications.


BUFFER SPECIFICATIONS:
5(1/120)


>>SET ORDERS BUFFSPECS=30(1/30)             Specify the new buffer
```

```
                                                  specifications.
                                                  Here the buffers are increased for
Number of buffers must not decrease for           the first 30 users.  However, the
increased users as below:                         previous buffer setting remains
                                                  the same for the other users.
BUFFER SPECIFICATIONS:
30(1/30),5(31/120)

Buffer specifications remain unchanged
as below:

BUFFER SPECIFICATIONS:
5(1/120)

>>SET ORDERS BUFFSPECS=30(1/30),40(31/120)       Specify larger number of buffers
                                                  for increased number of users.
BUFFER SPECIFICATIONS:                            DBUTIL confirms the specifications
30(1/30),40(31/120)                               by listing them.
>>
```

Index

E

M