Eloquence DBMS Manual
B.06.32

Edition E1202

Legal Notices

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

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

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

Marxmeier Software AG
Besenbruchstrasse 9
42285 Wuppertal
Germany

Printing History

The manual printing date indicates its current edition. The printing date will change when a new edition is printed. Minor changes may be made at reprint without changing the printing date. New editions are complete revisions of the manual.The dates on the title page change only when a new edition or a new update is published.

Manual updates may be issued between editions to correct errors or document product changes. Manuals that are published on the Eloquence website (www.hp-eloquence.com/doc) may be updated more often, please visit this website periodically for the most recent versions. To ensure that you receive the updated or new editions, you should also subscribe to the appropriate product support service.

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

First Edition	Apr 1990	A.01.00
Second Edition	July 1991	A.03.00
Third Edition	February 1992	A.03.10
Fourth Edition	August 1997	A.06.00
Fifth Edition	October 1997	A.06.00
Sixth Edition (E1202)	December 2002	B.06.32

Printed in the Federal Republic of Germany.

Printing History

Table of Contents

Contents

Contents

Contents

Contents

Contents

1

Things to Know Before You Start

Inside This Manual

This manual contains information on using the Eloquence database management system (DBMS). The manual is organized as follows:

Chapter 1 ”Things to Know Before You Start” is an introduction to the use of this manual.

Chapter 2 ”Introduction” provides a prelude to database concepts and the way the Eloquence DBMS is organized.

Chapter 3 ”Database Definition” describes how to define a database.

Chapter 4 ”Database Manipulation” lists and describes the database statements used to access and manipulate database information.

Chapter 5 ”Database Utilities” explains the database utilities used to maintain the database included in A.06 and later.

Chapter 6 ”Example Operations” shows example database operations and sample programs.

Appendix A ”Pack Statements” discusses transferring string and numeric data to and from a string variable by using Pack Statements.

Appendix B ”DBML Syntax” lists the syntax for the database manipulation statements.

Appendix C ”Error Messages” lists Eloquence DBMS error messages and gives a brief explanation after each message.

Appendix D ”Eloquence Library” lists and decribes the functions available in the library which users can integrate into their own programs.

Appendix E ”Obsolete Database Utilities” provides documentation for database utilities which are no longer used by the current Eloquence revision. (Since Eloquence A.05.xx will be in use for some time but did not include online documentation we thought it would be helpful to include the documentation here.)

Conventions

The following conventions are used throughout this manual:

- **Bold type** is used when a new term is introduced.
- **`Computer font`** indicates text to be input exactly as shown or text that is output from the system.
- *Italic type* is used for emphasis and titles of publications. It is also used to indicate parameters that are user defined.
- KEYCAP represents a key on the keyboard.
- shading represents the softkeys displayed on the computer screen.
- … indicates that the previous variable can be repeated.
- [ ] indicates that information inside the brackets is optional. If there are brackets within brackets, the information within the inner bracket may only be specified if the information in the outer bracket is specified. Information may also be stacked in brackets. For example, A or B or neither may be selected when the following is shown:

  $$\begin{bmatrix} A \\ B \end{bmatrix}$$

- { } indicates that one of the choices stacked within the braces must be selected. For example, A or B or C must be selected when the following is shown:

  $$\begin{Bmatrix} A \\ B \\ C \end{Bmatrix}$$

NOTE: Notes contain important information and are set off from the text.

Related User Documentation

Additional information is included in the following manuals:

The *Eloquence Manual* contains detailed information on the commands and statements available in the Eloquence environment.

NOTE: This manual assumes the reader has some familiarity with the Eloquence commands and statements discussed in the *Eloquence Manual*.

2

Introduction

This manual describes the Eloquence database management system (DBMS) software.

What is Eloquence DBMS?

Eloquence DBMS is a set of statements and utilities that operate on the Eloquence database. A **database** is a group of logically related files which contain all the data necessary to satisfy a user's information needs. A database also contains structural information describing how the various data files are related. Relationships that link data within a database allow access to both related data and data across files.

An example database system is shown in the following diagram. It performs many of the same tasks as standard file-processing systems. However, its files have been integrated into a database that is processed by application programs. As shown in the example, accounts receivable, order entry, and purchasing systems perform their usual file-processing function, but they call upon the Eloquence database management system to access the database. More importantly, Eloquence DBMS can process the data as an integrated whole. Since the files have been created by the same system, all the data is compatible. This allows for integrated processing, which is the ability to index across the various files to extract related information.

For example, inventory data can be logically "tied to" several sales orders to represent the relationship between items in the inventory file and the sale of those items in the invoice file. This relationship can provide management with information such as the source of sales by salesman, region, customer and product type.

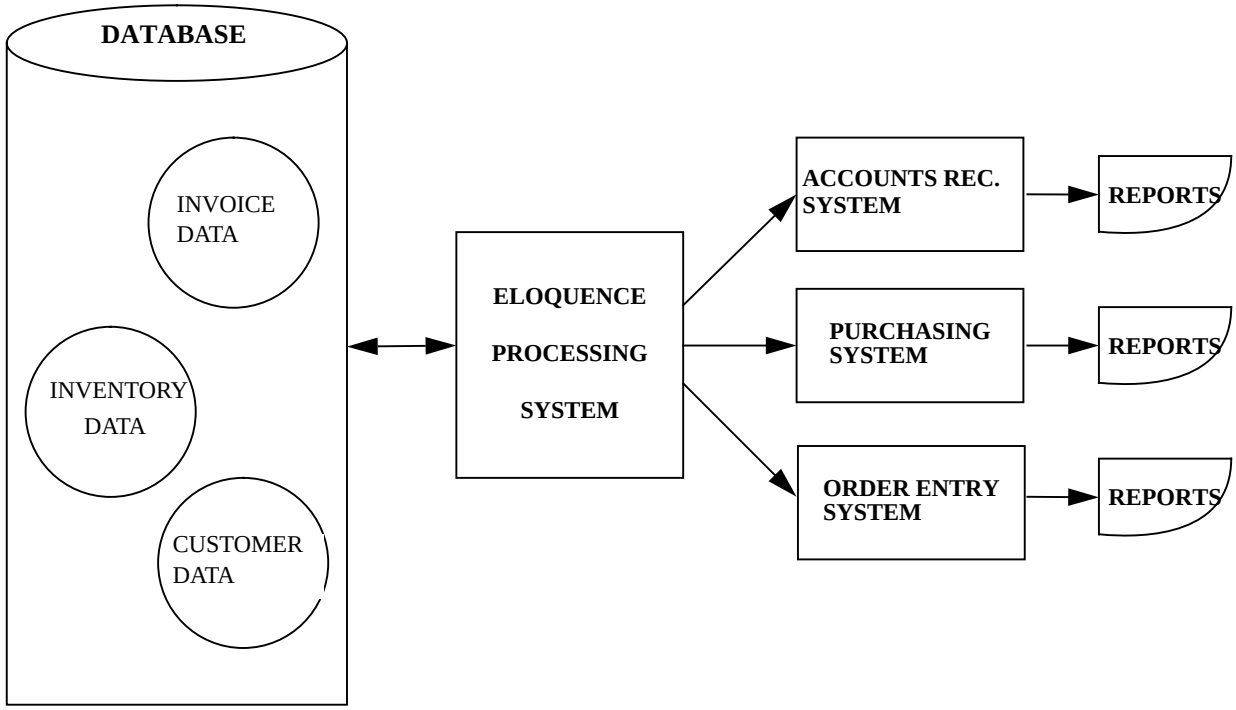


Figure 1 **A Database System**

The Eloquence A.06.00 Database

This chapter describes the operation and architecture of the Eloquence database.

NOTE: The new database is *not* binary compatible to the previous implementation and the database must be transferred by using dbexport/dbimport.

Introduction

When the original Eloquence database was implemented, it was intended as a compatible replacement of the HP260 Image database. With Eloquence release A.03.xx, index operations were added.

Since then, the amount of data stored in Eloquence databases has grown tremendously. While a typical database in 1990 had a size of less then 100 MB, Elquence databases today reach sizes of 4 GB and above. In addition Eloquence databases are used in a network environment.

It became evident, that the previous Eloquence database architecture would not allow to cope with the request for operating on ever increasing amounts of online data. Therefore Eloquence A.06.00 includes a new database system.

The main objectives for the new implementation were:

- Compatibility: The new database is transparent to existing Eloquence programs.
- Improved database security. This includes both, access control and protection against corruption due to program or system failures.
- Better performance for large databases.
- Transaction logging and recovery
- Databases are kept in a database environment which is built of volumes. There are no longer hundreds of database related files.
- A portable and extensible architecture

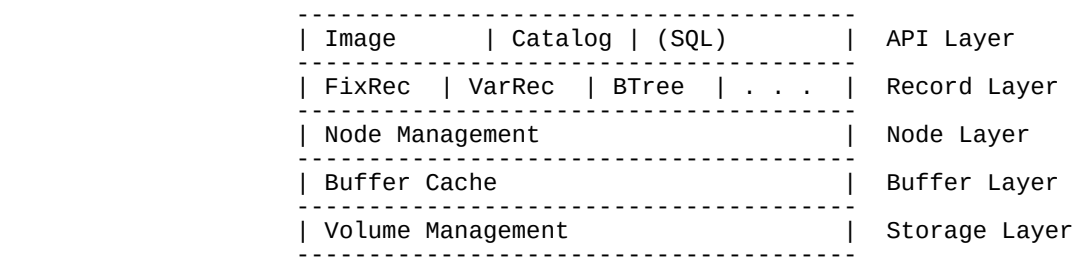
The new Eloquence database can be used as a building brick to efficiently realize relational and hierarchical database strcutures. In the sections below, we provide an overview on the Eloquence database architecture.

Database architecture

In former Eloquence releases, the database was implemented as a shared process. Each Eloquence process contained a common part of the database engine, while some central shared memory was used to coordinate database activities system wide.

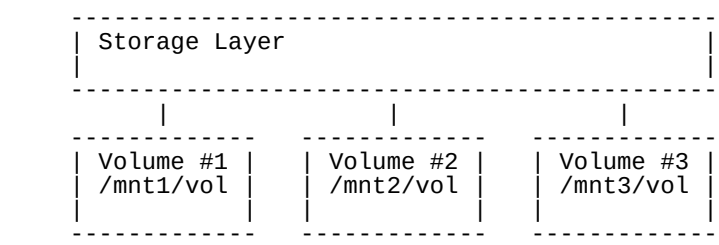
The new Eloquence database now utilizes one process per database environment. This process controls and performs all database operations. This provides better performance, since less system resources are used (for the client processes), less inter-process synchronization is required and the database server process can use dedicated system resources allocated for it.

The new Eloquence database uses a layered architecture. The Diagram below shows the major database layers:



The storage layer

All databases are maintained in a database environment. A database environment consists of a group of related files (called database volumes) which actually contain the data and a configuration file. The storage layer is responsible for volume management and block allocation.



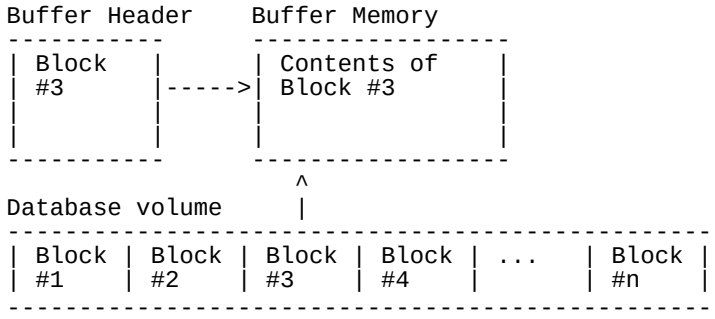
Each database volume contains a volume header, which describes the volume and its properties. It also has a block allocation map, which is used to locate available blocks in this volume. Below that are data blocks which can be allocated in groups for the various database objects.

A database environment can have up to 255 volumes, each volume up to 128 GB (this depends on the underlying operating system). A database volume can either be allocated with a fixed size or can grow by a specified amount when additional space is required. For example, you can have a volume which starts with a size of 100 MB, extends by 16 MB until it reaches a total size of 200 MB.

The Buffer Cache

The buffer cache is a memory area consisting of 8K blocks ("pages"). Each page is associated with a specific location on disc. A group of pages can be linked to a cluster to hold a consecutive disc area. This allows to read or write a group of related disc blocks in one operation.

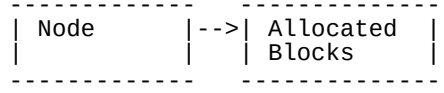
Each buffer cache page consists of a buffer header, holding status and link information for this page and the buffer memory, which holds the data from/for a disk location.



The buffer memory is either allocated on server startup. Modified pages are written back to the disc, when either additional buffer space is required or due to database consistency requirements.

Node Management

A Database Node is a generic database object. Database resources are allocated to database nodes. Each node has an associated list of blocks allocated to it, besides that, it is free to do what it wants. A database node is like a file. You know its dimensions, but you don't know what it contains, unless you have a look inside.



Nodes are allocated dynamically, unlimited in number.

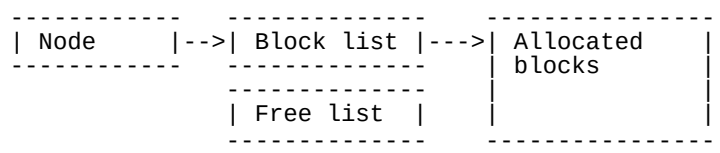
Record Layer

The Record layer implements different logical node types. It is responsible for the maintenance of logical records and is a building brick for the high level APIs (such as Image).

The following node types are currently available:

Fixed records (FixRec)

Fixed records are identified by a constant record address (a 32 bit number) which is used to identify the block address in the volume which actually contains the data. Fixed records are allocated in clusters of up to 64k to minimize disc space overhead.



Fixed records use a list of allocated Blocks to map a record address to a data block (actually a cluster) in a database volume. The Free List is used to keep track of available record numbers.

BTree

This is an index. It contains an association of key values and data address.

SysCat

This is a special node type used internally to manage database meta data. For example, the former ROOT file is now just a collection of records in various system nodes. For example, a node "SYSTABLES" contains a list of all available tables in all databases.

API Layer

The database directory (called catalog) services is an internal API, which is used to maintain metadata on the database. For example, it associates a table name with a node id, and an index with a node id and the table and the indices. In former Eloquence database, database meta data were kept in a ROOT file.

There is no ROOT file anymore and you can have any number of databases in a database environment. All database meta information is now stored in catalog tables in the database environment. Of course, this kind of information is protected against modifications via Eloquence application programs.

If you think of the old ROOT file as a collection of various information specific to a single database, then this will become obvious:

Passwords	Database access passwords
Items	Data Items
Index Items	Index Items
Data sets	Database tables

All these informations are now simply stored in catalog tables inside the database.

For example:

SYSDB	List of databases contained in the database environment
SYSTABLES	List of tables
SYSCOLUMNS	Column (item) definitions
SYSINDICES	Index definitions

The Image API provides the Image functions such as DBPUT or DBGET. The new Image API additionally provides functionality for transaction management.

The new database is not limited to the Image API. While not included with Eloquence, another API such as SQL could be used to access the data simultaneously.

Compatibility

The new Eloquence database will be highly compatible with the previous implementation. From a programming point of view, it does not matter, how a DBPUT is implemented internally, as long as it works as before.

However database utilities such as SCHEMA, DBCREATE or DBUTIL are affected:

For example, the schema processor will no longer create a ROOT file, but it will send the database structural information to the database srever.

The new database is *not* binary compatible to the previous implementation and the database must be transferred by using dbexport/dbimport.

Eloquence version A.06.00 is able to access previous Eloquence databases by the usage of the eloqdb5 server.

Eloquence DBMS Organization

The following sections describe the structure and access methods employed by Eloquence DBMS.

Eloquence DBMS Logical Structure

Eloquence DBMS is organized into three sections: **database definition, database manipulation**, and **database maintenance**.

Database definition is accomplished using an editing program (for example, vi on UNIX, Notepad on Windows) and the schema program. These programs are used in conjunction with the database definition language (DBDL) to define the structure, size and security of a database.

Database access and manipulation is performed using the database manipulation statements. These statements are invoked from Eloquence programs, and serve as an interface between databases and application programs. The manipulation statements handle database access and structural housekeeping.

Database maintenance operations are performed using the database utilities. These utilities provide the capability to create, purge, and erase data sets. You can also report on the structure of the database and, if desired, restructure the database.

Database Organization

There are three basic structures within an Eloquence database: **data items**, **data entries**, and **data sets**.

A data item is the smallest data element. Each data item has a value and is referenced by a data item name. Data items correspond to program variables within an applications program.

Some examples are:

Data Item Name	Data Item Values
PRODUCT-NO	50
	100
	1000
PRODUCT-DESC	Tricycle
	Standard Bicycle
	10-Speed Bicycle

A data entry, or record, is an ordered collection of related data items. All data is transferred to and from a database on a record basis.

For example:

Data-Entry Definition	PRODUCT-NO	PRODUCT-DESC
Data Entry Values	50	Tricycle
	100	Standard Bicycle
	1000	10-Speed Bicycle

A data set is a collection of data entries sharing a common definition. All entries in a data set are stored as a separate file in a directory and are referred to by a data set name. Some examples are shown below:

Data Set Name: PRODUCT

Data Entry Definition:
PRODUCT-NO
PROD-DESC

Entry Record No.		
1	50	Tricycle

2	1000	10-Speed Bicycle
3	100	Standard Bicycle

Types of Data Sets

There are two types of data sets in the Eloquence DBMS: **master data sets** and **detail data sets**. Detail data sets are used to store “line item” information. Master data sets are generally used as indexes to information within detail data sets. For example, the CUSTOMER detail data set shown below contains information about a customer order, such as the customer’s name and the product purchased. A related master set, PRODUCT, is used to point to all orders for a particular product. This association of an item in a master data set is known as a **data path**. The information in a master data set is unique (for example, one product number 100). Up to 16 data paths may be defined for a particular data set. The item used to link the master data set with the detail data set is known as a **search item**. Master data sets have only one search item, but detail data sets may have up to 16 search items.

Data entries contain pointer information used to link related entries. Detail entries contain pointers to other entries containing the same search item value. This linkage of related detail entries is known as a **data chain**. Master entries contain pointers to the beginning and end of data chains, along with the number of entries within the chain. This chain information is automatically maintained by the Eloquence DBMS.

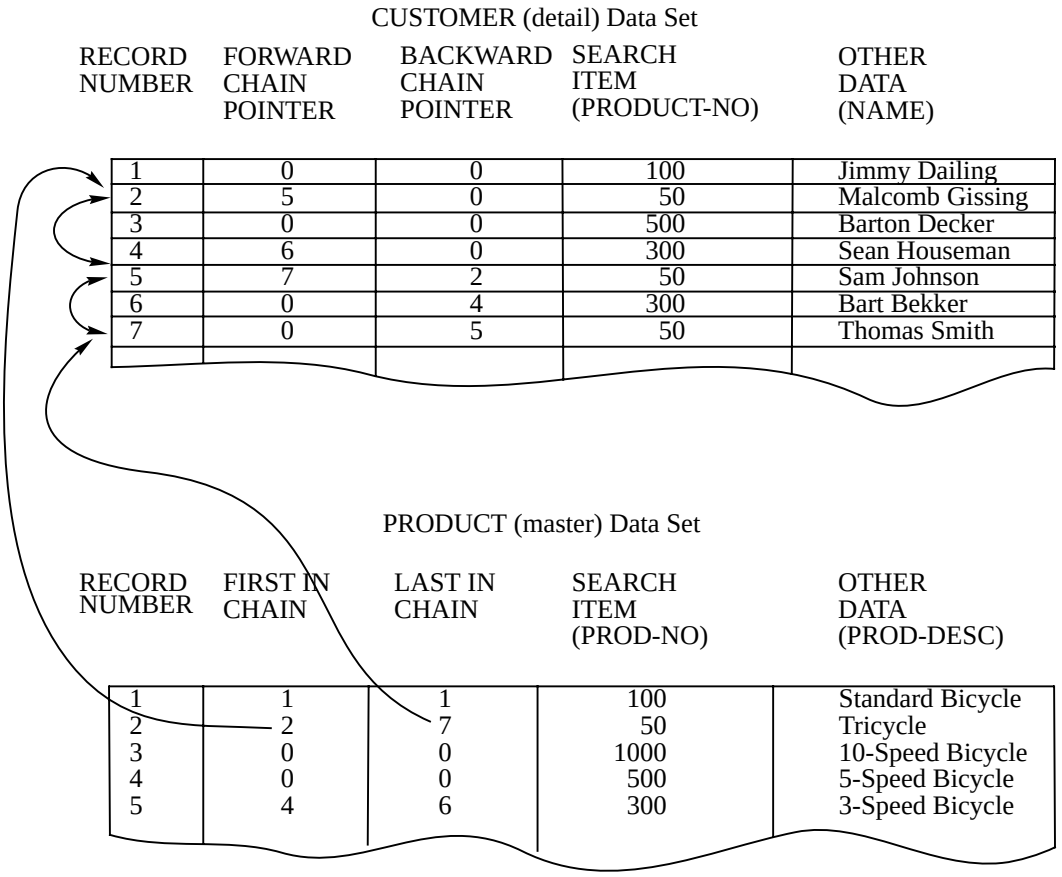


Figure 2 **Data Chain Example**

Data Access

Database access and manipulation is performed using the data manipulation statements. These statements, which are specifically designed to interact with an Eloquence database, are invoked through Eloquence language programs. These statements are structured so that each one suggests its function (for example, DBGET gets data from a data set). All data access is carried out at the data entry level (this is known as the “full record mode”). Data entries may be accessed in one of five modes: **serial, directed, chained, indexed** or **calculated**.

Serial Access

When accessing a data set in serial mode, Eloquence DBMS starts at the most recently accessed record (data entry), called the **current record** and sequentially examines records until the next, non-empty record is located. This record is then transferred to the data buffer and becomes the new current record. Serial access is often used to examine or list all entries in a data set.

The following example shows entries in the PRODUCT master data set. The record numbers are shown to the left of each entry. The arrows to the left of the record number show how entries will be retrieved in serial mode. If the current record is 4, for example, the next record accessed in serial mode will be record number 5.

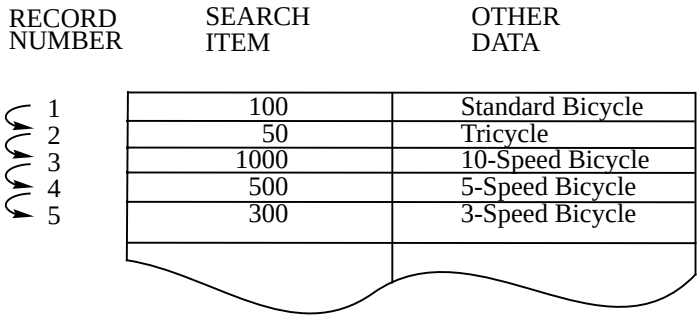


Figure 3 **A Serial Access of the PRODUCT Master Data Set**

Directed Access

A second method of accessing a data entry is directed access. With this method, Eloquence DBMS returns the record specified by a record number supplied by a program. If the specified record is non-empty the record is transferred to the data buffer. If the record is empty a status error is returned. In either case, the current record is set to the record specified. Directed access is used to read entries following a SORT or FIND operation.

The following example shows the retrieval of an entry using directed access. The record number 5, supplied by an application program, instructs Eloquence DBMS to retrieve record 5. Eloquence DBMS then copies the record into the data buffer and resets the current record to 5.

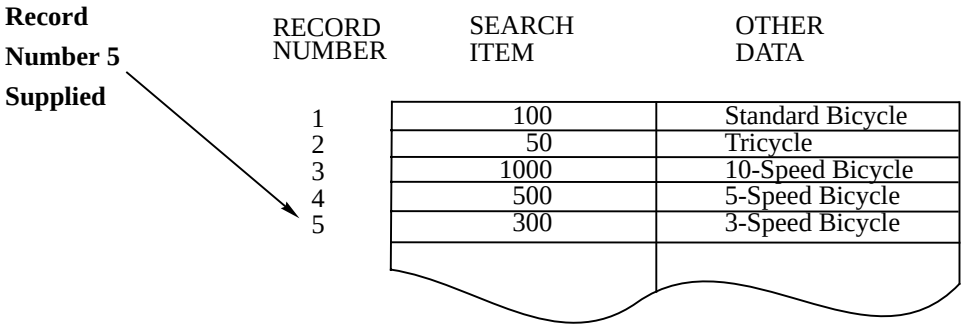


Figure 4 **Directed Access of the PRODUCT Master Data Set**

Chained Access

Chained access is used to retrieve detail data entries with common search item values. Eloquence DBMS supports chained access in a forward direction. Entries along a data chain may be accessed in a reverse direction, however, by using directed access and the status information returned by Eloquence DBMS. Chained access of detail data sets is often used for retrieving information about related events.

The following example shows the retrieval of detail entries using chained access. The corresponding chain pointer information, maintained by Eloquence DBMS, is shown along with the record number for the data set. Eloquence DBMS uses this pointer information to retrieve the next entry along the chain. The arrows to the

left of the record numbers show how entries will be retrieved in chained mode. If the current record is 5, for example, the next record accessed in chained mode will be 7.

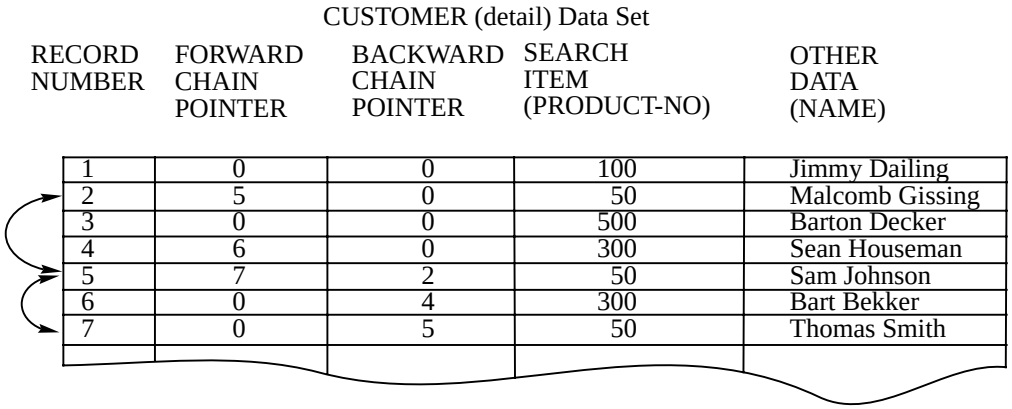


Figure 5 **Chained Access of the CUSTOMER Detail Data Set**

Calculated Access

Calculated access is based on a search item value, and may be used to access master data sets only. This access method involves assigning a search item to a data set, where the assignment of search item to data set is carried out by the ISAM software using index files.

The following example shows the retrieval of an entry using calculated access. The search item value 500, which is supplied by an application program, is used by Eloquence DBMS to locate record number 5. The record is copied into the data buffer and the current record is set to 5.

This example also shows two deleted (blank) records. When a record is deleted from a data file, the space taken by that record still remains in the file. When a new record is added, it takes an available space left by a deleted record. If there are no deleted (blank) records, the new record is added to the end of the file.

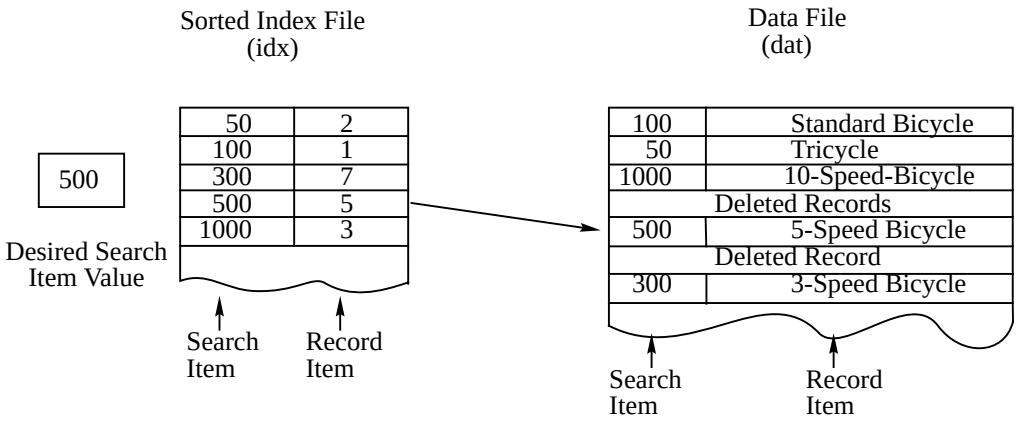


Figure 6 **Calculated Access of the PRODUCT Master Data Set**

Indexed Access

Indexed access is a further method of accessing individual data set entries or a group of data set entries. It is similar to chained access, but there are differences:

- No master set is required, so indexed access can be used for all data set types.
- Combined search-items are possible (e.g. the first four characters of a name plus date of birth). They don’t have to be unique.
- Access via just part of the search-item is possible (e.g., all customers whose names begin with SAM).
- Data can be accessed in sorted form. By default, the index item values are sorted in ASCII sequence, but other collating sequences can be specified at the time the database is created.

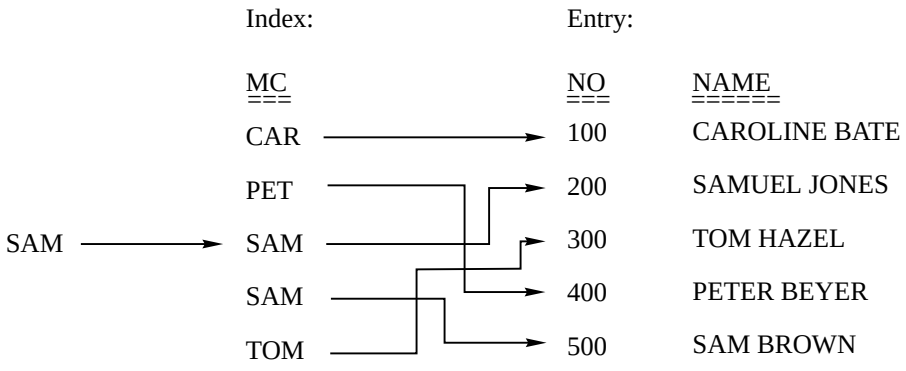


Figure 7 **Indexed Access**

Manual vs. Automatic Master Data Sets

A master data set may be either **manual** or **automatic**. These types of master data sets have the following characteristics:

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

Database security

The Eloquence A.06.00 database has a different security concept than the previous version. The database server maintains its own user list (stored in the server catalog). For each database, there are authorization groups where database specific privileges are assigned to. Users can become a member of those authorization groups and will have all rights granted to this group.

The users "dba" (administrator) and "public" (generic user) are predefined when the database environment is created. The "PASSWORDS" defined in the database schema are converted into authorization groups and the read/write list is converted to the apropriate privileges.

Eloquence has a new statement to deal with this:

DBLOGON(User$, Passwd$)

This will save the provided user id and password for a later connect to the database server. If you omit the DBLOGON statement, the user "public" will be assumed. When you do the first DBOPEN on a database server, the authorization information is submitted and verified by the server.

The Password field in the DBOPEN statement is no longer used, because the access capabilites are defined by the user/group.

The capabilities of a user for a specific database depends on the groups he/she is associated with. So while you cannot delete the predefined users, you can simply remove them from all authorization groups for a particular database and they end up with no access or deny the connect privilege and the server will deny the connection at all.

A sensible administrator would create real user names and associate them with authorization groups. As an additional benefit, you have a single user name/password for all databases (on a single server). Schema associates the public user with all authorization groups but this can easily be changed with the dbutil utility.

The user name is a random name, for example "marc". The password is an arbitrary string, eg. "The secret password". The server will validate the user and password on connction and associate the session with effective privileges.

A user may be a member of up to 8 groups per database. It will get all capabilities associated with those groups. There is no limit on the number of user names and groups.

User privileges

User capabilities which are not database specific are specified by user privileges. The following user privileges are available:

DBA	The user has server administration privileges
CONNECT	The user is allowed to connect the server. This is implied if a user has the **DBA** privilege.
UADMIN	The user is allowed to administrate user accounts

Group privileges

The Eloquence database uses groups (profiles) to manage database specific privileges. When a user is associated with a group, it will gain all capabilities granted to the group.

Group capabilities which are not data set specific are specified by group privileges. The following group privileges are available:

DADMIN	Group members have administration privileges for this database (this is implied for users which have the **DBA** privilege).
DBPRIV	Group members are allowed to assign database specific privileges.

Table privileges

The Eloquence database uses groups to manage database specific privileges. Table (or data set) specific privileges are granted to groups. When a user is associated with a group, it will gain all capabilities granted to the group.

The following table specific privileges are available:

READ	Group members are allowed to read the dataset
WRITE	Group members are allowed to write to the dataset This implies the **READ** privilege.
ERASE	Group members are allowed to erase the dataset.

Predefined users

When a new database environment is created (by dbvolcreate), two users are predefined.

Table 1 **Predefined users**

user id	Description	Default Privileges
dba	default administration user.	DBA, UADMIN

Table 1 **Predefined users**

user id	Description	Default Privileges
public	Default user. This is used when no user id is known when opening a database (missing DBLOGON before opening a database) and is provided for backward compatibility.	CONNECT

NOTE: The default users should *not* be deleted, as they are used when creating a new database to provide a default. If you don't want them, simply remove the user privileges and they are no longer active.

Predefined groups

When a new database catalog is created (by schema), two groups are created automatically in addition to the groups defined by schema:

Table 2

group id	Privileges	Assigned users
dba	GADMIN, DBPRIV	dba
public		public
As defined in schema	As defined in schema	public

Privilege usage

Table 3

Operation	Privileges
Connect to the server	CONNECT or DBA
Manage database user	UADMIN
Manage user privileges	UADMIN
Create database catalog (schema)	DBA
Add database group	DBPRIV
Assign user to database group	DBPRIV

Table 3

Operation	Privileges
Manage group privileges	DBPRIV
Create a database (DBCREATE)	DBA or DADMIN
Purge a database (DBPURGE)	DBA or DADMIN
Erase data sets	DBA, DADMIN or ERASE privilige on specific data set

Getting started with the Eloquence A.06.00 database

This section intends to provide a quick start to the new database. While this document is a bit UNIX centristic, it applies to the Windows NT platform as well.

The examples below assume, that the data base server is running on a system called "server" and is listening on port number 8800. If the server is running on the local system, you could simply omit the host name. If the server is listening on the port mapped to the eloqdb service, you can omit the port number as well. Of course, you could also use a service name instead of the port number.

Create a data base environment

Login to your server system. The server system must have been configured before. The configuration is described more detailed in the Installation and Configuration manual.

- On Windows NT, the data base server must have been registered with the Windows NT Service control manager.
- The TCP service name eloqdb should have been mapped to a port number (unless you intend to use the port number directly).
- Create a new user account and group for use with the data base server (not required on Windows NT)

Create a server configuration

First you need to create a server configuration file. You can either use the template file in /opt/eloquence6/newconfig/config/eloqdb6.cfg or start from the scratch. Since most configuration items are optional and provide a reasonable default it's actually easy to start with an empty file.

```
# eloqdb6.cfg

[Server]
Service=8800
UID = eloq
GID = eloq
LogFile = /tmp/eloqdb6.log

[Volumes]
```

Short description of the configuration settings:

- On Windows NT the UID and GID entries are not supported.
- On UNIX, you should create a separate uid/group for use with the Eloquence data base. All data base volume files are owned by the "data base user". For testing purposes it is

also possible to use your own user/group.

- Service = 8800

  By providing a port number we are not required to create the configuration in /etc/services and we do not desturb any other running eloqdb6 server. Of course, the port number must not already be in use.

- LogFile = /tmp/eloqdb6.log

  All server log messages will be written into the file /tmp/eloqdb6.log. You should use an absolute path here, because the current directory of the server could be a different than you expect. If you omit this entry the server log is written to the syslog (UNIX) or the Event Log (Windows NT) by default.

- [Volumes]

  This section is initially empty and will be filled in by the dbvolcreate and dbvolextend utilities.

Create the server root volume

dbvolcreate is used to create a new volume set. You need a config file with an empty [volumes] section.

```
dbvolcreate -v -c eloqdb6.cfg /path/volume01.vol
```

This creates the root volume /path/volume01.vol with the default size (~2.5 MB) and adds the volume path to the volume section of the config file. The config file could even be empty during volume creation, but it must exist.

Additional dbvolcreate options.

```
-s sz    initial volume size in MB
-e sz    chunks the volume is extended by (MB)
-m sz    max volume size (MB)
```

Create the log volume

Next you need to create a log volume. The server refuses to start without one. The dbvolextend utility is used to create additional volumes.

```
dbvolextend -v -c eloqdb6.cfg -t log /path/volume02.vol
```

This will create the log volume and append it in the volume section of the config file. Additional dbvolextend options.

```
-s sz    initial volume size in MB
-e sz    chunks the volume is extended by (MB)
-m sz    max volume size (MB)
```

The log volume will need at least enough space to hold all commited transactions between two check points. So starting a huge dbimport with a large Checkpoint-Freq setting may cause the log volume to grow and use a huge amount of disk space.

Start the server

Next start the server. Logging into a file with mode "*1" is probably a good idea in the beginning. This example will set the log mode on the commandline but it can also be defined in the config file (see eloqdb6.cfg).

On UNIX:

```
eloqdb6 -d"*1" -c eloqdb6.cfg
```

On Windows NT:

You normally start the data base server from the Windows NT control panel. You could also start the data base server from the commandline by specifying the **`-standalone`** option. However this is only considered a debug tracking option and should not be used regulary.

```
eloqdb6 -standalone -d"*1" -c eloqdb6.cfg
```

If the server comes up, you sucessfully created a db environment. Else have a look at the log file (/tmp/eloqdb6.log in this example).

If you kill the server (please no -9 or you will have stuck IPC resources) or if the server did crash due to a problem, all committed but not yet completed transactions are recovered automatically.

Create a data base

Since we now should have the server up and running, all remaining actions follow the client/server model. So you should now login to your client system.

The new data base server does no longer use ROOT files. Instead it includes a special data base (the system catalog) which is used to hold the data base structure. Actually, there are several catalogs: One which is used to hold server global information and a seperate one for each data base. You can use the dbdumpcat utility to have a look into the server catalogs. The new data base provides its own authorization scheme and list of users.

The server has two predefined users, "dba" and "public" which are created when the database environment is generated. By default the user dba has administrative capabilities (for example it can create a new data base) but is not allowed to access any data. The user "public" by default has no administrative capabilities

but is allowed to access the data base contents. The user "public" is also used as a default user whenever a data base is opened without providing authorization information to the server (please refer to the DBLOGON statement more information).

The dbutil utility can be used to create additional users and to maintain the access rights.

Create a data base

The new schema utility transmits the data base structure from a SCHEMA file to the data base server:

```
schema -u dba -h server -s 8800 db.txt
```

This will create the data base catalog on the specified server. The server name can be omitted if the server is running on the local system. Specifying a service name or port number can be omitted if the server is using the default eloqdb service.

The -u dba is required. It will identify you as user dba to the server. All commandline tools use $LOGNAME as the default user. After the schema succeeded, you should be on known grounds. dbcreate and dbimport should work as expected.

```
dbcreate -u dba server:8800/db
dbimport -vs db.exp -u public server:8800/db
```

Please note, that the -u public argument is required. It will identify you as user public to the server. All commandline tools use $LOGNAME as the default user.

3

Database Definition

Introduction

All Eloquence databases are defined using the **database definition language** (DBDL). Once the database has been defined using the DBDL, an editing program is used to create a text file containing the database definition. This definition, known as a **schema**, is used by the schema program to creste the **database catalog**. The database catalog is stored on the databse server and contains the structural information of the database. Database utilities are used to allocate server resources for the databases (beyound the structural information). This is called database creation. Transposed to the more common file system paradigm, creating the database catalog would be equivalent to the creation of a directory and creating the data sets would be equivalent to creating files in the directory. Once the database has been created, the database is ready to be accessed by either application programs or Eloquence Query. This database definition sequence is summarized below.

1. Define the database using the DBDL.
2. Use a text editor (for example, vi on UNIX or notepad on Windows) to create the schema.
3. Execute the schema program to create the database catalog.
4. Use utilities to create the database.

Database Definition Procedure

A database definition is organized into three sections—password, item, and set. Each section defines a particular part of the database. Additional statements are used to specify the database name, to specify page control, and to designate the end of the database definition. A database definition is organized as shown below.

```
BEGIN DATABASE database-name definition;

DEFAULT LANGUAGE collating sequence definition;

PASSWORDS:password-definition section;

ITEMS:item-definition section;

IITEMS:index-item-definition section;

SETS:set-definition section;

END.
```

Each database definition statement must begin on a new line. Comments, which start with either a # or double carets (<<), will be ignored until end of line.

Database Definition Language

Database-Name Definition

The first part of a database definition is the statement specifying the database name and the root file location. The format of this statement is as follows:

BEGIN DATA BASE *database name*;

The *database name* is from 1 through 6 characters and may consist of uppercase alphabetic characters, the numbers 0 through 9, or the minus sign (–). The name must begin with a letter. This statement must be terminated by a semicolon.

For example:

```
BEGIN DATA BASE EXAMPL;
```

Index item default collating sequence

You can optionally specify a default collating sequence which will control the order in which string elements in an index are to be sorted. You may find this useful for handling the order of national character-sets correctly. If you don't specify a default collating sequence or an index-specific collating sequence, the index order will be according to binary values. The sequence must be installed on your machine and will be stored in the root file. The syntax is as follows:

DEFAULT LANGUAGE *language* [*@modifier*] ;

For example:

```
DEFAULT LANGUAGE german@nofold;
```

Password Definition

The password-definition section follows the database-name definition. The password section begins with the following statement:

PASSWORDS:

It is followed by a list of user-class numbers and their corresponding passwords. Each password definition has the following form:

user-class number password;

Each user-class number, password pair must appear on a new line. The *user-class number* is an integer from 1 through 31 and must be unique within the password section. *Passwords* are from 1 through 8 ASCII characters, excluding semicolons, blanks and tabs. If the same password is assigned to multiple user class-numbers, the lowest-numbered class will be used. Lines containing only a user-class number and a semicolon will be ignored.

For example:

```
PASSWORDS:
31   Clerk;
 5   Gum-ball;
10   SECRET;
15   ;  <<Not currently assigned
22   %-+ ;
```

NOTE: The term "Passwords" is a bit misleading here, as passwords are now converted to database specific access groups to which acess privileges to individual tables (data sets) are assigned. The passwords section is included for backwards compatibility. It is recommended to use the DBUTIL utility to maintain database access privileges.

NOTE: Passwords defined in a schema text file are always upshifted, i.e. only passwords in uppercase letters can be defined.

Item Definition

The item-definition section follows the password section. The item section begins with the following statement:

```
ITEMS:
```

It is followed by a list of all items that are to be used in the database. Up to 1024 data items may be defined in a database. Each item definition has the following form:

item name, [*sub-item count*] *specifier* [(*control no.*)];

The *item name* is from 1 through 15 characters and may consist of uppercase alphabetic characters, the numbers 0 through 9, or the minus sign (–). The name must begin with a letter. Item names must be unique within the item section.

The *sub-item count* is used to define the array length of compound data items (one dimensional item array). An omitted sub-item count or a sub-item count of 1 specifies a simple item. The sub-item count, if specified, must be an integer 1 or greater.

The *specifier* is used to declare the item type. Items may be defined to contain numeric data or ASCII string data. The string designator must be followed by the maximum string length. The string length must be even and cannot exceed 4096 characters. The item specifiers are described below.

Specifier	Type	Description	Range	Item Length *	Max. Sub-item Count **
L or R8	Long(Numeric)	Denotes a 12-digit-floating-point number.	± 9.99999999999E125 through ±1.00000000000E-130	8 bytes	512
S or R4	Short(Numeric)	Denotes a 6-digit-floating-point number.	± 3.40282E+38 through ±1.17549E-38	4 bytes	1024
I or I2	Integer(Numeric)	Denotes a 16-bit integernumber (binary).	+32767 through -32768	2 bytes	2048
D or I4	DoubleInteger(Numeric)	Denotes a 32-bit integernumber.	+2147483647 through -2147483648	4 bytes	1024
X	String	Denotes an ASCII-characterstring. Must be followed by an integer-character count.	Up to 4096 characters	1 byte per character	4096 bytes

* These numbers are for simple items (sub-item count equal to 1). To compute the item length for compound items, multiply the item length shown by the sub-item count.

** The sub-item count is limited by the maximum record length which is 4096 bytes.

The *control number* is used for external item formatting and must be an integer from 0 through 127. This number may be retrieved using an Eloquence statement, but is otherwise ignored. The control number is provided for use by application programs. Query, for example, uses the number to determine the format of numeric data and to prevent Query from modifying sensitive data. Refer to the *Eloquence Query Manual* for more information.

Here is an example item-definition section:

```
ITEMS:
   IN-STOCK, I;
   COUNT, D;
   COST, S;
   TOT-SALE, L;
   DESCRIPTION, X30;
   MONTH, 12X10;  <<12 element array
```

Index Item Definition

The index item definition follows the item section. The index item definition begins with the following statement:

```
INDEX ITEMS:
```

or

```
IITEMS:
```

It is followed by a list of all index items to be used in the database. Each definition has the following format:

iitem name = *item name* [:*length*] [,*item name* ...];

The iitem name is from 1 to 15 characters in length and can consist of uppercase alphabetic characters, the digits 0 to 9, or the minus sign. The name must begin with a letter. Iitem names must be unique within the item and the iitem section. The item name is the name of an already defined item. The referenced item must not be a compound item. If the referenced item is a string item, it is possible to specify a different significant string length to be used. Up to 7 items can be used to form an iitem. The total index length must not exceed 116 bytes.

Example of an iitem definition section:

```
ITEMS:
  PRODUCT-DESC, X30;
  FIRSTNAME, X30;
  LASTNAME,  X30;
  BIRTHDATE, D;   # Format YYMMDD

IITEMS:
  PRODUCT-MC = PRODUCT-DESC:6;
  MATCHCODE = LASTNAME:3, FIRSTNAME:2, BIRTHDATE;
```

PRODUCT-MC is defined as the first 6 characters taken from PRODUCT-DESC. MATCHCODE is defined as the first 3 bytes taken from LASTNAME, the first 2 bytes from FIRSTNAME, and the BIRTHDATE.

NOTE: All index items are stored in separate index tree for the data set. The longer the definition of an index item, the more storage space is required to store the tree. Index items do not require storage space in the entry.

Set Definition

The set-definition section follows the item section. The set section begins with the following statement:

```
SETS:
```

It is followed by a list of data set definitions. The END statement may follow the last set defined in the database. Syntax for this statement is as follows:

```
END.
```

Master Data Set Definition

Manual master data sets are defined using the following format:

{NAME: / N:} *set name*, {MANUAL / M} (*read list/write list*);

{ENTRY: / E:} *item name* [(*path count*)] , *

```
[item name,]
[item name,]
    :
    :
[item name];
```

[{INDEX: / I:} *iitem name* [*collating sequence*]
[,*iitem name* [*collating sequence*] ...];]

[{CAPACITY: / C:} *maximum-entry count*;] **

* If the entry is defined with only one item name, the ENTRY line is terminated with a semicolon instead of a comma.

** Capacity is only available for backward compatibility and should be omitted.

Automatic master data sets are defined using the following format:

{NAME: / N:} *set name*, {AUTOMATIC / A} (*read list/write list*);

{ENTRY: / E:} *item name* [(*path count*)] ;

[{INDEX: / I:} *iitem name* [*collating sequence*]
[,*iitem name* [*collating sequence*] ...];]

[{CAPACITY: / *C*:} *maximum-entry count*;]**

The *set name* refers to the master data set being defined. The name is from 1 through 15 characters, and may consist of uppercase alphabetic characters, the numbers 0 through 9, or the minus sign (–). The first character must be a letter. All set names must be unique within the schema.

The *read list* and *write list* contain user-class numbers separated by commas, and are used to determine which user classes have access to the data set. Specifying a user-class number (an integer from 0 through 31) in the read list allows that user class to have read access to the data set. Specifying a user-class number in the write list allows that user to have read and write access to the data set. The read list may be null, but the write list must contain at least one user-class number.

NOTE:

Specifying a user-class number of 0 for read list or write list means read or write access without a password. This is because it is not possible to define a password for a user-class number of 0.

The *item name* is the name of a data item previously defined in the item-definition section. Each item name must be unique within the data set. A data entry in a manual data set may be defined with up to 1024 item names. A data entry in an automatic data set must contain only one item name. The line containing the last item name in the entry specification must be terminated with a semicolon. The first item appearing in the entry definition is known as the **search item**. The search item must have a sub-item count of 1 (simple item) and must be no longer than 120 bytes.

The *path count* specifies the number of paths to be established to detail data sets. Specifying a path count is optional. If specified, it must be an integer from 0 through 16 and must match the detail references. A manual master set with a path count of 0 is known as a **stand-alone master set** (not associated with a detail data set). For automatic master data sets, the path count is an integer from 1 through 16. Automatic master data sets cannot stand alone.

The *iitem name* is the name of an index item previously defined in the index item definition section. Each iitem must be unique for this set. All items in an iitem definition must be in the set entry list. The last iitem name must be terminated with a semicolon.

The optional *collating sequence* controls the order in which string elements in the index are to be sorted. You can specify an index-specific sequence which you may find useful for handling the order of national character-sets correctly. If you don't specify a collating sequence, the DEFAULT LANGUAGE will be used (if defined; otherwise the binary values). The sequence must be installed on your machine and during creation will be stored in database catalog. You can specify a language and a modifier, for example:

```
...
INDEX:   MATCHCODE /german@nofold;
```

The *maximum-entry count* must be an integer from 1 through 2^{32}–1. This number historically specified the maximum number of entries to be stored in the data set. It is now optional, and is for information purposes only. No space is reserved in advance. If the number of entries stored in the data set increases beyond this number, it increases accordingly. The value of the *maximum-entry count* can be determined by using the dbinfo utility. The count will remain at its highest value unless reset with the dbutil utility.

Examples of data set definitions are shown in chapter 6.

Detail Data Set Definition

Detail data sets are defined using the following format:

{NAME: | N:} *set name*, {DETAIL | 'D} (*read list/write list*);

{ENTRY: | E:} *item name* [(*master-set name*)] , *

```
[item name[(master-set name)],]
[item name[(master-set name)],]
      :
```

```
                :
        [item name];
```

```
[ { INDEX: } iitem name [ collating sequence ]
  { I:     }   [,iitem name [ collating sequence ] ... ];]
```

```
[ { CAPACITY: } maximum-entry count; ]**
  { C:        }
```

* If the entry is defined with only one item name, the ENTRY line is terminated with a semicolon instead of a comma.

** Capacity is only available for backward compatibility and should be omitted.

The *set name* refers to the detail data set being defined. The name is from 1 through 15 characters and may consist of uppercase alphabetic characters, the numbers 0 through 9, or the minus sign (–). The first character must be a letter. All set names must be unique within the schema.

The *read list* and *write list* contain user-class numbers separated by commas and are used to determine which user classes have access to the set. Specifying a user-class number (an integer from 0 through 31) in the read list allows that user class to have read access to the data set. Specifying a user-class number in the write list allows that user to have read and write access to the data set. The read list may be null, but the write list must contain at least one user-class number.

The *item name* is the name of a data item previously defined in the item-definition section. Each item name must appear on a new line and must be unique within the data set. A data entry may be defined with up to 1024 item names. The line containing the last item name in the entry specification must be terminated with a semicolon.

The *master-set name* refers to a previously defined master data set. When a master-set name follows an item name, it indicates that the data item is a search item linking the detail data set to the named master set. Up to 16 data paths may be defined. If no data paths are defined in the detail set, the set is known as a **stand-alone detail set**.

In order for a data path between a master and a detail set to be valid, the search-item type (I, D, S, L, or X) in each data set must be the same. For string items (type X), the string lengths must also be the same. The search-item name in the master data set does not have to match the search-item name in the detail data set.

Only simple items (sub-item count equal to 1 or not specified) can be search items. Also, the search item must not be longer than 120 bytes (an ISAM limitation).

The *iitem name* is the name of an index item previously defined in the index item definition section. Each iitem must be unique for this set. All items in an iitem definition must be in the set entry list. The last iitem name must be terminated with a semicolon.

The *collating sequence* controls the order in which string elements in the index are to be sorted. You can specify an index-specific sequence which you may find useful for handling the order of national character-sets correctly. If you don't specify a collating sequence, the DEFAULT LANGUAGE will be used (if defined; otherwise the binary values). The sequence must be installed on your machine and during creation will be stored in the database catalog. You can specify a language and a modifier, for example:

```
...
MATCHCODE /german@nofold;
```

The *maximum-entry count* must be an integer from 1 through 2^{32}–1. This count specifies the maximum number of entries to be stored in the set. For examples of data set definitions see "Set Definition" on page 47.

Schema Statements

The database definition (schema) can include statements to select schema options and to specify page control. Each statement must appear on a separate line, and may appear anywhere within the schema. These statements, known as schema commands, cannot contain comments.

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

The $TITLE Statement

The $TITLE statement specifies a character string to be printed at the top of each new page of the schema listing. It does not cause a page eject. Syntax for the $TITLE statement is as follows:

`$TITLE` ["*character string*"]

The title specified by the character string overrides any title specified by previous $TITLE or $PAGE statements. If the character string is omitted, no title will be printed until a subsequent $TITLE or $PAGE statement specifies one. Title strings longer than 30 characters are truncated.

The $PAGE Statement

The $PAGE statement causes the schema listing to eject to the top of a new page, unless the NOLIST option has been selected by a previous $CONTROL statement. The $PAGE statement is not listed. Syntax is as follows:

`$PAGE` ["*character string*"]

If a character string is specified, the string replaces the title string specified by a previous $TITLE or $PAGE statement. If no character string is specified, the title string is unchanged. Title strings longer than 30 characters are truncated.

The $CONTROL Statement

The $CONTROL statement selects schema options. Syntax is as follows:

`$CONTROL` *option list*

Options available are as follows:

LIST	Causes the schema program to list each source record from the text file. The listing is printed on the standard printer.
NOLIST	Turns off the LIST option. When an error is found during schema operation, the source record in which the error occurred is listed, followed by an error message.
ROOT	Causes the schema program to connect the database server and create a database catalog if no errors are detected in the schema.
NOROOT	Prevents the schema program from building a database catalog.
ERRORS=*nnn*	Sets the maximum number of allowed errors equal to *nnn*. If this number is exceeded during processing, the schema program terminates immediately. The number must be an integer value from 0 through 999.
LINES=*nnn*	Sets the number of lines to be printed on a page. The number must be an integer. Specifying 0 causes lines to be printed continuously (no page break).
TABLE	Causes the schema processor to print a table containing data set information following the listing. The information includes data set, name, type, number of fields, path number, length, media record length, capacity, number of sectors, and volume labels.
NOTABLE	Suppresses table option.

The $CONTROL options can be placed in any order, but each must be separated by a comma. At least one option must be specified when using $CONTROL.

Options not redefined by a $CONTROL statement default to the following:

```
NOLIST
ROOT
ERRORS = 0
LINES = 0
NOTABLE
```

These default options are equivalent to using the following $CONTROL command:

```
CONTROL NOLIST, ROOT, ERRORS = 0, LINES = 0, NOTABLE
```

Other examples of $CONTROL statements are as follows:

```
$CONTROL NOROOT, LIST
$CONTROL ERRORS = 20
```

NOTE: The listing and root file options can also be specified using the schema command. This is recommended over the $CONTROL statement. By using the schema command options, it is not necessary to edit a text file whenever you want to change an option, as with the $CONTROL statement.

NOTE: $CONTROL options override schema command options.

The Schema Program

The schema program is used to create the database catalog. Once the database has been defined using the DBDL, a text file containing the definition is created using an editing program (for example, vi). This text file is used by the schema program to generate a listing of the database definition and to produce the database catalog. If a database catalog with the same name is already present on the database server, schema will fail.

Using the Schema Program

To run the schema program, execute the following command *from the HP-UX prompt*:

`schema` [options] *filename*

Options:

-help	Give usage (this list)
-u *user*	Set user name to use to logon at the database server. By default, the login id is used.
-p *pswd*	Set password to use to logon at the database server.
-h host	The host name on which the database server is running. This defaults to the local system.
-s service	The service name or port number used to connect the database server. This defaults to the service name eloqdb.
-t	Table of sets to be output (summary of information about the datasets).
-l	Routes the result of the schema analysis to the standard output device (stdout). The standard output device can be the screen, a printer, or a file. When the schema command is executed without the -l option, nothing is sent to the standard output device.
-n	Tells the schema program not to generate the database catalog. This option is useful when you want to perform a syntax check of the schema file.
filename	This variable should be replaced with the name of the text file containing the data definition. The fully-qualified filename is

required.

NOTE: You need administrative capabilities on the database server in order to create a database catalog.

Schema Example

To syntax check the file SAD.txt and create the database catalog on the database server running on the local host, enter the followinf command:

```
schema -l SAD.txt
```

The file SAD.txt is analyzed and the contents of this file are printed on the screen. If the syntax analysis reveals no errors, the database catalog is generated. To contact a database server other than the default one, you must specify the -h or -s option.

Display:

```
B1368A SCHEMA (C) COPYRIGHT MARXMEIER SOFTWARE AG 2002 (A.03.10)

BEGIN DATA BASE    SAD;

PASSWORDS:
                   10      SALESMAN;
                   15      MANAGER;
                    3      SECRTARY;

ITEMS
                   ADDRESS,           2 X30;
                   CITY,                X16;
                   COUNTRY,             X12;
                   DATE,                I;
                   NAME,                X30;
                   OPTION-DESC,         X10;
                   OPTION-PRICE,        L;
                   OPTION-TYPE,         I;
                    ORDER-DATE,          I;
                   ORDER-NO,            X10;
                   PRICE,               L;
                   PRODUCT-NO,          I;
                   PROD-DESC,           X30;
                   REGION,              X6;
                   REGION-DESC,         X30;
                   REGION,TYPE,         I;
                   SALESPERSON,         X4;
                   SHIP-DATE,           I;    <<MUST BE YYMM>>
                   STATE,               X6;
                   ZIP-CODE,            X8

   IITEMS:
                   PRODUCT-MC    = PROD-DESC:6;
                   CUS-MC        = NAME:6;
                   I-SALES-PROD  = SALESPERSON, PRODUCT-NO;

   SETS:                          << Set defintion >>


   NAME:        DATE,  A (3/10,15);
```

```
ENTRY:      DATE (2);

NAME:       ORDER, A (3/10,15);
ENTRY:      ORDER-NO (2);

NAME:       PRODUCT, M (3,10/15);
ENTRY:      PRODUCT-NO (1)
            PROD-DESC;
INDEX:          PRODUCT-MC;

NAME:       LOCATION, M(3,10/15);
ENTRY:      REGION (1),
            REGION-DESC,
            REGION-TYPE;

NAME:       OPTION, D (3/10,15);
ENTRY:      ORDER-NO (ORDER),
            OPTION-DESC,
            OPTION-PRICE,
            OPTION-TYPE;

NAME:       CUSTOMER, DETAIL (3/10,15);
ENTRY:      ORDER-NO (ORDER),
            NAME,
            ADDRESS,
            CITY,
            STATE,
            COUNTRY,
            ZIP-CODE,
            ORDER-DATE (DATE),
            SHIP-DATE (DATE),
            REGION (LOCATION),
            PRODUCT-NO (PRODUCT),
            PRICE,
            SALESPERSON;
 INDEX:     CUS-MC, I-SALES-PROD;

 END.
```

4

Database Manipulation

Introduction

This chapter introduces the Eloquence DBMS manipulation statements. A functional description of each statement is provided as well. Since a working knowledge of the Eloquence language is assumed throughout this chapter, refer to the *Eloquence Manual* when necessary.

The following list summarizes the manipulation statements. For example programs using these statements, refer to page 141 . The syntax conventions used in this chapter are the same as those described in page 11 .

DBLOGON Provides authorization information used when contacting the database server.

DBOPEN Initiates access to a database. Sets up the access mode and user-class number for the specified database.

DBCLOSE Terminates access to a database.

DBGET Reads the data items of a specified entry in a data set.

DBUPDATE Modifies specified item values in an entry. (Search items cannot be modified.)

DBPUT Adds new entries to a data set.

DBDELETE Deletes existing entries from a data set.

DBFIND Locates the first and last entries of a data chain in a detail data set in preparation for access to that chain.

DBINFO Provides structural database information such as data item names, data set names, and field descriptions.

DBBEGIN Initiates a transaction.

DBCOMMIT This statement finishes a transaction.

DBROLLBACK This statement interrupts a transaction and all database modifications since the DBBEGIN of this transactionent.

DBLOCK Locks database records to allow the user exclusive access.

DBUNLOCK Unlocks database records locked with previous DBLOCKs.

DBASE IS Defines the database to be used prior to the IN DATA SET statement.

IN DATA SET	Automatically packs the buffer parameter during DBPUT and DBUPDATE. Automatically unpacks the buffer after DBGET.
PREDICATE	Defines the database records to be locked via DBLOCK.

The DBLOGON Statement

DBLOGON establishes a logon to a database server, not to a certain database. During the DBOPEN and the following database statements, the username and the password is used to examine the permissions on that dataset.

DBLOGON (*User$, Pswd$*)

The parameters are:

User$ A string variable containing the username (not case sensitive). This is used to identify the user and associated privileges for subsequent database operations.

Pswd$ A string variable containing the password for the specified user (case sensitive).

The DBLOGON statement does not perform authentification by itself. It saves the user name and password which is used subsequently when connecting to a database server or opening a database. DBLOGON is typically specified only once and the authorization information is used for all databases. When no DBLOGON statement is executed or an empty user name is passed, the default user "public" is used.

The Eloquence data base provides a its own authorization scheme. A list of users is maintained per data base server. For each data base, there are authorization groups which have specific rights on this particular data base. A user can be a member of up to eight authorization groups.

For Eloquence A.05.xx databases (through the eloqdb5 server) the logon information is ignored and the database password is used.

The DBOPEN Statement

DBOPEN syntax:

DBOPEN (*base name*, *password*, *mode*, *status*)

The parameters are:

base name	A string variable containing the database name preceded by two blank spaces.
password	A string expression containing a left-justified ASCII string.
mode	A numeric expression equal to 1, 3, 8 or 9.
status	An integer array variable that returns status information after DBOPEN is executed. The array must contain at least ten elements in its right-most dimension.

Eloquence A.06.00 uses the client/server model to connect to a database server. Since the data base server can run on a different machine and there could be more than one server on a machine, the DBOPEN syntax accomplish this.

- The data base name uses an extended syntax.
- The Eloquence A.06.00 data base provides a new authorization scheme. The data base password is ignored with the Eloquence A.06.00 data base. It is still used when connecting to a A.05.xx data base (through the eloqdb5 server).

The DBOPEN statement uses a database specification, which consists of three terms:

- The machine, the data base server is running on, this defaults to the local system. Otherwise, the server hostname or "IP address" is required.
- The data base server is listening on a certain port for connections. If this port has not the one mapped to the default service name (eloqdb), the service name or port number must be specified. This port is mapped to a service name in the con figuration file "/etc/services" (on UNIX).
- The database server handles any number of data bases in one data base environment. The data base name is required.
- The A.05.xx database server (eloqdb5) needs an absolute path to locate the data base ROOT file in the file system. An absolute path must be specified.
- All elements besides the data base name can be defined in a **VOLUME** definition, so there should be no impact for existing programs.

The syntax is as below:

[[server][:service]/][Database]

server The name or IP number of the system running the database server. If it is omitted, the local system is assumed.

service The service name or port number of the data base server. If it is omitted, the default service name "eloqdb" is assumed.

database The database name.

- For Eloquence A.05.xx databases, this is the absolute path.
- For Eloquence A.06.xx, this is simply the data base name.

NOTE: The data base name is not case sensitive.

For example:

```
Db$="  sampledb"
DBOPEN(Db$,"",1,S(*))
```

This opens the database sampledb on the local system, using the default service.

Db$=" server/sampledb"

This would connect the default data base server running on the system named "server".

Db$=" server:eloqdb5/path/to/sampledb"

This would connect the data base server, running on the system named "server" using the port associated with the service name "eloqdb5". For Eloquence A.05.xx compatible data bases, it is required to specify an absolute path.

All the connection details could be hidden in a VOLUME definition. If a volume DBVOL is defined as below (for example in your .eloqrc file)

DBVOL="server:eloqdb5/path/to"

then the code below

Db$=" sampledb,DBVOL"

would connect to data base server:eloqdb5/path/to/sampledb.

DBOPEN Modes

A database may be opened in one of four modes. These modes determine the type of operations that can be performed by all users accessing the database.

Mode 1: Modify shared with database locking. Data entries may be read and written within the constraint of the user-class number granted by DBOPEN. Databases opened in mode 1 should be locked (DBLOCK) before data set entries may be added, deleted, or modified. If one user uses DBLOCK, all others also have to lock.

Mode 3: Modify exclusive. DBLOCK and DBUNLOCK statements are not required in this mode. Exclusive access is obtained for reading and/or writing. Although the database may change in content, it does so under your exclusive control. Control is relinquished after executing a DBCLOSE operation. Other requests for access to the database are refused until a DBCLOSE operation is executed.

Mode 8: Read shared. The database is opened for shared read access. Writing to the database is not permitted.

Mode 9: Open for read, allow concurrent update. The database is opened for read access. Other programs may open database in modes 1 and 8.

If successful, DBOPEN replaces the first two characters of the base name string variable, formerly blanks, with an ASCII database number between 00 and 09. This is the internal ID number of the database and should not be altered.

A corrupt database is accessible in mode 8. Status **ERROR −94** is returned if the database is corrupt.

DBOPEN Status Array

A DBOPEN error assigns a non-zero conditional word (CW) to the first element of the status array. A list of all CW values and their meanings appear in page 197 . The following table describes the status array contents after a successful DBOPEN.

Array Element	Value	Description
1	0	CW.
2	0 through 31	User-class number.

Array Element	Value	Description
3	0	
4	0	
5	0	
6	Bits 0 through 11	The DBOPEN identification number (401).
	Bits 12 through 15	The mode value used to open the database.
7	Program line number	
8	0	
9	Mode number	DBOPEN-mode parameter value (same as bits 12 through 15 of element 6).
10	Any value	Reserved.

The DBCLOSE Statement

DBCLOSE terminates access to the specified database.

DBCLOSE (*base name*, *data set*, *mode*, *status*)

The parameters are:

base name	The same string variable used when opening the database.
data set	Any string or numeric expression.
mode	A numeric expression equal to 1 or 3.
status	An integer array variable that returns status information after DBCLOSE is executed. The array must contain at least ten elements in its right-most dimension.

DBCLOSE Modes

Two modes are available when closing a database.

Mode 1: Close the database. The database described is closed, the memory segment assigned by DBOPEN is released, and the first two bytes of the base name parameter are reset to blanks. The data set parameter is ignored.

Mode 3: Data set rewind. The specified data set pointer is reset to the first item in the set.

DBCLOSE Status Array

A DBCLOSE error assigns a non-zero condition word (CW) to the first element of the status array. A list of all CW values and their meanings appears in page 197 . The following table describes the status array contents after a successful DBCLOSE.

Array Element	Value	Description
1	0	CW.
2 through 4	Unchanged	
5	0	

Array Element	Value	Description
6	403	The DBCLOSE identification number.
7	Program line number	
8	0	
9	Mode number	The mode parameter value.
10	Any value	Reserved.

The execution of certain system commands causes an implicit DBCLOSE to be performed. For example, the RUN and SCRATCH C commands perform a mode 1 DBCLOSE on all databases currently opened by the user. This terminates access to the database until a DBOPEN is executed. The following list of statements and keys perform an implicit DBCLOSE in mode 1.

STOP or END

RUN

SCRATCH C

SCRATCH A

CTRL Y *

QUIT

QQUIT

* CTRL Y performs an implicit DBCLOSE *only* in the run-time version of Eloquence (*not* the development version).

The DBGET Statement

DBGET reads the specified data set entry into a string variable.

DBGET (*base name*, *data set*, *mode*, *status*, *list*, *buffer*, *argument*)

The parameters are:

base name	The same string variable used when opening the database.
data set	Either a string expression containing a left-justified data set name or a numeric expression containing a data set number corresponding to the data set's position in the schema definition.
mode	A numeric expression equal to 1, 2, 3, 4, 5, 6, 7, 15, 16.
status	An integer array variable that returns status information after DBGET is executed. The array contains at least ten elements in its right-most dimension.
list	A string expression containing @$\triangle$ or @; or @ ($\triangle$ represents a space). This value means that only the entire entry can be accessed and is referred to as the full record mode.
buffer	A simple string variable in which DBGET returns the specified record entry. The maximum buffer length must equal or exceed the data-set entry length.
argument	Direct access (mode 4)—A numeric expression representing a record number. Calculated access (mode 7)—An expression of the same data type as the master data set's search item.

DBGET Modes

DBGET is used to read entries from the various data sets in a database. The mode parameter determines the type of access requested—serial, directed, chained, or calculated.

Mode 1: Reread. DBGET retrieves the current record. The value of the argument parameter is ignored for this mode.

Mode 2: Serial read, forward. DBGET serially retrieves the record after the current record. The retrieved record becomes the current record. The value of the argument parameter is ignored for this mode.

Mode 3: Serial read backwards. DBGET serially retrieves the record before the current record. The retrieved record becomes the current record. The value of the argument parameter is ignored for this mode.

Mode 4: Directed read. DBGET examines the record located at the address contained in the argument parameter. If the record is not empty, the entry is copied into the buffer. An error condition is returned in the first word of the status array if the record is empty.

Mode 5: Chained read, forward. The first or next entry of the current chain of the specified detail set is read. DBFIND is used to set the current chain pointer for a detail data set. **Indexed read, forward.** DBGET retrieves the first or next entry in index order. DBFIND on index item is used to define current index. DBGET will fail with end-of-chain condition if no more entries with search value as given by DBFIND could be found.

The value of the argument parameter is ignored for this mode.

Mode 6: Chained read, backward. The last or previous entry of the current chain of the specified detail set is retrieved. DBFIND is used to set the current chain pointer for a detail data set. **Indexed read, backward.** DBGET retrieves the last or previous entry in index order. DBFIND on index item is used to define current index. DBGET will fail with end-of-chain condition if no more entries with search value as given by DBFIND could be found.

The value of the argument parameter is ignored for this mode.

Mode 7: Calculated read. This mode is used with master data sets only. The entry with a search item value matching the argument parameter is copied into the buffer. If the search item is numeric, the numeric argument will be converted to the proper type (integer, dinteger, short, or real) before the calculated read is performed.

Mode 15: Next entry in index order. DBGET reads the first or next entry in current index order. DBFIND on index item is used to establish the current index. The value of the argument parameter is ignored for this mode.

Mode 16: Previous entry in index order. DBGET reads the last or previous entry in current index order. DBFIND on index item is used to establish current index. The value of the argument parameter is ignored for this mode.

Eloquence DBMS supports the full record mode of data transfer (list equals @; @). Thus, in all modes, the data items read are those which represent the entire data entry.

DBGET Status Array

A DBGET error assigns a non-zero conditional word (CW) to the first element of the status array. A list of all CW values and their meanings appears in page 197 . The following table describes the status array contents after a successful DBGET.

Array Element	Value	Description
1	0	CW.
2	Buffer Length	Number of words transferred to the buffer.
3	0	
4	Record Number	Integer number of accessed record.
5	0	
6	0 or 1	0 for a detail data set. 1 for a master data set.
7	0	
8	Backward Address	Integer address of the previous record in a chain.
9	0	
10	Forward Address	Integer address of the next record in a chain.

For a detail data set, the forward and backward addresses are always updated relative to the path established by the previous DBFIND applied to the data set.

Array element 8 and 10 will be zero if retrieving in index order.

Record numbers, chain lengths, and forward and backward record addresses fall into the range of 1 through 2^{29}–1.

The DBUPDATE Statement

DBUPDATE modifies item values in a data entry.

DBUPDATE (*base name, data set, mode, status, list, buffer*)

The parameters are:

base name	The same string variable used when opening the database.
data set	Either a string expression containing a left-justified data set name or a numeric expression containing a data set number corresponding to the data set's position in the schema definition.
mode	A numeric expression equal to 1.
status	An integer array variable that returns status information after DBUPDATE is executed. The array must contain at least ten elements in its right-most dimension.
list	A string variable containing @$\triangle$ or @; or @ ($\triangle$ represents a space). This value means that only entire records can be accessed and is referred to as the full record mode.
buffer	A simple string variable containing data that is to replace the current data entry in the specified set.

Since search items cannot be updated, the search item value must match the present buffer value for that item. Before updating a data entry, DBUPDATE must be preceded by a DBGET or DBPUT to establish the current record pointer.

The data item values of the current data entry are replaced using the data supplied. Since data is only transferred in the full record mode, the buffer must contain all the values for the items contained in the data set entry. The search item values in the buffer must match the old values of the current record or the update is not performed.

DBUPDATE Status Array

A DBUPDATE error assigns a non-zero conditional word (CW) to the first element of the status array. A list of all CW values and their meanings appears in page 197 . For a successful DBUPDATE, the status array contents are as described under DBGET, except that the second element represents the number of words transferred from the buffer to the data set.

The DBPUT Statement

DBPUT adds new data entries to a manual or detail set.

DBPUT (*base name, data set, mode, status, list, buffer*)

The parameters are:

base name	The same string variable used when opening the database.
data set	Either a string expression containing a left-justified data set name or a numeric expression containing a data set number corresponding to the data set's position in the schema definition.
mode	A numeric expression equal to 1.
status	An integer array variable that returns status information after DBPUT is executed. The array must contain at least ten elements in its right-most dimension.
list	A string variable containing @$\triangle$ or @; or @ ($\triangle$ represents a space). This value means that only entire records can be accessed and is referred to as the full record mode.
buffer	A simple string variable containing data that is to be placed into the specified set.

The buffer parameter must contain values for all items in the data entry to be added. Eloquence DBMS determines the physical record placement of the entry within the data set.

When the data set is a manual-master data set, Eloquence DBMS verifies that no existing data entry has a search item value identical to the new data entry. If this test fails, an error condition is returned in the status array; otherwise, the new data entry is added.

When the data set parameter is a detail data set, Eloquence DBMS verifies the following:

- The related manual-master data sets have entries with search item values that match the corresponding detail search item values.
- All necessary entries in the related automatic-master data sets are present.

If any of these tests fail, an error condition is returned in the first element of the status array. Otherwise, the data entry is added to the detail data set with Eloquence DBMS performing all linkage maintenance and, when necessary, creating entries in the automatic-master data sets.

Eloquence DBMS determines the physical location of the entry within the data set. If the detail data set is related to one or more master data sets, however, the entry is logically linked to the end of each chain.

Data may not be added directly (DBPUT) to automatic masters. Data is added or deleted automatically as detail data set entries are added or deleted.

NOTE: Even though entries might have been put into the database in sequence, there is no guarantee they will appear in this sequence in the database.

DBPUT Status Array

A DBPUT error assigns a non-zero conditional word (CW) to the first element of the status array. A list of all CW values and their meanings appears in page 197 . For a successful DBPUT, the status array contents are as described under DBGET, except that the second element represents the number of words transferred from the buffer to the data set.

The DBDELETE Statement

DBDELETE deletes existing data entries in a manual-master or detail data set.

DBDELETE (*base name, data set, mode, status*)

The parameters are:

base name	The same string variable used when opening the database.
data set	Either a string expression containing a left-justified data set name or a numeric expression containing a data set number corresponding to the data set's position in the schema definition.
mode	A numeric expression equal to 1.
status	An integer array variable that returns status information after DBDELETE is executed. The array must contain at least ten elements in its right-most dimension.

When deleting entries from master data sets, all pointer information for chains indexed by the entry must indicate that the chains are empty. In other words, there must not be any detail entries on the paths defined by the master which have the same search item values as the master entry to be deleted.

When deleting detail set entries, Eloquence DBMS 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 has been deleted, Eloquence DBMS also deletes the master entry containing the chain heads.

DBDELETE Status Array

A DBDELETE error assigns a non-zero conditional word (CW) to the first element of the status array. A list of all CW values and their meanings appears in page 197 . The following table describes the status array contents after a successful DBDELETE.

Array Element	Value	Description
1	0	CW.

Array Element	Value	Description
2	Record length	Data set record length in words.
3	0	
4	Record Number	Address of deleted record.
5	0	
6	0	
7	0	
8	Backward Address	The unchanged backward address of a chain.
9	0	
10	Forward Address	The unchanged forward address of a chain.

The DBFIND Statement

DBFIND is a dual-purpose statement: DBFIND is used with detail data sets, and locates the head of a chain in a master data set whose search-item value is identified by the argument parameter.

DBFIND is used with indexed access and establishes current index and locates matching value in index table.

DBFIND (*base name, data set, mode, status, item, argument*)

The parameters are:

base name	The string variable used when opening the database.
data set	Either a string variable containing a left-justified data set name or a numeric variable containing a data set number corresponding to the data set's position in the schema definition.
mode	A numeric expression equal to 1, 2, 3, 4, 5.
status	An integer array variable that returns status information after DBFIND is executed. The array must contain at least ten elements in its right-most dimension.
item	A string expression containing a left-justified search item or index item name, or a numeric expression containing a search item or index item number. The search item number represents the relative position in the (index item part) schema definition starting with the number of defined items plus 1; search item or index item numbers are integers ranging from 1 to 1024.
argument	Contains a value for the search item or index item to be used.

DBFIND Modes

DBFIND is used to establish current chain or current index. The mode and item parameters determine the type of access requested - chained or indexed.

Mode 1: Find chain head. If the item refers to a search item (an item linked to a master set in schema definition), DBFIND will set up the current chain and locate the chain head. The specified search item defines the path to which the chain belongs. The *data set* parameter must reference a detail data set. DBFIND uses the argument value to locate the desired chain head in the master data set (using

calculated access). The argument and the search item data types must match. DBFIND converts numeric arguments to the search item numeric data type during execution.

First matching record. If an item refers to an index item, DBFIND will set up the current index and locate the first matching value in the index. If the argument is an empty string, the first record (in index order) will be located. (See discussion below on comparison between modes 1 and 2.)

Mode 2: First matching record. The *item* parameter must refer to an index item. DBFIND will set up the current index and locate the first matching value in the index. If the argument is an empty string, the first record (in index order) will be located.

Mode 3: Last matching record. The *item* parameter must refer to an index item. DBFIND will set up the current index and locate the last matching value in the index. If the argument is an empty string, the last record (in index order) will be located.

Mode 4: Find first matching record with matching regular expression. The *item* parameter must refer to an index item. DBFIND will set up the current index and locate the first matching value in the index. The index item must contain at least one leading string segment.

The given expression must describe the leading string segments exactly. There is no implicit '*' at the end (as in DBFIND Modes 2/3). If you store "AAA " (trailing space), in an entry, you won't find it using a value of "AAA", but you will find it if you use "AAA*" or "AAA?".

The entries will be retrieved using DBGET Mode 5/6 in index order.

Mode 5: Find last record with matching regular expression. Same as Mode 4, but locates the *last* entry.

NOTE: Access time depends on the regular expression given. We do not recommend specifying a character class or a wildcard character at the beginning of the regular expression, as this would result in a serial access to specified data set.

NOTE: Status may return to 0 in the first status array element, although there is no matching entry in the dataset. A subsequent DBGET will return 15 (end-of-chain) in the first status array element.

Chained Access

DBFIND verifies that the item parameter references a search item for the specified detail data set. It then locates the appropriate master data set entry whose search item value (or key) matches the value of the argument parameter. The internal status information relative to the data set parameter is adjusted in anticipation of subsequent chained references to that same data set (DBGET, modes 5, 6.). Note that DBFIND does not retrieve data entries; it simply establishes the current record pointer.

Indexed Access

DBFIND verifies that the *item* parameter references an index item for the specified data set. It then locates the argument value in the appropriate index. If the argument parameter is an empty string, this is simply the first or last record (in index order). If a matching value cannot be found, the record pointer will be located at the position in the index where the requested value would be inserted.

The numeric argument may be given as a string value independent of index item definition. String value must be set up by PACK USING statement. It is valid to give a shorter search value for a string item. DBFIND will locate the first (or last) entry with matching value. If numeric items are truncated they will be ignored in locating index position.

If the first part of the index item referenced by the *item* parameter is numeric then the *argument* parameter may be numeric. DBFIND converts numeric arguments to the item numeric data type during execution. Only the first item is significant in locating index position.

DBFIND/DBGET modes with indexed access

DBFIND mode	DBGET mode	Relation	Comment
2	5	equal	first record with matching index value
2	6		end-of-chain
2	15	equal/greater	
2	16	less	last record before matching index volume
3	5		end-of-chain
3	6	equal	last record with matching index value
3	15	greater	first record after matching index value
3	16	less/equal	

Mode 1 v. Mode 2

DBFIND mode 1 and mode 2 both locate the first matching record. DBFIND mode 1 will return status array elements 6, 8, 10, while DBFIND mode 2 will not (they are zero). DBFIND mode 1 using index item is a convenient way to extend a master/detail database design without changing your programs. (Note that DBGET in index order will not return status elements 6 or 8.) But this has a great disadvantage: it results in the data set entries being read twice, the first time to locate first/last record address and number of records, the second time if you retrieve the data using DBGET.

NOTE: Do not use DBFIND mode 1 on index items if you are expecting a large number of data set entries.

DBFIND Status Array

A DBFIND error assigns a non-zero conditional word (CW) to the first element of the status array. A list of all CW values and their meanings appears in page 197 . The following table describes the status array contents after a successful DBFIND.

Array Element	Value	Description
1	0	CW.
2	0	
3	0	
4	0	
5	0	
6	Chain Length	Integer count of entries in the current chain.
7	0	
8	End of Chain Address	Integer address of the last record in the chain.
9	0	
10	Chain Head Address	Integer address of the first record in the chain.

Status array elements 6, 8, 10 are zero in DBFIND modes 2 and 3.

Regular Expressions

Elements:

[starting delimiter of character class expression

] ending delimiter of character class expression

! negation expression (only as 1st character of character class)

- range expression (only inside a character class)

? any character

***** any string (including the empty string)

numeric character (same as [0-9])

The backslash character (\) loses its special meaning within the delimiters, except in the following combinations:

\b - becomes **backspace**

\t - becomes **tab**

\r - becomes **cr**

\n - becomes **lf**

\f - becomes **ff**

\s - becomes **space**

The above combinations conform to the HP-UX standard, and are extremely practical.

Evaluation An evaluation is only possible with index items, and then only for leading string segments. Index items without leading string segments cannot be accessed.

A regular expression must exactly describe the contents of the leading string segments. There is no implicit “*” at the end (as in DBFIND 2/3). For example, the value “AAA ”(trailing space) does not match the search expression “AAA”.

Examples of regular expressions:

A[BCD] Index value starts with A, followed by either a B, C or D.

BOB?* Index value starts with BOB, followed by at least one character.

The DBINFO Statement

DBINFO provides database structural information from the root file and does not access information within the data sets.

DBINFO (*base name, qualifier, mode, status, buffer*)

The parameters are:

base name	The same string variable used when opening the database.
qualifier	A variable that references a data set, data item, or volume either by name or number (see the following tables).
mode	A numeric expression specifying the type of information to be returned.
status	An integer array variable that returns status information after DBINFO is executed. The array must contain at least ten elements in its right-most dimension.
buffer	A string variable long enough for the requested information to be returned. The contents of the buffer varies according to the mode parameter used.

The various types of information requests are divided into four categories—data sets, data items, data paths, (for example, relationships between data sets), and data set volumes. In all cases, the information supplied is dependent on the access mode used and is restricted by the user-class number established when the database was opened. Any data sets or data paths in the database which are inaccessible to that specific user class are considered to be nonexistent by DBINFO.

For each mode, the information is returned to the buffer as a word string. Some of the string information may actually represent numeric data. The UNPACK statement, described in page 179 , provides a means of properly interpreting these values. The following tables describe the results associated with a successful execution of DBINFO (Conditional word equal to 0).

DBINFO Modes Returning Data-Path Information

Mode	Purpose	Qualifier	Buffer	Contents	Comments
101	Identifies the data-item number for a given data item.	Data item name or number.	Word 1	Data-item number.	Integer.
102	Describes a specific data item.	Data item name or number.	Words 1-8 Word 9 10 11 12 13	Data-item-name. Data-type. Item-word length. Sub-item count. 0 Control number.	Left-justified and filled with blanks. (L,S,I,X) Integer. Integer. Integer. Integer.
103	Identifies all data items in database	(ignored)	Word 1 Word 2... n+1	n Data-item number ...	n = number of data items listed. All words are integers
104	Identifies all data items in a specific data set. The data items are listed in order of occurrence in the data entry.	Data set-name or number.	Word 1 Word 2 ... n+1	n Data-item number ...	n = number of data items listed. All words are integers.

DBINFO Modes Returning Dataset Information

Mode	Purpose	Qualifier	Buffer	Con-tents	Comments
201	Identifies a data-set number for a given data set.	Data set name or number.	Word 1	Data set number.	If positive, entries-can be read only. If negative, entries can be read, writ-ten or modified.
202	Describes a specific data set.	Data set name or number.	Words 1-8 Word 9 10 11 12 13 14 & 15 16 & 17	Data set name. Set type. Entry word-length. 0 0 0 Number of data entries. Data-set capacity.	Left-justified and-filled with blanks. (M,A,D) Words 10-17 are four-byte integers.
-202	Describes a specific data set.	(Same as mode 202.)	(Same as mode 202.)	(Same as mode 202.)	This is the same as mode 202, how-ever it is processed locally (without contacting the database server) and the entry count is always zero.

Mode	Purpose	Qualifier	Buffer	Con-tents	Comments
203	Identifies all accessible datas ets in the data-base. The sets are listed in the order of their occurrence in the schema.	(Ignored)	Word 12...n+1		n number of acces-sible datasets in database. Arranged in data-set number order. If positive, the data set can be read only. If negative, the data set can be read, written or modified. All words are integers.
204	Identifies all accessible datasets which con-taina spe-cific data item.	Data-item name or num-ber.	(Same as mode 203.)	(Same as mode 203.)	(Same as mode 203.)

DBINFO Modes Returning Data-Item Information

Mode	Purpose	Qualifier	Buffer	Contents	Comments
301	Identifies paths defined for a specified data set.	Data-set-name or number.	Word 1 2 3 4 ..... 3n–1 3n 3n+1	n Data-set number. Search item-number. 0 ..... Data-set number. Search item-number. 0	n = Number of paths. Repeat for each path. If qualifier refers to master, set number is for detail. If qualifier refers to detail, set number is for master. Item numbers identify items in detail set. Path designators presented in order of their appearance in schema. All words are integers.
302	Identifies a search item for specified-data set.	Master data-set name or number. OR Detail data-set name or number	Word 1 2 Word 1 2	Search-item-number. 0 Search-item-number. Data-set number.	Search-item number in master set. All words are integers. First search item defined in the detail data set.All words are integers.

DBINFO Modes Returning Index-Item Information

Mode	Purpose	Qualifier	Buffer	Contents	Comments
501	Identifies the index-item name or number.	index-item name or number.	Word 1	Index-item-number.	Integer.

Mode	Purpose	Qualifier	Buffer	Contents	Comments
502	Describes a specific index item.	Index-item name or number.	Word 1-8 Word 9 Word 10 Word 11 Word n2*n+9… Word 2*n+10	Index-item-name. Segment count. Item number. Item length.	Left-justified and filled with blanks. Integer. Integer. Integer. All words are integers.
503	Identifies all index items in database	(ignored)	Word 1 Word 2 ... n+1	n Index-item number	n = number of index items listed. All words are integers
504	Identifies all index items in a specific-data set. The items are listed in order of occurrence in the data set.	Data-set name or number.	Word 1 Word 2 ... n+1	n Index-item number.	n = number of index items listed. All words are integers.

DBINFO Status Array

A DBINFO error assigns a non-zero conditional word (CW) to the first element of the status array. A list of all CW values and their meanings appears in page 197 . The following table describes the status array contents after a successful DBINFO.

Array Element	Value	Description
1	0	CW.

Array Element	Value	Description
2	Buffer Length	Number of words transferred to the buffer.
3	0	
4	Unchanged	
5	0	
6	Bits 0 through 11	The DBINFO identification number 402.
	Bits 12 through 15	The mode value used to open the database.
7	Program line number	
8	0	
9	Mode number	The mode parameter value.
10	Any value	Reserved.

The DBEXPLAIN$ function

The DBEXPAIN$ function returns a error description for a given status array or error number.

DBEXPLAIN$({ Status Array / Numeric value })

Examples:

```
Err_msg$=DBEXPLAIN$(S(*))
Err_msg$=DBEXPLAIN$(-803)
```

Transactions

The Eloquence A.06.00 data base provides transactions. Transactions are used to ensure data base integrity. After a DBBEGIN statement, all data base modifications are no longer stored permanently in the data base. A subsequent DBCOMMIT statement is required to make any pending operations permanent in the database.

The DBROLLBACK statement provides a rollback operation, that reverts pending database operations. After a successful (top level) commit, the transaction is guaranteed to be present in the data base, even in case of a server crash. So it can be guaranteed, that either all dependend operations are saved entirely or no modifications are done.

Pending data base changes are neither visible to other users nor can they be changed concurrently. All pending data base records are locked automatically by the data base server and any attempt to modify them will cause the concurrent task to become paused.

Transactions can be nested. The DBCOMMIT or DBROLLBACK statements usually operate on the last (sub-) transaction. Data base modifications are not stored permanently in the data base, until a top level DBCOMMIT is executed. The DBROLLBACK statement can be used to undo all pending modifications until a specific checkpoint.

Please note, that the transaction handling statements do not operate on a particular data base. Instead they operate on all data bases at once. A pending commit or rollback is even performed after closing the data base. In case a data base server connection is lost (for example, because the server has been killed), all pending modifications on all data bases are automatically reverted.

Each transaction gets a unique is assigned with the DBBEGIN statement and a name of this transaction can be defined to address this transaction later on.

The DBBEGIN Statement

The DBBEGIN statement begins a new (sub-) transaction. When this is the first transaction, it is called top level transaction. No modifications are permanently saved in the Eloquence database until the top level transaction is committed. A subsequent DBBEGIN begins a new subtransaction, which can be controlled separately with the DBCOMMIT and DBROLLBACK statements.

Up to 20 transactions can be nested. Each DBBEGIN statement returns a unique (process specific) transaction id, which can be used with the DBROLLBACK statement to revert all modifications until this state.

DBBEGIN (*comment, mode, status(*)*)

The parameters are:

comment A string variable containing a comment which can be used to give this transaction a name. The name is optional and can be empty.

mode A numeric expression equal to 1.

status An integer array variable that returns status information after DBOPEN is executed. The array must contain at least ten elements in its right-most dimension.

Array Element	Value	Description
1	0	CW.
2	0	Transaction ID
3	0	Transaction Level

The DBCOMMIT Statement

The DBCOMMIT statement commits a transaction. If this is a top level transaction, modifications are made permanently in the data base. If a subtransaction is committed, it becomes part of its parent transaction.

DBCOMMIT (*mode, status(*)*)

The parameters are:

mode A numeric expression equal to:

1 commit all transactions

2 commit top level transaction

status An integer array variable that returns status information after DBCOMMIT is executed. The array must contain at least ten elements in its right-most dimension.

Array Element	Value	Description
1	0	CW.

The DBROLLBACK Statement

The DBROLLBACK statement is used to undo a pending transaction. If this is a top level transaction, all pending modifications are reverted. If applied to a sub-transaction all modifications including the enclosing DBBEGIN statement are reverted.

DBROLLBACK (*ID*, *mode*, *status(*)*)

The parameters are:

ID Transaction ID, used with mode 2.

mode A numeric expression equal to:

- **1** Rollback current (sub-)transaction
- **2** Rollback given transaction
- **3** Rollback top level transaction

status An integer array variable that returns status information after the DBROLLBACK is executed. The array must contain at least ten elements in its right-most dimension.

In Mode 2, a transaction id must be specified which was obtained from DBBEGIN. This can be used to revert up to a specific checkpoint.

Array Element	Value	Description
1	0	CW.
3	0	Transaction Level

The DBLOCK Statement

DBLOCK locks all or a part of a database and provides either exclusive write access or read access which excludes all write access.

DBLOCK (*base name, qualifier, mode, status*)

The parameters are:

base name The same string variable used when opening the database.

qualifier A variable which references a data set or a string expression that describes the lock to be applied (see table below).

mode A numeric expression defining type of lock (see table below).

status An integer array variable that returns status information after DBLOCK is executed. The array must contain at least ten elements in its right-most dimension.

MODES	QUALIFIER	COMMENTS
1,2,11,12	Ignored.	Database level locks.
3,4,13,14	Data set name or number.	Data Set level locks.
5,6,15,16	String expression lock descriptor.	General-purpose lock entry level.

DBLOCK locks the section requested if no other user currently has a conflicting lock. If the lock cannot be granted immediately and a “wait” (odd) mode is used, the request is placed in a queue to wait for access. Access is granted only after all conflicting locks ahead in the queue have been granted and released. (Although the section requested is not locked but is in queue, no other request which would lock a subsection and hold up the first request is granted. Refer to Database Locking in page 141).

Even mode numbers request a lock without wait; if the lock cannot be granted immediately, no lock is made. The reason for the lock failure is indicated in the condition word of the status array. The request is not queued.

If a program has a lock in effect, additional locks can be made only with even modes (”no-wait” modes). If additional locks are made with odd modes (”wait” modes), the “wait” modes are automatically changed to “no-wait” modes. If these

"no-wait" modes cannot be granted, a status error is returned. Using "no-wait" modes prevents successive lock requests from causing a deadlock error. All databases currently opened are checked.

A write lock is exclusive. No other locks may be made on that section. A read lock may be made on a section that already has a read lock.

Summary of DBLOCK Modes

Table 4 **Type of Access**

Mode	Wait	Write	Read
1	yes	Entire database.	
3	yes	Data set.	
5	yes	Predicate.	
11	yes		Entire database.
13	yes		Data set.
15	yes		Predicate.
2	no	Entire database.	
4	no	Data set.	
6	no	Predicate.	
12	no		Entire database.
14	no		Data set.
16	no		Predicate.

For example, the following statement requests *without wait* a write lock on the data set PARTS:

```
100 DBLOCK (BASE$,"PARTS",4,STATUS(*))
```

However, the following statement requests *with wait* a read lock on the data sets and data entries specified in the string formed by the concatenation of the lock descriptors in P$ and Q$:

```
110 DBLOCK (Base$,P$&Q$,15,Status(*))
```

DBLOCK Status Array

A DBLOCK error assigns a non-zero conditional word (CW) to the first element of the status array. A list of all CW values and their meanings appears in page 197 . The following table describes the status array contents after a successful DBLOCK.

Table 5 **Type of Access**

Array Element	Value	Description
1	Conditional Word	0 for successful lock.
2	Descriptor Number	Number causing failure.
	1	For successful lock in modes 1 through 4.
3	0	If CW=20, database locked.
	1	If CW=20, data set or data entries locked.
4	Reserved	
5	0	
6	Bits 0 through 11	The DBLOCK identification number 409.
	Bits 12 through 15	The mode value used to open the database.
7	Program line number	
8	0	
9	Mode number	The mode parameter value.
10	Any value	Reserved.

The DBUNLOCK Statement

DBUNLOCK release all locks for a database or release a specific lock as specified by the qualifier argument.

DBUNLOCK (*base name, qualifier, mode, status*)

The parameters are:

base name	The same string variable identifying the database name when the database was opened.
qualifier	Defines which specific lock will be released.
mode	A numeric expression equal to 1.
status	An integer array variable that returns status information after DBUNLOCK is executed. The array must contain at least ten elements in its right-most dimension.

DBUNLOCK mode relinquishes locks to all sections of the database previously acquired by DBLOCK. If DBUNLOCK is executed at a time when the user does not have a lock, no error is returned.

The following DBUNLOCK mode values are supported:

Mode	**UNLOCK Operation**
1	Unlock database. All locks for the database are released.
3	Unlock dataset. A lock mode 3/4/13/14 is released. The qualifier argument must match the DBLOCK argument.
5	Unlock predicate. A lock mode 5/6/15/16 is released. The qualifier argument must match the DBLOCK argument.

In addition to the "official" modes above, DBUNLOCK also accepts and translates the following mode values:

Mode 2/11/12 is mapped to 1
Mode 4/13/14 is mapped to 3
Mode 6/15/16 is mapped to 5

This makes it possible to use the same DBLOCK and DBUNLOCK modes.

DBUNLOCK Status Array

A DBUNLOCK error assigns a non-zero conditional word (CW) to the first element of the status array. A list of all CW values and their meanings appears in page 197 . The following table describes the status array contents after a successful DBUNLOCK.

Table 6 **Type of Access**

Array Element	Value	Description
1	Conditional word	0 for successful unlock.
2 through 4	Unchanged	
5	0	
6	Bits 0 through 11	The DBUNLOCK identification number 410.
7	Program line number	
8	0	
9	Mode number	The mode parameter value.
10	Any value	Reserved.

Conflicting Item lock resolution

The LOCK CONFLICTING ITEM configuration directive makes it possible to configure, how predicate locks with conflicting items are resolved.

Former Eloquence revisions rejected a predicate lock with a conflicting item, because this could lead to a situation, where two processes own a lock on an overlapping subset of data. If LOCK CONFLICTING ITEM is set to 1, predicate locks with conflicting items are granted. However any write attempt to data where another process owns a lock will result in a status error -12.

Please refer to the Eloquence configuration section for further information.

Advanced Access Statements

Two advanced access statements are described on the following pages. These statements facilitate the use of the Eloquence programming statements discussed earlier by loading or unloading information during DBGET, DBPUT, or DBUPDATE execution.

The DBASE IS Statement

The DBASE IS statement defines the database to be referenced by subsequent IN DATA SET and WORKFILE IS statements. WORKFILE IS is used in conjunction with the FIND and/or SORT statements.

DBASE IS *base name*

The parameter is:

base name The same string variable identifying the database name when the database was opened.

DBOPEN must be executed prior to executing the DBASE IS statement. The database being referenced is reset either by specifying another DBASE IS statement or by closing the referenced database in mode 1.

The IN DATA SET Statement

Through IN DATA SET, data may be automatically transferred from program variables to the buffer variable prior to DBPUT and DBUPDATE. Data may also be automatically transferred from the buffer to program variables after DBGET.

IN DATA SET *data set* [IN COM] { USE ALL | USE *item list* | DIM ALL }

IN DATA SET *data set* [IN COM] USE REMOTE LISTS *line id list*

IN DATA SET *data set* LIST *item list*

IN DATA SET *data set* LIST *item list* FREE

IN DATA SET *data set* USE STRUCT *Instance name*

IN DATA SET *data set* DEFINE TYPE *type name*

The parameters are:

data set Either a string expression containing a left-justified data set name or a numeric expression containing a data set number corresponding to the data set position in the schema definition.

item list A group of numeric, string, or array variables used to associate data items to Eloquence variables. If there are more items in the schema entry than variables in the list, the extra items are skipped. This can also be done by specifying SKP *n* in the item list, where *n* is the number of items you want to skip, or *nX*, where *n* is the number of bytes you want to skip (see PACKFMT). Note that you have to skip entire items. The variable names must be in the order as defined by the schema definition, and must be separated by commas.

line id list A list of line numbers or line labels separated by commas.

During IN DATA SET execution, the data set must be a member of the database recently referenced through DBASE IS. Once the item list has been established for a data set, the default database may be changed through DBASE IS. The correct data-set/data-base relationship is maintained during DBGET and DBUPDATE operations.

In the USE ALL mode, all program variables are searched for a match against the data set's item names, as stored in the root file. Before a match can be determined the schema names undergo the following conversion:

- All letters except the first one are converted to lowercase.
- All dashes (-) are converted to underscores (_).
- If the schema item is a string, a dollar sign ($) is appended to the name.

NOTE: Item names containing native language characters (for example, Ae and E^) are *not* allowed as variable names; therefore, they are not automatically converted and cannot be accessed.

The converted schema name is then compared against all program variables. In addition to a name match, a match must exist for the following:

- The data item types.
- The dimension types (simple or one-dimensional array).
- In the case of strings, the maximum string length.

If the variable was not previously defined during program execution, it will be dimensioned according to the schema item definition.

If all of these conditions are satisfied, the program variable is automatically updated when a data set entry is read into the buffer by DBGET. Before an entry is added (DBPUT) or modified (DBUPDATE), data is automatically copied from the program variable to the buffer.

DIM ALL, like USE ALL, establishes a relationship between program variables and matching data set item names. If a match is not found, however, the data set's field is not treated as a skip field, as with the USE ALL mode. Instead, a variable is automatically created within the program with attributes (field type and length) and name matching the corresponding data item.

If an item list is specified, the variables in the list must exactly match the type and length of the corresponding fields in the referenced data set. Comparisons are made on a positional basis; that is, the first data set field is compared to the first variable in the list, the second field to the second variable and so on.

The SKP option may be used to skip unwanted fields. To skip two or more fields in a row, an appended integer may be used (SKP2 or 10X). This option is useful where only selected fields of a data set entry need to be accessed. Skipped fields are not modified by DBUPDATE and are assigned a null value by DBPUT. If a particular value is desired in a skipped field, this value can be assigned to the string before the DBPUT or DBUPDATE. Note that assigning a value to the buffer string does not change the value of the program variables specified in the IN DATA SET statement.

USE REMOTE LISTS is a means of referencing item lists that appear in other program lines. The lines referenced in the line id list (either by number or label) contain the actual item list. The item lists are evaluated in the order of the line id list. This option is a method of extending an item list length beyond the maximum program line length of 500 characters.

If the IN COM option is selected, an IN DATA SET statement executed in the main program remains active across all subprograms and functions. IN COM may be used with any of the previously described IN DATA SET options. However, all variables referenced must be explicitly dimensioned in common. This option can only appear in the main program, and not in subprograms or functions.

The FREE option releases the internal relationship between the data set and the program variables. Data is no longer transferred to and from the program variables. FREE is necessary only if the IN COM option is used in the main program and another link is desired. Otherwise FREE is implied with IN DATA SET.

DEFINE TYPE is used with user defined types to define a new type from a data set. The type name must not already been defined.

USE STRUCT is used with user defined types to bind the items in data set to the member variables. The specified variable must have been deimensioned.

For example:

```
DBOPEN(Db$,"",1,S(*))
   ...
   IN DATA SET "CUSTOMER" DEFINE TYPE Tcust
   NEW Cust:Tcust
   IN DATA SET "CUSTOMER" USE STRUCT Cust
   ...
   DBGET(Db$,"CUSTOMER",7,S(*),"@",Buf$,Key$)
   ...
```

Of course, types can also be defined statically in your program:

```
   TYPE Tcust
      DIM No$[6]
      DIM Name$[30]
      ...
   END TYPE
   DIM Cust:Tcust
!
   DBOPEN(Db$,"",1,S(*))
   ...
   IN DATA SET "CUSTOMER" USE STRUCT Cust
   ...
   DBGET(Db$,"CUSTOMER",7,S(*),"@",Buf$,Key$)
   ...
```

NOTE:

Care must be taken when executing DBGET to establish the current record pointer for DBUPDATE. Since IN DATA SET automatically transfers the contents of the buffer to program variables during DBGET, these variables must be updated following the DBGET operations.

The PREDICATE Statement

PREDICATE is provided as an aid in setting up predicate strings. It sets up the qualifier parameter that defines the database entries to be locked via DBLOCK.

PREDICATE *predicate* FROM set_1[,$item_1$][,*relop*,*value*]] [;set_2...[;set_n..]]

The parameters are:

predicate	A string variable returned by PREDICATE and used as the qualifier parameter in DBLOCK.
set$_1$	A string expression specifying the data set to be locked or unlocked.
item$_1$	A string expression specifying the data item within set $_1$ to be locked.
relop	A string expression containing a relational operator: **`= or EQ, >= or GE, <= or LE`**
value	A string or numeric expression giving the value of the item to be locked.
set$_{2...}$	A second set of expressions defining the next lock descriptor.
set$_{n...}$	The n^{th} set of expressions defining the next lock descriptor.

Predicate does no type-checking on the value parameter. The programmer should be careful to match string values with database items or string type and numeric values with database items of numeric type. Any discrepancies will result in a status error from DBLOCK. Blanks within item names and set names are ignored. Where multiple descriptors are specified, the descriptor blocks appear in reverse order in the predicate string.

Each of the following examples defines a predicate to request a lock of all data entries in data set TRANS having a data item value of 100:

```
100 PREDICATE P$ FROM ”TRANS”,”PART-NO”,”=”,”100”
```

```
100 Set$=”TRANS”
110 Item$=”PART-NO”
120 Relop$=”=”
130 Value=100
140 PREDICATE P$ FROM Set$ , Item$ , Relop$ , Value
```

The following statement requests a lock on data sets TRANS and INVENTORY (the @ specifies the entire data set):

```
100 PREDICATE Q$ FROM "TRANS", "@", "INVENTORY", "@"
```

The following statement requests a lock on all values of PART-NO greater than or equal to 100 in every data set where PART-NO occurs:

```
100 PREDICATE Q$ FROM "@", "PART-NO", ">=", 100
```

NOTE:

Numeric values will be stored as REAL or INTEGER, depending on expression format and value. INTEGER/DINTEGER will be converted to INTEGER or REAL (depending on value), SHORT and REAL will be converted to REAL.

5

Database Utilities

This chapter describes the Eloquence A.06.00 database utilities. The documentation of previous database utilities (from former Eloquence) which are no longer used has been moved to Chapter 6. The documentation is kept so that this manual is usable with previous Eloquence revisions.

Introduction

The Eloquence DBMS utilities create, initialize, and purge database files and perform various maintenance operations. The utilities consist of Eloquence commands, in addition to external programs which are executed in the operating system environment. The database utilities are summarized below:

Table 7 **Type of Access**

Eloquence Statements	Programs which can be executed from the OS Env.	Description
	dbvolcreate	Creates the database environment.
	dbvolextend	Extends the database environment.
	dbvolchange	Modifies database volume parameters.
	dblogreset	Reset the log volume(s) to minimum size.
	dbutil	Database Maintenance program. This utility is used to maintain Eloquence database security.
DBCREATE	dbcreate	Create data sets and indices from database structure.
DBERASE	dberase	Erases data set entries from all or selected data sets.
DBPURGE	dbpurge	Purges either specific data set files or the entire database, including the root file and all data sets.
	dbexport	Copies data entries from all or selected data sets to ASCII files. Database structural information is *not* saved.
	dbimport	Copies data entries from ASCII files into data sets of a database.

The dbvolcreate utility

Introduction

The Eloquence databases are stored in a database environment. This environment consists of database volume files and a server configuration. This database environment must be created before the Eloquence database server can be started.

The **dbvolcreate** statement creates the main database volume and initializes the system catalog.

dbvolcreate [options] volume_file_name

Arguments:

-v	Specifying the -v option will cause dbvolcreate to output additional information during processing.
-d flags	Set debug flags. This is used to debug dbvolcreate and is normally not used.
-c cfg	Specifies the server configuration file name.
-s size	Initial size (in MB) of the volume to be created. The default size is 2.5 MB (which is also the minimum size).
-e size	Extension size (in MB). When the volume is getting full it will be extended of *size* MB. If *size* is 0, the volume will not be extended automatically by the server. The default size is 1 MB.
-m size	Maximum volume size (in MB).

volume_file_name The volume will be created with this name.

The dbvolextend utility

Introduction

The Eloquence databases are stored in a database environment. This environment consists of database volume files and a server configuration. This database environment must be created before the Eloquence database server can be started.The database environment is initialized with the **dbvolcreate** command.

The **dbvolextend** statement is used to extend the database environment with additional volume files. The most common usage of dbvolextend is to create the mandantory log volume for the database environment.

dbvolextend [options] volume_file_name

Arguments:

-v	Specifying the -v option will cause dbvolextend to output additional information during processing.
-d flags	Set debug flags. This is used to debug dbvolextend and is normally not used.
-c cfg	Specifies the server configuration file name.
-t log	Create a data log volume. The default volume type is data volume.
-s size	Initial size (in MB) of the volume to be created.The default size is 2.5 MB (which is also the minimum size)
-e size	Extension size (in MB). When the volume is getting full it will be extended of *size* MB. If *size* is 0, the volume will not be extended automatically by the server. The default size is 1 MB.
-m size	Maximum volume size (in MB).

volume_file_name The volume will be created with this name.

The dbvolchange utility

Introduction

The parameters of a database volume can be modified by the **dbvolchange** statement.

dbvolchange [options] volume_file_name

Arguments:

-v Specifying the -v option will cause dbvolchange to output additional information during processing.

-d flags Set debug flags. This is used to debug dbvolchange and is normally not used.

-c cfg Specifies the server configuration file name.

-e size Extension size (in MB). When the volume is getting full it will be extended of *size* MB. If *size* is 0, the volume will not be extended automatically by the server.

-m size Maximum volume size (in MB).

-f flag Define volume flags. The volume flags define the database behaviour. The following flags are supported:

[no]dsync - When set, this causes the operating system to force all changes related to database volumes to be written to disk. The dsync flags is disabled by default.

[no]lsync - When set, this causes the operating system to force all changes related to database log volumes to be written to disk on every commit operation. The lsync flag is disbled by default.

The dsync and lsync flags improve database consistency in case of a hardware failure or a power loss but have an impact on the runtime performance, because additional disk i/o operations need to be performed.

volume_file_name The name of the volume file.

The dblogreset utility

Introduction

The **dblogreset** utility can be used to reset the log volumes to minimal size. The log volume of a database environment is required to hold all committed transactions between checkpoints in addition to information about transactions in progress. A checkpoint operation will remove the information about committed transactions, because it is guaranteed that the changes are made permanent to the data volume at that time. Although space is freed in the log volume, the size of the logvolume will not shrink. This can be achieved by the **dblogreset** utility. It will check the log volumes for any commited transactions which may be present (eg. due to an unclean server shutdown) and then truncate the log volume.

dblogreset [options] volume_file_name

Arguments:

-v	Specifying the -v option will cause dblogreset to output additional information during processing.
-d flags	Set debug flags. This is used to debug dblogreset and is normally not used.
-c cfg	Specifies the server configuration file name.

When using **dblogreset**, the database server may not be active.

The DBUTIL utility

Introduction

Dbutil is used to maintain database security with the new Eloquence database.

Database changes are defined in a file using the DBUTIL script language. This makes it possible for a software vendor to provide a control script to a customer to perform database changes without manual interaction.

NOTE:

The DBUTIL utility included with Eloquence A.06.00 is different than the one included with Eloquence A.05.xx. It does currently neither support database restructuring nor does it provide a dialog based user interface.

The following actions can be performed:

- ❒ Create or delete users, change user passwords
- ❒ Grant or revoke user properties
- ❒ Create or delete database specific access groups
- ❒ Grant or revoke group specific properties
- ❒ Assign users with access groups
- ❒ Grant or revoke data set specific access rights to an access group

DBUTIL commandline arguments

Synopsis:

```
usage: dbutil [options] file
options:
 -help   = show usage (this list)
 -v      = verbose          (batch mode only)
 -e cnt  = abort processing after encountering cnt errors
 -t tmp  = where temporary files are stored
 -d flg  = set debug flags
```

Arguments:

-v[v] Specifying the -v option will cause dbutil to output a summary of changes after analyzing the control file and some descriptive text during the database restructuring.

Specifying two -v options will cause dbutil to echo the control file to stdout as it is analyzed.

-e cnt Abort processing the control file after encountering the given

	number of syntax or validation errors.
-t tmp	This option makes it possible to specify where temporary files will be created. If the -t argument is not specified, dbutil will allocate temporary files at the default location of the operating system.
-d flags	This is used internally to debug dbutil itself. You should not use this option.
file	The name of a script file to use by dbutil. Specifying the file name **-** will use stdin.

DBUTIL script file syntax

The dbutil script file is a plain text file. The following general rules apply:

- Everything after a hash character (#) is considered a comment and will be ignored.
- dbutil does recognize keywords in either upper or lower case (but not mixed).
- Each statement must be delimited by a semicolon (;)
- The **`EXIT`** statement can be used to stop processing of a script file before the end of file is reached.
- Strings must be enclosed in double quotes. To include a quote character in a string, the quote character must be preceded by a backslash (\) character.
- A statement may span multiple lines.

The syntax description below uses the following conventions:

- All keywords are given in upper case.
- Optional syntax elements are enclosed in brackets.

Specify the database server

`CONNECT "[server][:service]" ;`

The **`CONNECT`** statement specifies, which database server should be connected. When connecting the local server using the default eloqdb service, it can be omitted.

For example:

```
CONNECT "server";
```

This specifies to connect the server on host "server".

```
CONNECT "server:eloqdb";
```

This specifies to connect to the database server on host "server" which is listening on the port associated with service name "eloqdb".

Authorizing to the database server

In order to connect to a Eloquence database server, authorization information must be provided.

LOGON "user" [PASSWORD "password"];

user	The user name.
password	The password associated with the user. The password is case sensitive.

For example:

```
LOGON "dba" PASSWORD "secret";
```

This specifies to connect as user "dba" using the password "secret".

Managing database users

The **UADMIN** privilege is required to maintain database user.

Creating a database user

CREATE USER "user" [PASSWORD "password"];

Create a new database user. When a password is present, it will be associated with the user. The password is case sensitive. Please note, that a user must be granted the connect privilege in order to connect the server.

For example:

```
CREATE USER "mike" PASSWORD "secret":
```

Deleting database user

DROP USER "user" [,"user2" ...];

Remove the specified user from the database server.

For example:

```
DROP USER "mike","marc":
```

This will remove the users "mike" and "marc".

Changing user password

ALTER PASSWORD FOR USER "user" TO "password";

Change the password for the specified user. Passwords are case sensitive

For example:

```
ALTER PASSWORD FOR USER "mike" TO "secret":
```

User privileges

User capabilities which are not database specific are specified by user privileges. The following user privileges are available:

DBA	The user has server administration privileges
CONNECT	The user is allowed to connect the server. This is implied if a user has the **DBA** privilege.
UADMIN	The user is allowed to administrate user accounts

The **UADMIN** privilege is required to maintain database user.

Syntax:

```
GRANT {privilege [,privilege ...]}
TO {PUBLIC | "user" [,"user" ...]} ;

REVOKE {privilege [,privilege ...]}
FROM {PUBLIC | "user" [,"user" ...]} ;
```

Description:

The **GRANT** statement is used to add the specified privileges to the capabilities of the given users.

The **REVOKE** statement is used to remove the specified privileges from the capabilities of the given users.

When **PUBLIC** is specified instead of a user list, the statement applies to all users.

For example:

```
REVOKE CONNECT FROM PUBLIC;
GRANT CONNECT TO "mike","marc";
```

This will disallow all users besides mike and marc to connect the database server.

Setting database context

`DATABASE "db";`

A Eloquence database server can hold more than one database. For database specific statements, the database on which they should operate must be specifid.

For example:

```
DATABASE "db";
```

This switches the database context to database "db".

Managing database groups

The Eloquence database uses groups to manage database specific privileges. When a user is associated with a group, it will gain all capabilities granted to the group.

A database context must be defined before managing database groups. The **DBPRIV** privilege is required to maintain database groups.

Creating a database group

CREATE GROUP "group";

Create a new database access group.

For example:

```
CREATE GROUP "users";
```

Deleting database groups

DROP GROUP "group" [,"group2" ...];

Remove the specified groups from the database.

For example:

```
DROP GROUP "users";
```

This removes the group "users".

Group privileges

The Eloquence database uses groups to manage database specific privileges. When a user is associated with a group, it will gain all capabilities granted to the group.

Group capabilities which are not data set specific are specified by group privileges. The following group privileges are available:

DADMIN Group members have administration privileges for this database (this is implied for users which have the **DBA** privilege).

DBPRIV Group members are allowed to assign database specific privileges.

A database context must be defined before managing database groups. The **DBPRIV** privilege is required to maintain database groups.

Syntax:

```
GRANT {privilege [,privilege ...]}
TO {"group" [,"group" ...]} ;

REVOKE {privilege [,privilege ...]}
FROM {"group" [,"group" ...]} ;
```

Description:

The **GRANT** statement is used to add the specified privileges to the capabilities of the group.

The **REVOKE** statement is used to remove the specified privileges from the capabilities of the given groups.

For example:

```
REVOKE DBPRIV FROM "users";
GRANT DBPRIV,DADMIN TO "dba";
```

Associating users with a group

The Eloquence database uses groups to manage database specific privileges. When a user is associated with a group, it will gain all capabilities granted to the group. A user can be a member of up to 8 groups per database.

A database context must be defined before managing database groups. The **DBPRIV** privilege is required to maintain database groups.

Syntax:

```
GRANT {"group" [,"group" ...]}
TO {PUBLIC | "user" [,"user" ...]} ;

REVOKE {"group" [,"group" ...]}
FROM {PUBLIC | "user" [,"user" ...]} ;
```

Description:

The **GRANT** statement is used to associate the specified users to the given groups.

The **REVOKE** statement is used to disassociate the specified users from the given groups.

When **PUBLIC** is specified instead of a user list, the statement applies to all users.

For example:

```
GRANT "users" TO "mike",marc";
```

This will make the users mike and marc members of the group "users".

Managing Table privileges

The Eloquence database uses groups to manage database specific privileges. Table (or data set) specific privileges are granted to groups. When a user is associated with a group, it will gain all capabilities granted to the group.

The following table specific privileges are available:

READ	Group members are allowed to read the dataset
WRITE	Group members are allowed to write to the dataset This implies the **READ** privilege.
ERASE	Group members are allowed to erase the dataset.

A database context must be defined before managing database groups. The **DBPRIV** privilege is required to maintain database groups.

Syntax:

```
GRANT {ALL PRIVILEGES|privilege [,privilege ...]}
ON {ALL | "set-name" [,"set-name" ...]}
TO "group" [,"group" ...];

REVOKE {ALL PRIVILEGES|privilege [,privilege ...]}
ON {ALL | "set-name" [,"set-name" ...]}
FROM "group" [,"group" ...];
```

Description:

The **GRANT** statement is used to add the specified privileges to the given groups.

The **REVOKE** statement is used to remove the specified privileges from the given groups.

For example:

```
REVOKE ALL PRIVILEGES ON ALL FROM "users";
GRANT WRITE ON "CUSTOMERS","PARTS" TO "users";
GRANT READ ON ALL TO "users";
GRANT ERASE ON "HISTORY" TO "priv";
```

This will provide read access on all data sets to all members of the group "users" and write access to the data sets CUSTOMERS and PARTS. Members of the group "priv" are allowed to erase the data set "HISTORY".

Example script

```
# connect to server

ONNECT "server:8800";
LOGON "dba" PASSWORD "dbat";

# create user mike with an associated password "secret"
# and allow connection to the database server
# this is global to all databases

CREATE USER "mike" PASSWORD "secret";
GRANT CONNECT TO "mike";

# now switch to database "db"

DATABASE "db";

# create group users.
# let members of group users read/write all sets

CREATE GROUP "users";
GRANT WRITE ON ALL TO "users";

# create group priv.
# let members of group priv erase the set "HISTORY"

CREATE GROUP "priv";
GRANT ERASE ON "HISTORY" TO "priv";

# now let mike become member of groups "users" and "priv"

GRANT "users","priv" TO "mike";
```

DBCREATE, DBERASE and DBPURGE commands

Three commands (DBCREATE, DBERASE and DBPURGE) are used to create, erase and purge selected data set files or entire databases. Each command requires exclusive access to the database (meaning the database cannot be open).

DBCREATE

The DBCREATE command creates and initializes the data sets of a database. The schema processor only saves the meta information in the database server catalog, however no resources are allocated.

DBCREATE is used after the schema program has installed the database structure in the database server catalog. DBCREATE is available as a commandline utility or a Eloquence statement.

The following command can be executed from the operating system command-line:

dbcreate [options] *database* [*data set* [*data set*] . . .]]]

NOTE: The user must have administrative capabilities for either the server or the database. When using the eloqdb5 server, the user information is ignored and the maintenance password must be specified.

The DBCREATE statement can be used in a Eloquence program:

DBCREATE root file spec [;*maintenance word*] [,*set list*] [,*return status*]

NOTE: The user must be authorized with the DBLOGON statement before using executing DBCREATE. The user must have administrative capabilities for either the server or the database. When using the eloqdb5 server, the user information is ignored and the maintenance password must be specified.

The dbcreate commandline utility

Syntax of the dbcreate command is as follows:

dbcreate [-u *user*] [-p *password*] [-v] *database* [*data set* [*data set*] . . .]]]

The parameters are:

-u *user*	Specify user id. The user must have administrative capabilities for the server or the database. The user name is obtained by default from the LOGNAME or USER environment variable.
-p *password*	Specify password. When using the eloqdb5 server, this must be the maintenance password of the database.
-v	An optional parameter that displays the processing procedure. -v stands for “verbose”.
database	A string expression identifying the database name (for example, SAD). If not specified, user will be prompted for database name and password.
data set	Name or number which identifies a particular data set. Specifying one or more data sets is optional. If no data sets are specified, all the sets in the database definition file are created. When data set identities are supplied, dbcreate creates only the data sets specified. Data sets may be identified by name or number.

dbcreate Example:

```
dbcreate -v SAD
```

Display:

```
B1368B DBCREATE (C) COPYRIGHT MARXMEIER SOFTWARE AG

Processing data base: SAD

DATA SET              INDEX(ES)
--------------- -- - --------------------
DATE            01 A
ORDER           02 A
PRODUCT         03 M   1
LOCATION        04 M
OPTION          05 D
CUSTOMER        06 D   2
  |               |  |   |
  Dataset Name    |  |   Number of Indexes defined
                  |  |
                  |  Dataset Types
                  |    A= Automatic Master
                  |    M= Manual Master
                  |    D= Detail
                  |
```

```
Dataset Number
```

In this example, all the data sets in the SAD data definition file are created.

The DBCREATE statement

Syntax of the DBCREATE statement is as follows:

DBCREATE *database* [;*maintenance word*] [,*set list*] [,*return status*]

The parameters are:

database A string expression identifying the database name. An optional volume label can be appended to the database name.

maintenance word A string expression identifying a security password. This expression can be from 1 through 16 characters in length. The maintenance password is only used with eloqdb5 servers.

set list A string expression identifying particular data sets. Data sets are specified by either name or number. Set identifiers are separated by commas.

return status A numeric variable in which an error number is returned (refer to page 197). 0 is returned if no error occurs.

When a set list is supplied, DBCREATE creates only the sets specified. Sets may be identified by name or number (for example: **`"1,2,CUSTOMER"`**). If no set list is supplied, DBCREATE attempts to create all data sets of the database.

When executed from the keyboard without a return status parameter, certain errors may be reported by DBCREATE without terminating execution (see page 197 for a description of these non-fatal errors). However, when DBCREATE is executed from a program or when it is executed from the keyboard and a return variable is used, the first error encountered terminates execution of the command. When the return status variable is used, the return variable contains the error number (or 0 if no errors are encountered), but no error message is displayed.

DBERASE

The DBERASE command erases all entries in data set files. All associated path information in related data sets is also erased. This command is often used prior to reloading data entries. DBERASE is available as a commandline utility or a Eloquence statement.

The following command can be executed from the operating system commandline:

dberase [-u *user*] [-p *password*] [-v] *database* [*data set* [*data set*] . . .]]]

NOTE: The user must have administrative capabilities for either the server or the database or must have the erase privilege for a dataset. When using the eloqdb5 server, the user information is ignored and the maintenance password must be specified.

The DBERASE statement can be used in a Eloquence program:

DBERASE root file spec [;*maintenance word*] [,*set list*] [,*return status*]

NOTE: The user must be authorized with the DBLOGON statement before using executing DBERASE. The user must have administrative capabilities for either the server or the database or must have the erase privilege for a dataset. When using the eloqdb5 server, the user information is ignored and the maintenance password must be specified.

The dberase commandline utility

Syntax of the dbcreate command is as follows:

dberase [-u *user*] [-p *password*] [-v] *database* [*data set* [*data set*] . . .]]]

The parameters are:

-u *user*	Specify user id. The user must have administrative capabilities for the server or the database. The user name is obtained by default from the LOGNAME or USER environment variable.
-p *password*	Specify password. When using the eloqdb5 server, this must be the maintenance password of the database.
-v	An optional parameter that displays the processing procedure. -v stands for “verbose”.
database	A string expression identifying the database name (for example, SAD). If not specified, user will be prompted for database name and password.
data set	Name or number which identifies a particular data set. Specifying one or more data sets is optional. If no data sets are specified, all the sets in the database definition file are created. When data set identities are supplied, dbcreate creates only the data sets specified. Data sets may be identified by name or number.

dberase Example:

```
dberase -v SAD
```

Disaplay:

```
B1368B DBCREATE (C) COPYRIGHT MARXMEIER SOFTWARE AG

Processing database: SAD

DATA SET               INDEX
---------------- -- -  -----------------
DATA             01 A
ORDER            02 A
PRODUCT          03 M
LOCATION         04 M
OPTION           05 D
CUSTOMER         06 D
  |              |  |
  |              |  |
  Data Set Name  |  |
                 |  |
                 |  Data Set Type
                 |    A = Automatic Master
                 |    M = Manual Master
```

```
|    D = Detail
|
Data Set Number
```

In this example, the contents of all the data sets are erased.

The DBERASE Statement

Syntax of the DBERASE statement is as follows:

DBERASE *database* [;*maintenance word*] [,*set list*] [,*return status*]

The parameters are:

database A string expression identifying the database name. An optional volume label can be appended to the database name.

maintenance word A string expression identifying a security password. This expression can be from 1 through 16 characters in length. The maintenance password is only used with eloqdb5 servers.

set list A string expression identifying particular data sets. Data sets are specified by either name or number. Set identifiers are separated by commas.

return status A numeric variable in which an error number is returned (refer to page 197). 0 is returned if no error occurs.

When executed from the keyboard, DBERASE displays either the set number of the set being erased or the related set number followed by a **P** when path information is being erased.

When a set list is supplied, DBERASE erases only the data entries in the sets specified. Sets may be identified by name or number (for example: "**1,2,CUSTOMER**"). If no set list is supplied, DBERASE attempts to erase all data sets of the database.

When executed from the keyboard without a return-status parameter, certain errors may be reported by DBERASE without terminating execution (see page 197 for a description of these non-fatal errors). However, when DBERASE is executed from a program or when it is executed from the keyboard and a return variable is used, the first error encountered terminates execution of the command. When the return status variable is used, the return variable contains the error number (or 0 if no errors are encountered), but no error message is displayed.

NOTE: Executing a DBERASE on a master data set erases all chain information linking the master set entries with related detail entries. This erased chain information may cause unexpected errors on subsequent accesses of the database.

DBPURGE

The DBPURGE command deletes specified data sets or all data sets and the associated root file. This command is often used prior to recreating the database. DBPURGE is available as a commandline utility or a Eloquence statement.

The following command can be executed from the operating system commandline:

dbpurge [options] *database* [*data set* [*data set*] . . .]]]

NOTE: The user must have administrative capabilities for either the server or the database. When using the eloqdb5 server, the user information is ignored and the maintenance password must be specified.

The DBPURGE statement can be used in a Eloquence program:

DBPURGE root file spec [;*maintenance word*] [,*set list*] [,*return status*]

NOTE: The user must be authorized with the DBLOGON statement before using executing DBCREATE. The user must have administrative capabilities for either the server or the database. When using the eloqdb5 server, the user information is ignored and the maintenance password must be specified.

The dbpurge commandline utility

Syntax of the dbpurge command is as follows:

dbpurge [-u *user*] [-p *password*] [-v] [-f] *database* [*data set* [*data set*] . . .]]]

The parameters are:

-u *user*	Specify user id. The user must have administrative capabilities for the server or the database. The user name is obtained by default from the LOGNAME or USER environment variable.
-p *password*	Specify password. When using the eloqdb5 server, this must be the maintenance password of the database.The maintenance password is only used with eloqdb5 servers.
-v	An optional parameter that displays the processing procedure. -v stands for “verbose”.
-f	Option to force the utility to purge the database, even if deletion of sets fails.
database	A string expression identifying the database name (for example, SAD). If not specified, user will be prompted for database name and password.
data set	Name or number which identifies a particular data set. Specifying one or more data sets is optional. If no data sets are specified, all the sets in the database definition file are created. When data set identities are supplied, dbcreate creates only the data sets specified. Data sets may be identified by name or number.

dbpurge Example:

```
dbpurge -v SAD
```

Display:

```
B1368B DBPURGE (C) COPYRIGHT MARXMEIER SOFTWARE AG
Processing database: SAD

DATA SET
---------------- -- -
DATA             01 A
ORDER            02 A
PRODUCT          03 M
LOCATION         04 M
OPTION           05 D
CUSTOMER         06 D
  *              |  |
  |              |  |
```

```
|                |  |
Data Set Name    |  |
and ROOT File    |  |
                 |  Data Set Type
                 |    A = Automatic Master
                 |    M = Manual Master
                 |    D = Detail
                 |
                Data Set Number
```

In this example, all the data sets and the root file of the SAD database are deleted.

The DBPURGE Statement

Syntax of the DBPURGE statement is as follows:

DBPURGE *database* [;*maintenance word*] [,*set list*] [,*return status*]

The parameters are:

database A string expression identifying the database name. An optional volume label or unit specifier can be appended to the database name.

maintenance word A string expression identifying a security password. This expression can be from 1 through 16 characters in length.The maintenance password is only used with eloqdb5 servers.

set list A string expression identifying a particular data set. Data sets are specified by either name or number. Set identifiers are separated by commas.

return status A numeric variable in which an error number is returned (refer to page 197 . A 0 is returned if no error occurs.

When executed from the keyboard, DBPURGE displays either the number of the set being purged or an ***** when the root file is being purged.

When a set list is supplied, DBPURGE deletes the data set files of the specified sets. Sets may be identified by either name or number (for example, **"1,2,CUSTOMER"**). If no set list is supplied, DBPURGE attempts to purge all data set files of the database, and if successful, attempts to purge the root file.

When executed from the keyboard without a return-status parameter, certain errors may be reported by DBPURGE without terminating execution (see page 197 for a description of these non-fatal errors). However, when DBPURGE is executed from a program or when it is executed from the keyboard and a return variable is used, the first error encountered terminates execution of the command. When the return status variable is used, the return variable contains the error number (or 0 if no errors are encountered), but no error message is displayed.

The dbexport and dbimport Programs

The two program files, dbexport and dbimport, are used to copy the entries in data sets of a database to and from ASCII files. The database structural information is *not* saved. These two program files are useful when restructuring a database.

dbexport

The HP-UX program file dbexport copies data entries from all or selected data sets to ASCII files (see Appendix for format description). There are two export modes - multiple files and single file.

If exporting into multiple files the contents of each data set is written into a separate file. These export files are named *database_name.dataset_number*.exp. This is the default export mode. We recommend using a directory to hold all export files of a database.

If exporting into a single file, you can choose any filename or stdout.

The dbexport command must be executed from the operatind system prompt. Syntax of the command is as follows:

dbexport [options] *database* [*data set* [*data set*] . . .]]]

Options:

-u *user*	Specify user id. The user name is obtained by default from the LOGNAME or USER environment variable.
-p *password*	Specify password. When using the eloqdb5 server, this must be the maintenance password of the database.
-v	Detailed listing of procedures. -v stands for “verbose”.
-o *path*	Enter directory where export (.exp) files should be created. If -o *path* is *not* used, the export files are created in the current directory. This option cannot be used with the *single* file option.
-c	chained export
-a	export automatic sets
-r	create restructure information
-f sep	Specifies a different field separator. The default field separator is comma (’,’). With this option dbexport utility program may

	be used to export data from a Eloquence database to an external application.
-s *file*	output into single file, '-' = stdout
databasename	Enter name of database to be exported.
dataset	Enter name or number of the data set to be exported. If no data set names are specified, all data sets associated with the named database are exported.

Execute the dbexport command as shown:

```
dbexport -p secret -v -o sadexp SAD
```

This command exports the data from all SAD data sets to export files (SAD.03.exp, SAD.04.exp, SAD.05.exp, and SAD.06.exp). SAD data sets 01 and 02 are defined in the root file as automatic data sets and are therefore not exported. The **`-o sadexp`** option specifies that the export files are to be created in the sadexp directory. The **`-p secret`** option specifies the user password.

NOTE:

Data sets of type automatic master are filled automatically by dbimport. These data sets are not exported by dbexport, unless option -a is used with dbexport. However, automatic sets are not imported from exported files, but are built up during the loading of corresponding detail data.

Display:

```
B1368B DBEXPORT (C) COPYRIGHT MARXMEIER SOFTWARE AG 2002 (A.06.00)

Processing database : SAD
Export path         : sadexp

DATA SET              Records   Count
--------------- -- - --------  --------
DATE             01 A       11
ORDER            02 A       11
PRODUCT          03 M        5         5
LOCATION         04 M        1         1
OPTION           05 D       16        16
CUSTOMER         06 D       11        11
  |               |  |        |         |
  Data Set Name   |  |        |          Number of exported entries counted
                  |  |        |
                  |  |        Number of entries in the data set
                  |  |
                  |  Data Set Type
                  |    A = Automatic Master
                  |    M = Manual Master
                  |    D = Detail
                  |
                  Data Set Number
```

NOTE: If a data set contains no entries, no export file will be created for that data set.

dbimport

The HP-UX program file dbimport copies data from export files into the entire database or to selected data sets. Dbimport can be used to restructure an existing Eloquence database, as outlined in page 135 . In such instances, dbimport copies data from export files, created by dbexport, into a database. Dbimport can also be used to import data from export files created by an application program.

NOTE: Files to be imported must have the naming format *databasename.datasetnumber*.exp. Therefore, if you use an application program to dump data to a file that you plan to use dbimport on later, make sure that data is dumped to a file named *databasename.datasetnumber*.exp.

The dbimport command must be executed from the operating system prompt. Syntax of the command is as follows:

dbimport [options] *database* [*data set* [*data set*] . . .]]]

Options:

-u *user*	Specify user id. The user name is obtained by default from the LOGNAME or USER environment variable.
-p *password*	Specify password. When using the eloqdb5 server, this must be the maintenance password of the database.
-v	Detailed listing of procedures. -v stands for “verbose”.
-i *path*	Enter directory where export (.exp) files are situated. If -i *path* is not used, program looks in current directory.
-p *pswd*	Enter database password with write access.
-d	trace item value assignment
-r *file*	restructure database. ‘-’ = no file
-s *file*	import from single file, ‘-’ = stdin
-f sep	Specifies a different field separator. The default field separator is comma (’,’). With this option dbimport utility program may be used to import data into a Eloquence database from an external application.
databasename	Enter name of database to receive the data.
dataset	Enter name or number of the data set to be imported into the

database.

Execute the dbimport command as shown:

```
dbimport -v -i sadexp -p secret SAD
```

This command imports the data from the SAD export files (SAD.03.exp, SAD.04.exp, SAD.05.exp, and SAD.06.exp) into the .idx and .dat files created using dbcreate. SAD data sets 01 and 02 are defined in the root file as automatic data sets and are therefore filled automatically when dbimport is executed.

The **`-i sadexp`** option specifies that the export files are in the sadexp directory. The **`-p secret`** option specifies the user password.

Display:

```
B1368B DBIMPORT (C) COPYRIGHT MARXMEIER SOFTWARE AG 2002 (A.06.00)

Processing database: SAD
Import path        : sadexp

DATA SET              Count
--------------- -- - --------
DATE            01 A
ORDER           02 A
PRODUCT         03 M        5
LOCATION        04 M        1
OPTION          05 D       16
CUSTOMER        06 D       11
  |               |  |        |
  Data Set Name   |  |        Number of imported entries counted
                  |  |
                  |  Data Set Type
                  |    A = Automatic Master
                  |    M = Manual Master
                  |    D = Detail
                  |
                  Data Set Number
```

NOTE: If you attempt to import a file that does not follow the naming conventions *databasename.datasetnumber*.exp, the following message appears: *export file name*: **`No such file or directory`**.

NOTE: If a detail data set contained no entries, no export file will be created for that data set and the following message appears: *export file name*: **`No such file or directory`**.

Database Restructuring with dbimport

If you specify the -r *file* option in the commandline, dbimport will operate in restructure mode. In this mode, dbimport will load data sets/items on a symbolic base. Data sets and items are identified by name rather than number or position in record. Unless you have changed set or item names, or you want to initialize new data sets or items, you don’t need to worry about the database reload process.

dbimport restructure file syntax

```
IMPORT SET data set [ = data set]
{
   item spec = item spec;
   item spec = :NULL;
   item spec = :CONST constant;
   ...
}
```

data set data set name or number

item spec item name or number. If data item is defined with a subitem count (array) you may handle item elements individually. For example:

ITEM	this will specify all elements
ITEM [1]	this will specify first element
ITEM [1-3]	this will specify elements 1 to 3

:NULL initialize item with default value. This is zero for any numeric item type or spaces for strings. This is the default for new items.

:CONST initialize item with constant value. Only integer or string constants may be specified here. So you have to specify a LONG or SHORT constant as a string constant (eg. “3.1415”).

All characters past ’#’ (unless in a quoted string) will be ignored until end-of-line.

NOTE: For the database to be loaded, data set/item names are taken from the ROOT file. For the export file(s) the data set/item names are saved by the dbexport utility if run with the -r option. If no data set/item names are saved in the export file(s) (dbexport not run with -r option) you have to specify each item that has a different position in the export set.

Formal syntax specification

```
    T_NUMBER = positive integer constant

    import_spec:
      /* empty */
      | set_spec import_spec

    set_spec:
       IMPORT SET to_set from_set
        set_spec_item_part

     set_spec_item_part:
      /* empty */
      | “{” item_spec_list “}”

     to_set:                          /* target set */
      set name | set number

     from_set:                        /* set in import file */
      /* empty */ | “=” set name | “=” set number

     item_spec_list:
       /* empty */
       | item_spec item_spec_list

     item_spec:                       /* item conversion spec */
      to_spec “=” from_spec “;”

     to_spec:                         /* target item (range) */
      to_item range_spec

      to_item:
       item name | item number

      from_spec:
       T_NUMBER               /* field number in exported record */
       | item_name             /* or exported field name (range) */
         | “:NULL”                  /* NULL (default) value */
         | “:CONST” T_NUMBER        /* integer constant */
         | “:CONST” string_in_quotes /* string constant */


      range_spec:                   /* array element range */
        /*  empty */
        | “[” T_NUMBER “]”
        | “[” T_NUMBER “-” T_NUMBER “]”
```

Example 1:

```
# Import set CUSTOMER from ACCOUNTS
IMPORT SET CUSTOMER = ACCOUNTS

# Fill PRODUCT from ORDER
IMPORT SET PRODUCT = ORDER
{
   PRODUCT-NO = ORDER-NO;
   PROD-DESC  = :CONST “* Unknown *”;
   # all other default
}
```

Example 2:

```
New database layout ...

    ITEMS:

      ORDER-NO,      X8;
      PRODUCT-NO,    X6;
      QUANTITY,      I;
      SHIPMENT-DATE, X4;
      QTY-AVAIL,     I;
      ARRAY,         4I;

      SETS:

   N: ORDER-DETAIL D(/0);
   E: ORDER-NO,
      PRODUCT-NO,
      QUANTITY,
      SHIPMENT-DATE,
      QTY-AVAIL,
      ARRAY;
```

```
Old database layout ...

      ITEMS:

       ORDER-NO,      X8;
       PRODUCT-NO,    X6;
       QUANTITY,      I;
       SHIPMENT-DATE, X4;
       ARRAY,         3I;

         SETS:

     N: PRODUCTION D (/0);
       E: ORDER-NO,
          PRODUCT-NO,
          QUANTITY,
          SHIPMENT-DATE,
          ARRAY;
```

Figure 8 **Example of new layout**

Database reload process…

You want to load your new data set ORDER-DETAIL from your old data set PRODUCTION. All data items should be taken from the the old data set with the following exceptions:

- The new QTY-AVAIL should be filled from the old QUANTITY item.
- The new QUANTITY item should be filled with zero.
- The new SHIPMENT-DATE should be set to a constant value.
- The new ARRAY item should be shifted.

Your import specification file should contain…

```
IMPORT SET ORDER-DETAIL = PRODUCTION
{
```

```
   QTY-AVAIL = QUANTITY;
   QUANTITY  = :NULL;
   SHIPMENT-DATE = :CONST “0000”;
   ARRAY[1-2] = ARRAY[1-2];
   ARRAY[4] = ARRAY[3];
   ARRAY[3] = :CONST 1;
}
```

6

Example Operations

This chapter contains examples of defining, using, and maintaining an Eloquence database. One section describes the design and definition of an example database used to store sales analysis data. The next section contains programs utilizing the database manipulation statements to enter and retrieve data from the database. A third section shows how the database utilities are used for database backup and restructuring.

Database Design

The next figure shows a customer order form for a fictitious company. The company owner has decided to design an Eloquence database to be used to store information from the order form. Once the data has been stored in the database, the owner wishes to generate various sales reports using the order data.

Demonstration Bicycle Company

ORDER FORM

Order Date _____ (MMYY) Order Number _____
Ship Date _____ (MMYY) Region _____
Salesperson _____

Name _____
Address _____
City _____ State_____
Zip Code _____Country_____

Product _____ Price _____

Options

Option	Type	Price
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____

Total: _____

The database structure shown next could be used to store the required data. This structure offers little advantage over using a direct (random) access data file, but does provide a starting point for the database design. Notice that in order to generate a list of all orders for a particular product, all entries in the detail set must be scanned. Also no provision is made in this structure to handle an order which includes more than four options.

In order to provide a common basis for diagramming database designs, pentagons are used to represent manual master data sets, triangles are used to represent automatic master data sets, and trapezoids are used to represent detail data sets. The item names and item types used to define the data entry within the data set are also shown.

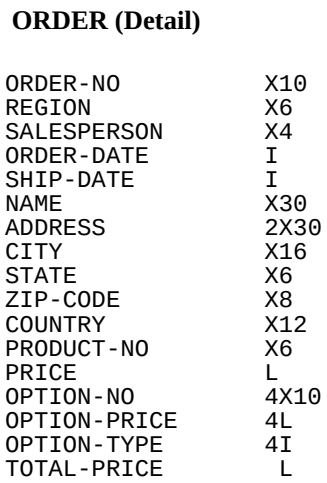


Figure 9 **Possible Database Structure**

The next database structure shows a quick method of generating lists of orders for a particular product number. Using this structure, an application program can perform a DBFIND (see page 59) on the order detail data set to locate the chain head for a particular product number (PRODUCT-NO). The program can then perform successive chained DBGETs (see page 59) to retrieve the required orders. Only the desired orders are accessed, thus reducing the time required to generate the list when a large number of orders are stored in the database.

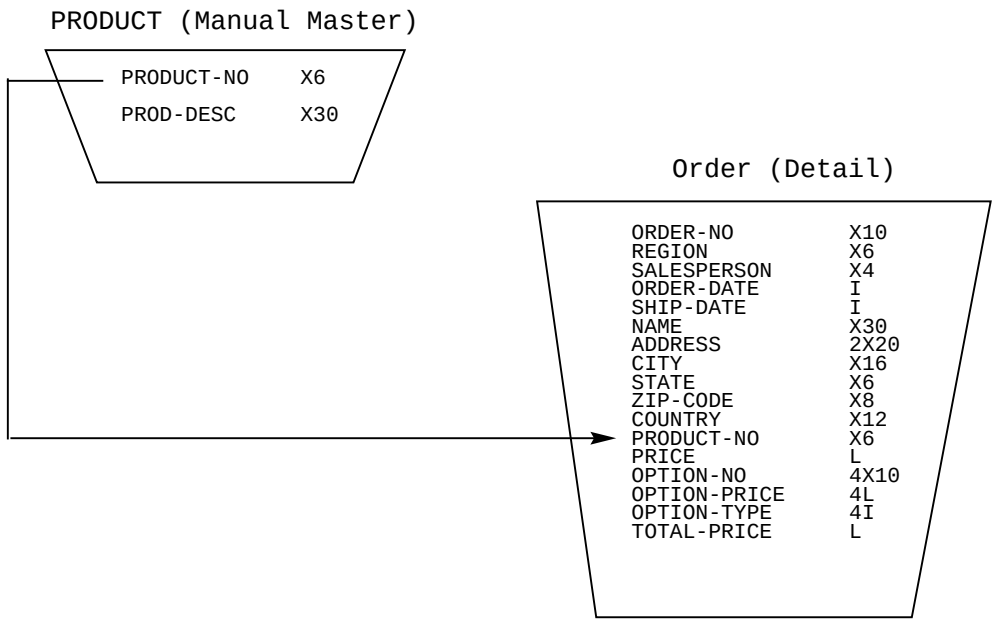


Figure 10 **Possible Database Structure**

Although an automatic master set could have been used for the PRODUCT master set, a manual master set was chosen for two reasons. First, since the PRODUCT set is a manual master, Eloquence DBMS automatically checks the validity of the product number (PRODUCT-NO) as an order is entered into the database. Second, additional data can be stored in the PRODUCT data set, such as a product description (PRODUCT-DESC). Automatic master entries cannot contain items other than the search item.

The database structure shown next utilizes a detail data set to store the option information and a master data set to store the rest of the order data. This organization of data corresponds to dividing an order into two forms, as shown in the next figure. Unlike the previous structures shown, the number of options that can be stored with any order is limited only by the number of free entries in the OPTION data set. In addition, this organization requires fewer disk space to store order information when fewer than four options are ordered. (In the previous examples, the space to store four options with each order is allocated whether the options are purchased or not.) Notice that all order numbers within the ORDER data set must be unique, since the order number is a search item.

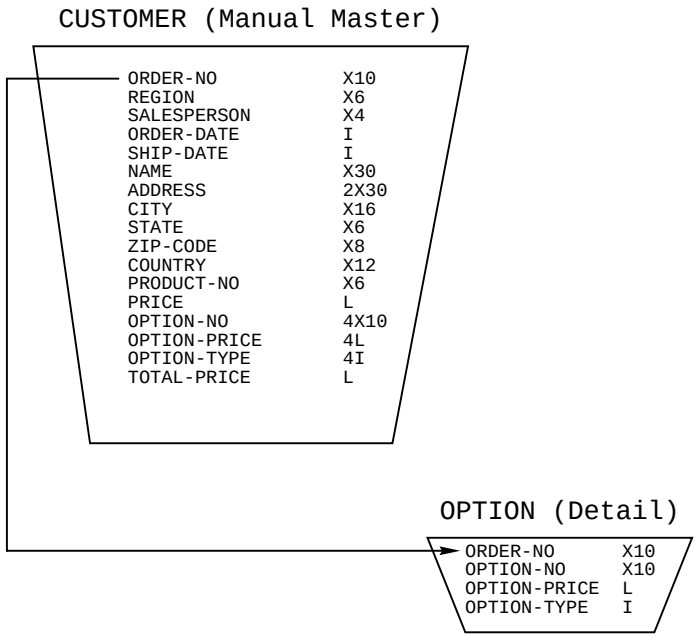


Figure 11 Possible Database Structure

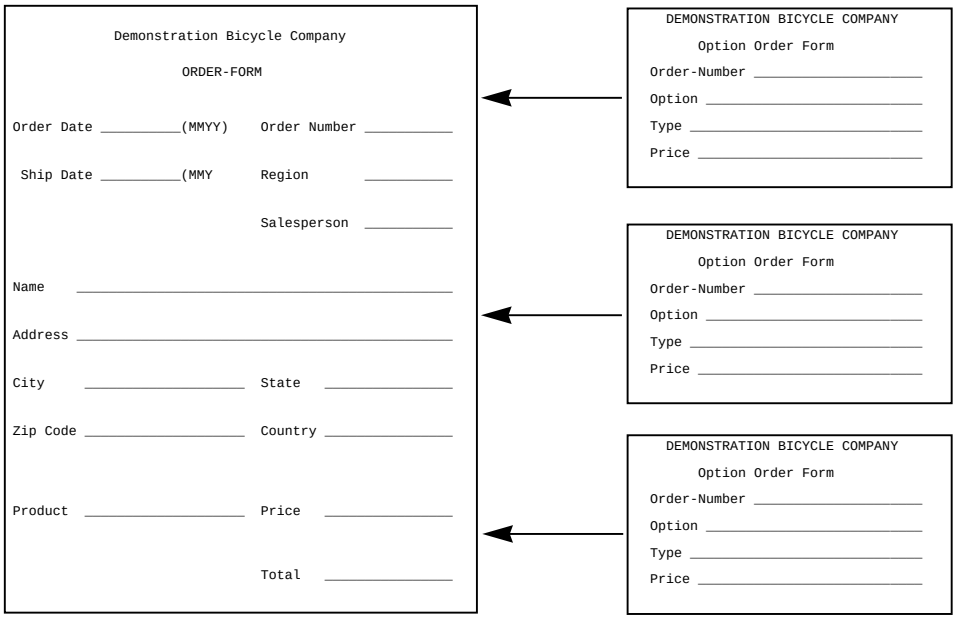


Figure 12 **Order Form with Separate Option Forms**

The advantages of the previous two structures are incorporated into the structure shown below. The CUSTOMER data set contains all order information except for option data, which is stored in the OPTION data set. Options are logically linked to the order through the ORDER master data set. Orders for a particular product are chained together using the PRODUCT master set.

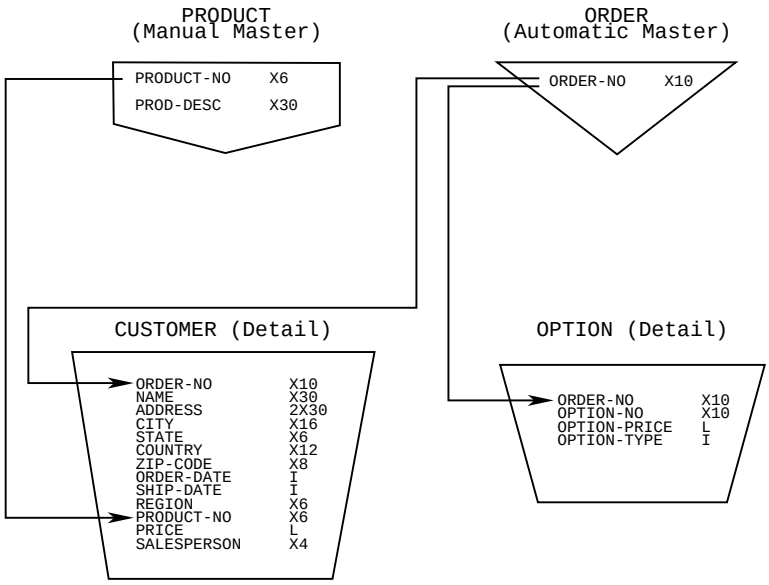


Figure 13 **Possible Database Structure**

Although Eloquence DBMS prevents orders from being entered for products not in the PRODUCT master, it does not prevent duplicate order numbers (ORDER-NO) from being entered. An order entry program can easily prevent the entry of duplicate order numbers, however, by performing a calculated DBGET on the ORDER master set before entering the order into the CUSTOMER and OPTION data sets.

The final database design, which is used throughout the rest of this chapter, is shown below. A second manual master set, LOCATION, is used to chain orders from a particular sales region. A second automatic master, DATE, is used to locate orders for a particular order date or ship date. Dates are stored as an integer number (to reduce disk space requirements), and are easily converted to and from an ASCII character date within an application program.

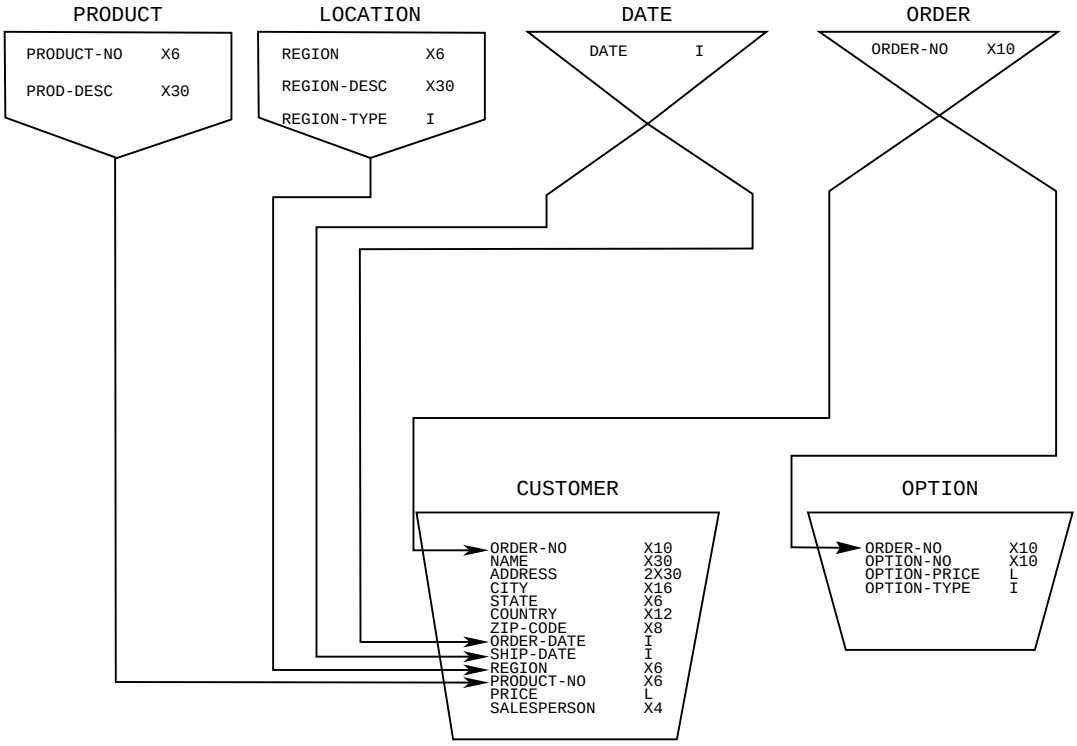


Figure 14 **Sales Analysis Database**

Database Definition and Creation

Once the database has been designed, the database definition language (DBDL) is used to define (describe) the database. An editing program, such as vi, is used to create a data file with the definitions. The following listing defines the Sales Analysis Database (SAD) described in the previous section, page 142 :

```
BEGIN DATABASE SAD;

PASSWORDS:
         3 SECRETARY;
        10 SALESMAN;
        15 MANAGER;

ITEMS:
        ADDRESS,        2 X30;
        CITY,           X16;
        COUNTRY,        X12;
        DATE,           I;
        NAME,           X30;
        OPTION-DESC,    X10;
        OPTION-PRICE,   L;
        OPTION-TYPE,    I;
        ORDER-DATE,     I;
        ORDER-NO,       X10;
        PRICE,          L;
        PRODUCT-NO,     I;
        PROD-DESC,      X30;
        REGION,         X6;
        REGION-DESC,    X30;
        REGION-TYPE,    I;
        SALESPERSON,    X4;
        SHIP-DATE,      I;
        STATE,          X6;
        ZIP-CODE,       X8;
        DUMMY,          S;

SETS:

N:      DATE, AUTOMATIC (3/10,15);
E:      DATE (2);

N:      ORDER, A (3/10,15);
E:      ORDER-NO (2);

N:      PRODUCT, MANUAL (3,10/15);
E:      PRODUCT-NO (1),
        PROD-DESC;

N:      LOCATION, M (3,10/15);
E:      REGION (1),
        REGION-DESC,
        REGION-TYPE;

N:      OPTION, DETAIL (3/10,15);
```

```
E:      ORDER-NO (ORDER),
        OPTION-DESC,
        OPTION-PRICE,
        OPTION-TYPE;

N:      CUSTOMER, D (3/10,15);
E:      ORDER-NO (ORDER),
        NAME,
        ADDRESS,
        CITY,
        STATE,
        COUNTRY,
        ZIP-CODE,
        ORDER-DATE (DATE),
        SHIP-DATE (DATE),
        REGION (LOCATION),
        PRODUCT-NO (PRODUCT),
        PRICE,
        SALESPERSON,
        DUMMY;

END.
```

The next step is to create the database root file. This is done by using the schema program, described in page 41 . For example, if the database definition is saved in the data file SAD.txt, start the schema program from the HP-UX prompt as follows:

```
schema SAD.txt
```

Once the schema program has created the root file, the data set files are created using the dbcreate command or the DBCREATE statement (discussed in page 105). An example of the dbcreate command follows:

```
dbcreate SAD
```

Eloquence DBMS Programming Examples

Once a database has been defined using the schema program and created using dbcreate, data can be written to and read from the database using the manipulation commands (page 59). This section gives examples of each of the database manipulation statements. All programs work with the Sales Analysis Database (SAD) discussed in the previous section.

Example Program 1

One of the simplest reports that can be produced is a list of a data set's contents. A sample listing of the contents of the CUSTOMER data set is shown next, as produced by example program 1.

Table 8 **OUTSTANDING ORDERS LIST**

ORDER NUMBER	CUSTOMER NAME	PRICE
100	SMITH THOMAS A.	175.50
101	NONAME, JOSEPH	77.50
102	JOHNSON, SAM	162.50
103	HERNANDES, JOSE	109.75
104	HOUSEMAN, SEAN	133.00
105	SONO, JOMO A.	135.00
106	HEINZ HEINING	175.00
107	DALLING, JIMMY	150.00
108	ARAUJA, LUCIANO	80.00
109	BEKKER,BART	125.00
110	GISSING,MALCOMB	45.00
		======
	TOTAL ORDERS	1368.25

For the programmer who has not used Eloquence DBMS, there are several small, but important, details which should be noted. In line 1080, B$ is defined as the database name. Two blanks must precede the name. The DBOPEN statement (line

1100) fills these blanks with a database id number (two ASCII digits from 00 through 09). This id number is used in subsequent DBML statements to identify the database, rather than the database name.

Note that the DBOPEN statement opens the database for exclusive access. This means that if another user attempts to open the database an error occurs when DBOPEN is executed. This error takes the form of a non-zero value in the first element of the status array S(*). S(*) must be of type integer and must contain at least ten elements, in this case S(0) through S(9).

```
1000 !   EXAMPLE PROGRAM 1
1010 !
1020 !   OUTSTANDING ORDERS REPORT (NOT INCLUDING ALL DETAIL)
1030 !
1040     INTEGER S(9)
1050     DIM B$[12],P$[10],Buf$[170]
1060     DIM Desc$[30],order_no$[10],Name$[30]
1070     DISP "Cr/H Cl/S";         ! CLEAR SCREEN
1080     B$="  SAD,SALES"
1090     P$="MANAGER"
1100     DBOPEN (B$,P$,3,S(*))   ! OPEN FOR EXCLUSIVE ACCESS
1110     IF S(0) THEN Dberr
1120  !
1130  !  INITIALIZE VARIABLES & PRINT REPORT HEADER
1140  !
1150  Rep: Total=0
1160     Eof=11
1170     PRINT TAB(20);"OUTSTANDING ORDERS LIST";LIN(1)
1180     PRINT "ORDER NUMBER     CUSTOMER NAME";SPA(14);"PRICE";
                  LIN(1);RPT$("-",48);LIN(1)
1190  !
1200  !  PRODUCE THE REPORT
1210  !
1220  START report:DBGET (B$,"CUSTOMER",2,S(*),"@",Buf$,0)
1230     IF S(0)=Eof THEN End_report
1240     IF S(0) THEN Dberr
1250     UNPACK USING Pf2;Buf$
1260  Pf2:PACKFMT Order_no$,Name$,60X,16X,6X,12X,8X,2X,2X,6X,2X,
          Price
1270     PRINT USING Itm_image;Order_no$,Name$,Price
1280  Itm_image: IMAGE 16A,22A,2X,5D.DD
1290  !
1300  !  ACCUMULATE TOTAL
1310  !
1320     Total=Total+Price
1330     GOTO Start_report
1340  !
1350  !  PRINT FINAL TOTALS
1360  !
1370  End_report:PRINT USING Tot_image;Total
1380     END
1390  Tot_image:IMAGE 39X,9("=") / 3X,"TOTAL ORDERS",24X,6D.DD /
1400  !
1410  !  DATABASE ERROR HANDLER
1420  !
1430  Dberr:DISP LIN(1);"UNEXPECTED DATABASE ERROR ";
            VAL$(S(0));" IN LINE";S(6)
1440     END
```

In line 1070 of the program on the previous page, the characters **`Cr/H`** and **`Cl/S`** should be replaced by the cursor home and clear display special control characters respectively.

The status array should be checked for an abnormal condition (non-zero first element) after each DBML operation. If an abnormal condition is detected in this program, control is transferred to the line labeled Dberr, which displays the error code in the status array and the line number where the error occurred.

The process of reading all orders is accomplished by a loop containing a serial-access DBGET (see line 1220). This line reads the next non-empty record from the CUSTOMER set and puts the record into the string Buf$.

The pertinent items from this buffer are extracted via the UNPACK USING statement and then printed. Note the use of Xs to skip unused fields in the PACKFMT. For clarity, each X field corresponds to a field in the database. The whole group could be replaced, however, by 114X. Before reading the next record, the price of the current order is added into the total price of all orders read so far.

When the orders in the CUSTOMER set are exhausted, the first element of the status array, S(0), indicates when an end-of-file has been reached. Line 1230 detects this and branches to print the total of all order prices.

Example Program 2

Example program 2 prints a list of all orders grouped by product. This is accomplished by serially reading each entry in the PRODUCT set (see line 1220). Then, for each product, a listing of the record contents with the same product number in the CUSTOMER detail data set is produced.

It is not necessary to scan the entire CUSTOMER section to find entries with the correct product number. The chain between the CUSTOMER set and the PRODUCT set with PRODUCT-NO as the search field allows direct access to those entries in the CUSTOMER set with a particular product number. To access the entries on the chain, a DBFIND is first executed (line 1340). Status array element S(5) returns the number of entries in the chain. A FOR-NEXT loop going from 1 to S(5) with a chained mode DBGET (line 1370) extracts the information for each order with the desired product number.

The procedure reads the chain in a “forward” direction. It is possible to read the chain backwards by using a direct mode DBGET and chain pointers which are returned in the status array by both DBFIND and DBGET. It is also possible to use DBGET mode 6 (chained backward). To change example program 2 to do a backward chain read, replace line 1370 with the following:

```
1370 DBGET (B$,"CUSTOMER",4,S(*),"@",Buf$,S(7))
```

or

```
1370 DBGET (B$,"CUSTOMER",6,S(*),"@",Buf$,(0))
```

Example program 2 also solves one of the problems of program 1. By opening the database in mode 8 (read-only mode) instead of mode 3 (exclusive access), other users can also open the database in mode 8 and do concurrent reads. Opening in mode 8, however, fails if another user has the database opened in either mode 3 or mode 1. As long as the database is opened by one user in mode 8, no one can open it in a mode which permits modifications of the database.

```
                    OUTSTANDING ORDERS LIST

PRODUCT            ORDER NUMBER        CUSTOMER NAME          PRICE

-----------------------------------------------------------------
50   (Tricycle)
                   110                 Gissing, Malcomb       45.00
                                                            =========
  TOTAL ORDERS FOR 50                                         45.00

1000 (10-Speed Bicycle)
                   102                  Johnson, Sam         162.50
                   106                  Heining, Heinz       175.00
                   107                  Dalling, Jimmy       150.00
                                                            =========
  TOTAL ORDERS FOR 1000                                       487.50

100  (Standard Bicycle)
                   101                  Noname, Joseph        77.50
                   103                  Hernandes, Jose      109.75
                   108                  Arauja, Luciano A.    80.00
                                                            =========
  TOTAL ORDERS FOR 100                                        267.25

300  (3-Speed Bicycle)
                   104                  Houseman, Sam        133.00
                                                            =========
  TOTAL ORDERS FOR 300                                        133.00

500  (5-Speed Bicycle)
                   100                  Smith, Thomas A.     175.50
                   105                  Sono, Jomo A.        135.00
                   109                  Bekker, Bart         125.00
                                                            =========
  TOTAL ORDERS FOR 500                                        435.50

                                        TOTAL ORDERS        $1368.25
                                                           ===========
```

```
1000  !   EXAMPLE PROGRAM 2
1010  !
1020  !   OUTSTANDING ORDERS REPORT (NOT INCLUDING ALL DETAIL)
1030  !
1040     INTEGER S(9), Prod_no
```

```
1050     DIM B$[12],P$[10],Buf$[170]
1060     DIM Desc$[30],Order_no$[10],Name$[30]
1070     DISP "Cr/H Cl/S";         ! CLEAR SCREEN
1080     B$="  SAD,SALES"
1090     P$="MANAGER"
1100     DBOPEN (B$,P$,8,S(*))     ! OPEN FOR READ-ONLY ACCESS
1110     IF S(0) THEN Dberr
1120  !
1130  !   INITIALIZE VARIABLES & PRINT REPORT HEADER
1140  !
1150  Rep:Total=Master_total=0
1160     Eof=11
1170     PRINT TAB(20);"OUTSTANDING ORDERS LIST";LIN(1)
1180     PRINT "PRODUCT       ORDER NUMBER      'CUSTOMER NAME";
         SPA(14);"PRICE";LIN(1);RPT$("-",63);LIN(1)
1190  !
1200  !   PRODUCE THE REPORT
1210  !
1220  Start_report:DBGET (B$,"PRODUCT",2,S(*),"@",Buf$,0)
1230     IF S(0)=Eof THEN End_report
1240     IF S(0) THEN Dberr
1250     UNPACK USING Pf;Buf$
1260  Pf: PACKFMT Prod_no,Desc$
1270  !
1280  !   PRINT HEADER FOR PRODUCT
1290  !
1300     PRINT VAL$(Prod_no);" (";TRIM$(Desc$);")"
1310  !
1320  !   PRINT ORDERS
1330  !
1340     DBFIND (B$,"CUSTOMER",1,S(*),"PROD_NO",Prod_no)
1350     IF S(0) THEN Dberr
1360     FOR I=1 TO S(5)
1370       DBGET (B$,"CUSTOMER",5,S(*),"@"Buf$,0)
1380       IF S(0) THEN Dberr
1390       UNPACK USING Pf2;Buf$
1400  Pf2: PACKFMT Order no$,Name$,60X,16X,6X,12X,8X,2X,2X,6X,
             2X,Price
1410       PRINT TAB(16);
1420       PRINT USING Itm_image;Order_no$,Name$,Price
1430  Itm_image:IMAGE 16A,22A,2X,5D.DD
1440  !
1450  !    ACCUMULATE TOTALS
1460  !
1470       Total=Total+Price
1480       Master_total=Master_total+Price
1490     NEXT I
1500  !
1510  !   PRINT TRAILER FOR PRODUCT
1520  !
1530     PRINT TAB(54);
1540     PRINT USING Tot_image;VAL$(Prod_no),Total
1550     Total=0
1560     GOTO Start_report
1570  !
1580  !   PRINT FINAL TOTALS
1590  !
1600  End_report:PRINT USING Mstr_image; Master_total
1610     END
1620  Tot_image:IMAGE 9("=") / 3X,"TOTAL ORDERS FOR ",10A,24X, 6D.DD
```

```
/
1630     Mstr_image:IMAGE   //   25X,"TOTAL   ORDERS",14X,"$"8D.DD   /
54X,9("=")
1640  !
1650  !   DATABASE ERROR HANDLER
1660  !
1670  Dberr:DISP LIN(1);"UNEXPECTED DATABASE ERROR ";
             'VAL$(S(0));" IN LINE";S(6)
1680     END
```

In line 1070 of the above program, the characters **Cr/H** and **Cl/S** should be replaced by the cursor home and clear display special control characters respectively.

Example Program 3

Example program 3 allows other users to perform write operations (DBPUT, DBDELETE, and DBUPDATE) by opening the database in mode 1. Other users can now open the database in mode 1 for reading. By using the locking capability, other programs can perform puts, deletes, and updates.

This program is basically an expansion of the previous example. The options for each order are listed by inserting a chained-access DBGET through the OPTION set for each order found in CUSTOMER. The DBFIND on the OPTION set (line 1470) using the order number from CUSTOMER finds the head of the chain of options. The FOR-NEXT loop (line 1490) then chains through the entries in the OPTION set, doing chained-mode DBGETS.

A significant feature of this program is the replacement of the UNPACK USING statements with IN DATA SETs. Lines 1180 through 1200 set up a correspondence between variables in the Eloquence program and fields in the data sets. Thus, when the DBGET on PRODUCT is performed (line 1310), the values of the fields PRODUCT-NO and PRODUCT-DESC in the PRODUCT set are automatically assigned to the variables Prod_no and Desc$. Similarly, when the DBGET in line 1500 is executed, new values are assigned to Option_desc$ and P0.

The use of SKP in the IN DATA SET for the OPTION set (line 1200) instructs the system to ignore the value of the ORDER-NO field. It is not assigned to any variable since this field was read by DBGET on the CUSTOMER set and assigned to Order_no$; reassigning it would be superfluous.

The USE ALL option on the IN DATA SET for the CUSTOMER set (line 1190) specifies that only fields whose names correspond to variables already in the program are to be unpacked into their corresponding variables.

Only the variables Order_no$ and Name$ in this program correspond to fields in the CUSTOMER set. Thus, when the DBGET in line 1410 is executed, only the value of the fields ORDER-NO and NAME are assigned to variables (Order_no$ and Name$, respectively).

Another feature of example program 3 is error control and program termination. Lines 1080 and 1090 allow the program to trap the HALT key and any error conditions and wrap-up gracefully. Both termination conditions, as well as the database routine, then attempt to close the database (line 1900). In this instance, however, the DBCLOSE is not critical. Had any write operations been performed, the close would be necessary to properly record the changes made to the database.

A mode 4 DBCLOSE is automatically executed when the program ENDs when the database is not closed prior to program completion. This close updates the data stored on the disk, but leaves the database open for further access. Certain operations, such as DBERASE, require exclusive access to the database, and cannot be performed until a mode 1 DBCLOSE is executed.

```
                        OUTSTANDING ORDERS LIST

PRODUCT    ORDER NUMBER      CUSTOMER NAME     OPTIONS     PRICE
-------------------------------------------------------------------
---
50   (Tricycle)
           110               Gissing, Malcomb             45.00
                                                         --------
                                                          45.00
                                                         ========
          TOTAL ORDERS FOR 50                             45.00

1000 (10-Speed Bicycle)
           102               Johnson, Sam                150.00
                                              Chrome       12.50
                                                         --------
                                                         162.50

           106               Heining, Heinz              150.00
                                              Light        10.00
                                              Basket       15.00
                                                         --------
                                                         175.00

           107               Dalling, Jimmy              150.00
                                                         --------
                                                         150.00
                                                         ========
          TOTAL ORDERS FOR 1000                          487.50

100  (Standard Bicycle)
           101               Noname, Joseph               75.00
                                              Horn          2.50
                                                         --------
                                                          77.50
```

```
          103                Hernaned, Jose                75.00
                                                  Light      5.00
                                                  Mud Flaps  2.50
                                                  Horn      10.00
                                                  Stripes    2.50
                                                  Fan       10.00
                                                            --------
                                                           109.75

         108                 Arauja, Luciano A.            75.00
                                                  Horn       5.00
                                                            --------
                                                            80.00
                                                            ========
          TOTAL ORDERS FOR 100                             267.25

300  (3-Speed Bicycle)
         104                 Houseman, Sean                110.00
                                                  Light      5.00
                                                 Super Tire 18.00
                                                            --------
                                                           133.00
                                                            ========
          TOTAL ORDERS FOR 300                             133.00

500  (5-Speed Bicycle)
         100                 Smith, Thomas A.              125.00
                                                  Light      5.00
                                                  Basket    45.00
                                                            --------
                                                           175.50

         105                 Sono, Jomo A.                 125.00
                                                  Horn       2.50
                                                  Reflector  7.50
                                                            --------
                                                           135.00

         109                 Bekker, Bart                  125.00
                                                            --------
                                                           125.00
                                                            ========
          TOTAL ORDERS FRO 500                             435.50

                             TOTAL ORDERS                $1368.25
                                                         ==========
```

```
1000  !   EXAMPLE PROGRAM 3
1010  !
1020  !   OUTSTANDING ORDERS REPORT (INCLUDING ALL DETAIL)
1030  !
1040     INTEGER S(9), Product_no,Prod_no
1050     DIM B$[12],P$[10],Buf$[170]
1060     DIM Desc$[30],Order_no$[30],Name$[30],
                  Option_desc$[10]
1070     DISP “Cr/H Cl/S”;        ! CLEAR SCREEN
1080     ON ERROR GOTO Error      ! SET UP ERROR AND HALT TRAPS
1090     ON HALT GOTO Halt
1100     B$=”  SAD,SALES”
1110     P$=”MANAGER”
```

```
1120     DBOPEN (B$,P$,1,S(*))     ! OPEN FOR SHARED ACCESS
1130     IF S(0) THEN Dberr
1140  !
1150  !   SET UP ALL APPROPRIATE RELATIONSHIPS
1160  !
1170     DBASE IS B$
1180     IN DATA SET "PRODUCT" USE Prod_no,Desc$
1190     IN DATA SET "CUSTOMER" USE ALL
1200     IN DATA SET "OPTION" USE SKP 1,Option_desc$,P0
1210  !
1220  !   INITIALIZE VARIABLES & PRINT REPORT HEADER
1230  !
1240  Rep:Total=Master_total=0
1250     Eof=11
1260     PRINT TAB(30);"OUTSTANDING ORDERS LIST";LIN(1)
1270     PRINT "PRODUCT";SPA(8);"ORDER NUMBER";SPA(10);
         "CUSTOMER NAME";SPA(9);"OPTIONS";SPA(8);"PRICE";
         LIN(1);RPT$("=",79);LIN(1)
1280  !
1290  !   PRODUCE THE REPORT
1300  !
1310  Start_report:DBGET (B$,"PRODUCT",2,S(*),"@",Buf$,0)
1320     IF S(0)=Eof THEN End_report
1330     IF S(0) THEN Dberr
1340  !
1350  !   PRINT HEADER FOR PRODUCT
1360  !
1370     PRINT VAL$(Prod_no);" (";TRIM$(Desc);")"
1380     DBFIND (B$,"CUSTOMER",1,S(*),"PRODUCT-NO,Prod_no)
1390     IF S(0) THEN Dberr
1400     FOR I=1 TO S(5)
1410       DBGET (B$,"CUSTOMER",5,S(*),"@",Buf$,0)
1420       IF S(0) THEN Dberr
1430  !
1440  !     PRINT HEADER FOR ORDER
1450  !
1460       PRINT TAB(20);Order_no$;TAB(38);Name$[1,21];
1470       DBFIND (B$,"OPTION",1,S(*),"ORDER-NO",Order_no$)
1480       IF S(0) THEN Dberr
1490       FOR J=1 TO S(5)
1500         DBGET (B$,"OPTION",5,S(*),"@",Buf$,0)
1510         IF S(0) Then Dberr
1520  !
1530  !      PRINT OPTIONS
1540  !
1550         PRINT TAB(60);
1560         PRINT USING Itm_image;Option_desc$,P0
1570  Itm_image:IMAGE 10A,2X,5D.DD
1580  !
1590  !      ACCUMULATE TOTALS
1600  !
1610         Total=Total+P0
1620         Sub_total=Sub_total+P0
1630         Master_total=Master_total+P0
1640       NEXT J
1650       PRINT TAB(71);
1660       PRINT USING Sub_image;Sub_total
1670       Sub_total=0
1680     NEXT I
1690  !
```

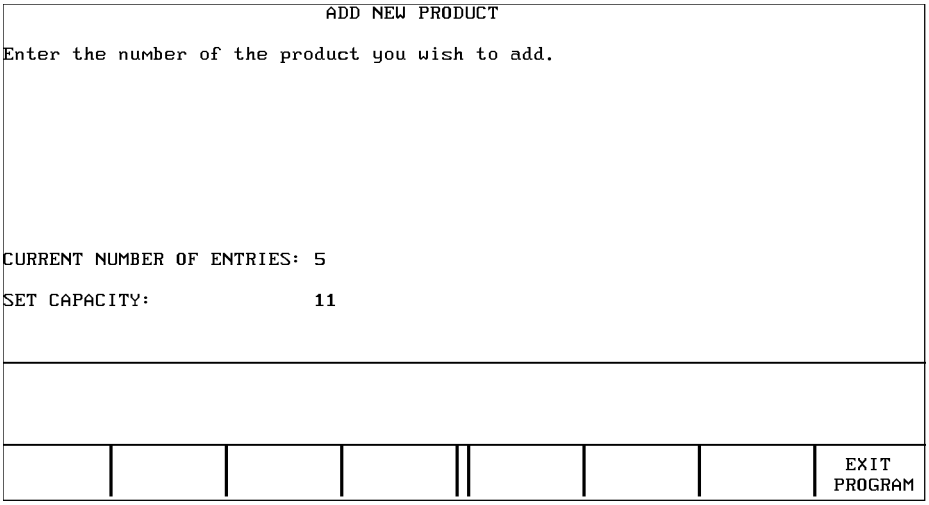
```
1700  !   PRINT TRAILER FOR PRODUCT
1710  !
1720     PRINT TAB(70);
1730     PRINT USING Tot_image;VAL$(Prod_no),Total
1740     Total=0
1750     GOTO Start_report
1760  !
1770  !   PRINT FINAL TOTALS
1780  !
1790  End_report:PRINT USING Mstr_image;Master_total
1800     GOTO Close
1810  Sub_image:IMAGE 8(”-”) / 71X,5D.DD /
1820  Tot_image:IMAGE 9(”=”) / 11X,”TOTAL ORDERS FOR “,10A,32X,6D.DD
/
1830     Mstr_image:IMAGE  //  31X,”TOTAL  ORDERS”,24X,”$”8D.DD  /
70X,9(”=”)
1840  !
1850  !     ERROR AND HALT TERMINATION ROUTINES
1860  !!
1870  Dberr:DISP LIN(2);”STATUS ERROR “;VAL$(S(0));” IN LINE”; S(6)
1880     GOTO Close
1890  Error: DISP LIN(2);”UNEXPECTED “;ERRM$
1900     GOTO Close
1910  Halt:PRINT LIN(2)
1920  Close:DBCLOSE (B$,” “,1,S(*))
1930     DISP LIN(2);”END OF OUTSTANDING ORDERS REPORT.”
1940     END
```

In line 1070 of the above program, the characters **Cr/H** and **Cl/S** should be replaced by the cursor home and clear display special control characters respectively.

Example Program 4

The last two programs are used to enter new products into the database and make modifications and deletions to existing products. Example program 4 allows new products to be added to the database. Since write operations must be performed, the database is opened in mode 3 (see line 1130). This program also contains the necessary lines to trap HALTs and errors (see lines 1090 and 1100).

When the program is first RUN it produces a screen like that shown below.



Press EXIT PROGRAM to terminate the program. Otherwise, a new product number is entered and the program prompts for a product description. Note that the set capacity and the current number of entries are displayed. This information is obtained via DBINFO in line 1330. Note also that since the result of DBINFO is left in the buffer, an UNPACK is used (line 1350) to extract these values into integer variables. Since integers range from -2^{31} through $+2^{31}-1$ and capacities range from 1 through 999,999, a conversion is necessary. The FNCorrect function defined in line 1240 provides this conversion.

The same would be possible using a DINTEGER variable to unpack instead of 2 INTEGERS.

If a duplicate product number is entered, the program displays an error and prompts for a correction. For example:

```
                          ADD NEW PRODUCT

Enter the number of the product you wish to add.
50
PRODUCT ALREADY IN DATABASE.




CURRENT NUMBER OF ENTRIES: 5

SET CAPACITY:              11

                                                                  EXIT
                                                                  PROGRAM
```

This duplicity is detected by attempting a calculated access (line 1460) of the master for the product using the given product number.

If the specified product is not found, the product description is requested as shown here:

```
                          ADD NEW PRODUCT

Enter the number of the product you wish to add.
1

Enter product description.
Uni-cycle

                                                                  EXIT
                                                                  PROGRAM
```

After the description is entered, a DBPUT (line 1600) is used to store the new product in the database. The program then reports that the DBPUT was successful and gives two options which are key selected.

The user can either add ANOTHER product or EXIT the program.

```
                          ADD NEW PRODUCT

Enter the number of the product you wish to add.
1

Enter product description.
Uni-cycle
NEW PRODUCT ADDED.
```

ANOTHER							EXIT PROGRAM

```
1000  !   EXAMPLE PROGRAM 4
1010  !
1020  !   ADD NEW PRODUCT
1030  !
1040     INTEGER S(9),Prod_no,Entries,Capacity,Entries2,Capacity2
1050     DIM B$[12],P$[10],Buf$[170]
1060     DIM Desc$[30]
1070     DISP "Cr/H Cl/S";         ! CLEAR SCREEN
1080     DISP TAB(32);"ADD NEW PRODUCT'
1090     ON ERROR GOTO Error       !SET UP ERROR AND HALT TRAPS
1100     ON HALT GOTO Halt
1110     B$="  SAD,SALES"
1120     P$="MANAGER"
1130     DBOPEN (B$,P$,3,S(*))     ! OPEN FOR EXCLUSIVE ACCESS
1140     IF S(0) THEN Dberr
1150  !
1160  !   SET UP ALL APPROPRIATE RELATIONSHIPS
1170  !
1180     DBASE IS B$
1190     IN DATA SET "PRODUCT" USE Prod_no,Desc$
1200  !
1210  !   FUNCTION TO CONVERT TWO 16-BIT INTEGERS TO
1220  !   ONE 32-BIT INTEGER
1230  !
1240     DEF FNCorrect(INTEGER N,N2)=N2+N*65536
1250  !
1260  !   INITIALIZE
1270  !
1280     Not_found=17
1290     ON KEY #8:"EXIT" GOTO Halt
1300     ON KEY #16 GOTO Halt
1310  Cont:DISP "Cr/H";LIN(3);"Cl/S";LIN(10)
1320     OFF KEY #1,9
1330     DBINFO (B$,"PRODUCT",202,S(*),Buf$)
1340     UNPACK USING Fmt;Buf$
```

```
1350  Fmt:PACKFMT 28X,Entries,Entries2,Capacity,Capacity2
1360          DISP  “CURRENT  NUMBER  OF  ENTRIES  :”;FNCorrect(En-
tries,Entries2);
                LIN(2)
1370    DISP “SET CAPACITY:          “;FNCorrect(Capacity,Capacity2)
1380  !
1390  !   ASK FOR NEW PRODUCT NUMBER
1400  !
1410  Again:DISP “Cr/H”;LIN(3)
1420     Badp=-1                 ! ALLOW FOR A NULL USER RESPONSE.
1430     INPUT “Enter the number of the product you wish to add.”,
               Prod_no
1440     IF Prod_no%<1 THEN Badp
1450     DBGET (B$,”PRODUCT”,7,S(*),”@”,Buf$,Prod_no)
1460     IF S(0)=Not_found THEN Enter
1470     IF S(0) THEN Dberr
1480     DISP “PRODUCT ALREADY IN DATABASE.”
1490     BEEP
1500     GOTO Again
1510  Badp:DISP “ILLEGAL PRODUCT NUMBER”
1520     BEEP
1530     GOTO Again
1540  !
1550  !   PUT THE NEW PRODUCT IN THE DATABASE
1560  !
1570  Enter:DISP “Cl/S”
1580     INPUT “Enter product description”,Desc$
1590     DBPUT (B$,”PRODUCT”,1,S(*),”@”,Buf$)
1600     IF S(0) THEN Dberr
1610     DISP “NEW PRODUCT ADDED.”
1620     ON KEY #1:”ANOTHER GOTO Cont
1630     ON KEY #9 GOTO Cont
1640     WAIT
1650  Dberr:DISP LIN(2);”STATUS ERROR “;VAL$(S(0));” IN LINE”;
                S(6)
1660     GOTO Close
1670  Error:DISP LIN(2);”UNEXPECTED “;ERRM$
1680     GOTO Close
1690  Halt:DISP “    END OF ADD PRODUCT PROGRAM.”
1700  Close:DBCLOSE (B$,” “,1,S(*))
1710     END
```

In lines 1070, 1310, 1570 and 1690 of the above program, the characters **Cr/H** and **Cl/S** should be replaced by the cursor home and clear display special control characters respectively.

Example Program 5

Example program 5 allows products in the PRODUCT data set to be changed or deleted. The program prompts for the number of the product to be edited. The EXIT key can be pressed at any time to stop the program. The initial display is as follows:

```
                                EDIT PRODUCT

Enter the number of the product you wish to edit.
```

```
                                                                  EXIT
                                                                  PROGRAM
```

If a number is entered for a non-existent product, it is detected by the calculated access DBGET in line 1330, as shown by the following screen:

```
                                EDIT PRODUCT

Enter the number of the product you wish to edit.
23
NO SUCH PRODUCT IN DATABASE.
```

```
                                                                  EXIT
                                                                  PROGRAM
```

Note that the database was opened in mode 1. Line 1310 locks the database. It is essential that the database be locked before the DBGET. If it is locked afterwards, another user could lock the database anytime before the lock in this program and then make modifications to the record retrieved in line 1330. If such a user deleted the record, a subsequent DBUPDATE or DBDELETE would fail.

Once a correct product number has been provided, the old description is displayed for user edit:

```
                              ADD NEW PRODUCT

Enter the number of the product you wish to add.
1

Enter new description.
Uni-cycle
```

```
+---------+---------+---------+---------+---------+---------+---------+---------+
| ANOTHER |         |         |         | DELETE  |         |         | EXIT    |
|         |         |         |         |         |         |         | PROGRAM |
+---------+---------+---------+---------+---------+---------+---------+---------+
```

The ANOTHER key can be pressed here to abort the modification and return to the initial display.

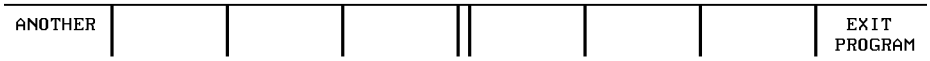
If the description is altered and RETURN is pressed, the DBUPDATE line (1520) alters the text of the product description in the database. In this case, either ANOTHER or EXIT PROGRAM is pressed to continue:

```
                              ADD NEW PRODUCT

Enter the number of the product you wish to add.
1

Enter new description.
Unicycle
UPDATE COMPLETE.
```

```
+---------+---------+---------+---------+---------+---------+---------+---------+
| ANOTHER |         |         |         |         |         |         | EXIT    |
|         |         |         |         |         |         |         | PROGRAM |
+---------+---------+---------+---------+---------+---------+---------+---------+
```

If the DELETE key is pressed, the product is removed from the database (see line 1670). The program then indicates that the delete was successful and waits for the user to respond with the appropriate key:

```
                              ADD NEW PRODUCT

Enter the number of the product you wish to add.
1

Enter new description.
Unicycle
PRODUCT DELETED.
```



If the entry in the PRODUCT master had any entries associated with it in the CUSTOMER detail, an error would have been issued (see line 1690).

Note that extreme care must be taken so that DBUNLOCKs are performed either after a successful operation or following an error (see lines 1350, 1550 and 1600). The DBCLOSE in line 1780 automatically performs the unlock in case either HALT or EXIT PROGRAM is pressed or an unforeseen error occurs.

```
1000  !   EXAMPLE PROGRAM 5
1010  !
1020  !   PRODUCT EDITOR
1030  !
1040     INTEGER S(9),Prod_no
1050     DIM B$[12],Buf$[170]
1060     DIM Desc$[30]
1070     DISP “Cr/H Cl/S”;        ! CLEAR SCREEN
1080     DISP TAB(34);”EDIT PRODUCT”
1090     ON ERROR GOTO Error       ! SET UP ERROR AND HALT TRAPS
1100     ON HALT GOTO Halt
1110     B$=”  SAD,SALES”
1120     P$=”MANAGER”
1130     DBOPEN (B$,P$,1,S(*)      ! OPEN FOR SHARED ACCESS
1140     IF S(0) THEN Dberr
1150  !
1160  !   SET UP ALL APPROPRIATE RELATIONSHIPS
1170  !
1180     DBASE IS B$
1190     IN DATA SET “PRODUCT” USE Prod_no,Desc$
1200  !
```

```
1210  !   INITIALIZE AND ASK FOR PRODUCT NUMBER
1220  !
1230     Not_found=17
1240     Chain_not_empty=44
1250     ON KEY #8:"EXIT" GOTO Halt
1260     ON KEY #16 GOTO Halt
1270  Cont:DISP "Cr/H";LIN(3);"Cl/S";
1280     OFF KEY #1,9
1290  Again:DISP "Cr/H";LIN(3);
1300     INPUT "Enter the number of the product you wish to edit.",
               Prod_no
1310     DBLOCK (B$," ",1,S(*))     ! LOCK DATABASE BEFORE WRITE
1320     IF S(0) THEN Dberr
1330     DBGET (B$,"PRODUCT",7,S(*),"@",Buf$,Prod_no)
1340     IF S(0) %<> Not_found THEN Maybe
1350    DBUNLOCK (B$," ",1,S(*))!UNLOCK THE DATABASE AFTER AN ERROR
1360     IF S(0) THEN Dberr
1370     DISP "NO SUCH PRODUCT IN DATABASE."
1380     BEEP
1390     GOTO Again
1400  Maybe:IF S(0) THEN Dberr
1410  !
1420  !   GET NEW DESCRIPTION AND PERFORM THE UPDATE
1430  !
1440     DISP "Cl/S"
1450     Desc$=Trim$(Desc$)
1460     ON KEY #5:"DELETE" GOTO Del
1470     ON KEY #13 GOTO Del
1480     EDIT "Enter New description",Desc$
1490     OFF KEY #5,13
1500     DBUPDATE (B$,"PRODUCT",1,S(*),"@",Buf$)
1510     IF S(0) THEN Dberr
1520     DISP "UPDATE COMPLETE."
1530  WAIT:DBUNLOCK (B$," ",1,S(*)) ! UNLOCK DATABASE AFTER WRITE
1540     IF S(0) THEN Dberr
1550     ON KEY #1:"ANOTHER GOTO Cont
1560     ON KEY #9 GOTO Cont
1570     WAIT
1580  !
1590  !   DELETE THE ENTRY
1600  !
1610  Del:OFF KEY #5,13
1620     DBDELETE (B$,"PRODUCT',1,S(*))
1630     IF S(0) %<>Chain_not_empty THEN Del2
1640     DISP LIN(1);"THERE ARE STILL ORDERS FOR THIS PRODUCT."
1650     BEEP
1660     GOTO Wait
1670  De12:IF S(0) THEN Dberr
1680     DISP LIN(1);"PRODUCT DELETED."
1690     GOTO Wait           ! GO DO UNLOCK.
1700  !
1710  !   TERMINATION ROUTINES
1720  !
1730  Dberr:DISP LIN(2);"STATUS ERROR ";VAL$(S(0));" IN LINE";
            S(6)
1740     GOTO Close
1750  Error:DISP LIN(2);"UNEXPECTED ";ERRM$
1760     GOTO Close
1770  Halt:DISP "Cr/H END OF EDIT PRODUCT PROGRAM."
1780  Close:DBCLOSE (B$," ",1,S(*))
1790     END
```

In lines 1070, 1270, 1290, 1440 and 1770 of the above program, the characters **`Cr/H`** and **`Cl/S`** should be replaced by the cursor home and clear display special control characters respectively.

Database Locking

Locking is a means of communication and control used by mutually cooperating programs. A lock on a particular section of the database prevents other programs from modifying that section. DBLOCK is used only on databases opened in mode 1, shared write access. In access mode 3 and 8, DBLOCK is ignored.

The DBLOCK statement operates in one of twelve modes. Modes 1 through 6 apply a write lock to the section specified; modes 11 through 16 apply a read lock to the section specified. Modes 1, 2, 11, and 12 are used to lock an entire database. Modes 3, 4, 13, and 14 are used to lock a data set. Modes 5, 6, 15, and 16 are used to lock an entry or group of entries specified by a lock descriptor.

A program can read a section of the database without locking it even if the section is locked by another program. If a second program is modifying the database during the read, unexpected results can occur. For example, while a program is performing chained GET's, the address of the next entry can be modified by a second program. To prevent this, the appropriate data set or entries in the data set should be locked.

Locks used to protect read operations should be read locks. A read lock prevents other programs from getting a write lock on an entry, thus preventing any modification of that entry. A read lock will not prevent other programs from getting a read lock on the same entry. The shared read modes allow for greater access to the database while preventing write operations.

The DBLOCK operation makes no modifications to the database itself. The entries locked do not have to exist in the database. This will be the case when a new entry is created by a DBPUT.

Lock Descriptors

In modes 5, 6, 15, and 16, the program specifies the entries to be locked through a lock descriptor. A descriptor consists of a set name, a relational operator, and an associated value. (Refer to page 94 for a complete description). Multiple descriptors can be concatenated to build complex locks. The string of lock descriptors, called a lock predicate, is passed to the locking system in the QUALIFIER parameter of DBLOCK.

If @ is specified for the data set name, Eloquence DBMS interprets this to mean “lock all data sets” (meaning, the whole database. This is equivalent to the operation of modes 1, 2, 11, and 12). Similarly, if @ is specified for the item name, the interpretation is “lock all items” in the specified data set (meaning, lock that data set; equivalent to modes 3, 4, 13, and 14).

Eloquence DBMS also allows set name to be @ with item name, relational operator, and value specified for a desired entry value. This means “lock this entry wherever it occurs in any set”.

A lock descriptor is a string expression with a very specific format, as described below:

Length One word integer containing the physical length in words of this descriptor. The length includes the length parameter itself.

Set Name Eight words containing the name of an existing Eloquence DBMS data set up to 15 bytes long padded on the right with blanks or a set number stored as a binary integer from 1 through 99 in the first word; the remaining seven words are ignored. If the first (leftmost) byte of the name is @, then the remaining 15 bytes are ignored and Eloquence DBMS interprets this to mean “all sets in the database”. If the first word of this field is a binary zero, the whole predicate is ignored.

Item Name Eight words containing the name of an existing Eloquence DBMS data item (need not be a key item) up to 15 bytes long padded on the right with blanks or an item number stored as a binary integer from 1 through 1024 in the first word; the remaining seven words are ignored If the first (leftmost) byte of this field is @, then the remaining 15 bytes are ignored. Eloquence DBMS interprets the @ to mean “all items in the set” (the whole set); the value field is ignored.

Relop One word containing one of these relational operators stored in its ASCII representation:

ASCII Comarison
= equal
>= greater than or equal
<= less than or equal

Lexical Comparison
EQ equal
GE greater than or equal

LE less than or equal

NOTE: Lexical comparison is currently handled the same as ASCII comparison.

For the equal operator, the operator character can appear in either byte and the other byte must be a blank character (octal 40).

Value The value of the specified item to be locked. It should be stored exactly as stored within the database. However, for numeric types, Eloquence DBMS will perform any necessary conversion, if possible. If a conversion error is caused by a length incompatibility or size incompatibility, Eloquence DBMS returns a status error. Eloquence DBMS uses as many words as required by the corresponding item definition. If a string value is given that is shorter than that specified for the item, blanks are added as needed. If the string value is longer, it is truncated.

Setting Up Predicates

The predicates required by DBLOCK in modes 5, 6, 15, and 16 can be created by several methods. Simply building a string expression with concatenation can be the first approach. For example:

```
CHR$(0) & CHR$(21) & Set$ & Item$ & "%<=" & "GEORGE"
```

NOTE: The example above implies a CPU with big endian byte order (eg. HPPA). For litte endian system (eg. Intel) the CHR$(0/21) must be exchanged.

The one-word integer length field is formed by two-byte values using the CHR$ function. The length value must be carefully specified to include the complete descriptor. The Set$ and Item$ strings must have a current length of 16.

The PACK USING statement can also be used to construct a predicate. The next sequence could be used in a program to lock PRODUCT-NO=16117 in the data set PRODUCT.

```
100   INTEGER N, Product_no, Stat(9)
110   DIM Q$[40],Lock_set$[16],Lock_item$[16],Relop$[2]
120   Product_no=16117
130   Lock_item$="PRODUCT-NO        "
140   Lock_set$="PRODUCT            "
150   Relop$="=  "
160   N=19
170   Mode=5
180   PACK USING 190;Q$
190   PACKFMT N, Lock_Set$, Lock_item$, Relop$, Product_no
```

```
200   DBLOCK (Base$, Q$, Mode, Stat(*))
 *
 *
 *
```

The disadvantage of these methods is that the programmer must manually specify the descriptor length and guarantee the correct length for each field. The PREDICATE statement computes the descriptor length and insures that the format is correct. To create the descriptor in the previous example, the programmer need only write the following:

```
500   PREDICATE Q$ FROM “PRODUCT”,”PRODUCT-NO”,”=”,16177
```

The programmer can often choose from many equivalent lock sequences. For example, to apply a read lock to the data set LOCATION, the programmer could use any of the following sequences:

```
200   INTEGER N, Stat(9)
210   DIM Lock_set$[16],Lock_item$[16],Q$[40]
220   Lock_item$=”@”
230   Lock_set$=”LOCATION”
240   N=17
250   PACK USING 170;Q$
260   PACKFMT N,Lock_set$,Lock_item$
270   DBLOCK (Base$,Q$,15,Stat(*))
 *
 *
 *
```

or

```
200   Mode=13
210   Lock_set$=”ITEM-MASTER”
220   DBLOCK (Base$,Lock_set$,Mode,Stat(*))
 *
 *
 *
```

or

```
200   PREDICATE P$ FROM ”ITEM-MASTER”,”@”
210   DBLOCK (Base$,P$,16,Stat(*))
 *
 *
 *
```

Multiple descriptors can be combined in a predicate to specify complex locks. Descriptors can be combined by concatenation or by specifying multiple set-item-value groups on the PREDICATE statement. The following example locks the data set LOCATION as well as all values of PRODUCT-NO less than or equal to 10,000 in the data set PRODUCT.

```
110   INTEGER Lock_itemnum,Stat(9)
```

```
120   DIM Q$[40],Lock_set$(1:2)[16],Relop$[2]
130   Lock_item$=”@”
140   Lock_itemnum=14        ! ITEM #14 for PRODUCT-NO.
150   Lock_set$(1)=”LOCATION   “
166   Lock_set$(2)=”PRODUCT    “
170   Relop$=”<=”
180   Keyinfo=10000
190   Mode=5
200   PREDICATE Q$ FROM Lock_set$(1),Lock_item$;Lockset$(2),
Lock_itemnum,Relop$,Keyinfo
210   DBLOCK (Base$,Q$,Mode,Stat(*))
 *
 *
 *
```

The maximum length of a predicate string is 4095 bytes.

Lock Conflicts

The locking system recognizes and acts upon the relationships that implicitly exist between lock descriptors. For example, an attempt to lock a data set must wait if the whole database is locked or if an item/value within the set is locked.

It is also necessary to restrict entry lock requests such that at any one time only a single item name in a data set can be used for locking purposes. Lock requests on different values of the same item are acceptable. However, the locking system cannot determine if the collection of entries locked using different item names have any entries in common. In this case, DBLOCK assumes that a conflict exists and queues the later request or returns a status error.

When a request is queued, no other request which would conflict with the request in queue is granted. For example, assume program A has locked a data entry in data set X, and program B wants to lock data set X and data set Y. Program B’s request is queued. When program C requests to lock a data entry in data set Y, that request is queued because program B is waiting to lock data set Y.

When using DBLOCK wait modes, the programmer should be careful to avoid possible deadlock conditions. If a program makes multiple resource requests using the commands DBLOCK, LOCK# or REQUEST, a potential deadlock can occur if another program is also making requests for the same resources. For example, suppose program A holds resource 1 and is queued waiting for resource 2 which is held by program B. If program B makes a wait request for resource 1, a deadlock situation occurs. Programs requesting the same resources can avoid deadlock by making their resource requests in the same order.

The following table summarizes the conditions for granting a lock:

Table 9 **Lock conditions**

Lock Request	Conflicting Lock	Action
Whole database-write-lock	Whole database already write-locked.	Wait
	Whole database already read-locked.	Wait
	One or more sets write-locked.	Block and wait *
	One or more sets read-locked.	Block and wait *
	One or more item/values write-locked.	Block and wait *
	One or more item/values read-locked.	Block and wait *
Whole database-read-lock	Whole database already write-locked.	Wait
	Whole database already read-locked.	Grant lock
	One or more sets write-locked.	Block and wait *
	One or more sets read-locked.	Grant lock
	One or more item/values write-locked.	Block and wait *
	One or more item/values read-locked.	Grant lock
Whole data set-write-lock	Whole database already write-locked.	Wait
	Whole database already read-locked.	Wait
	Requested set write-locked.	Wait
	Requested set read-locked.	Wait
	One or more item/values in requested set write-locked.	Block and wait *
	One or more item/values in requested set read-locked.	Block and wait *
Whole data set-read-lock	Whole database already write-locked.	Wait
	Whole database already read-locked.	Grant lock

Table 9 **Lock conditions**

Lock Request	Conflicting Lock	Action
	Requested set write-locked.	Wait
	Requested set read-locked.	Grant lock
	One or more item/values in requested set write-locked.	Block and wait *
	One or more item/values in requested set read-locked.	Grant lock

* “Block” means that no more locks capable of impending the request will be granted.

Table 10 **Predicate Lock conditions**

Lock Request	Conflicting Lock	Action
Item/value inset write-lock	Whole database already write-locked.	Wait
	Whole database already read-locked.	Wait
	Set write-locked.	Wait
	Set read-locked.	Wait
	Requested item/value write-locked.	Wait
	Requested item/value read-locked.	Wait
	Different item in set write-locked.	Block and wait *
	Different item in set read-locked.	Block and wait *
Item/value inset read-lock	Whole database already write-locked.	Wait
	Whole database already read-locked.	Grant lock
	Set write-locked.	Wait
	Set read-locked.	Grant lock
	Requested item/value write-locked.	Wait
	Requested item/value read-locked.	Grant lock

Table 10 **Predicate Lock conditions**

Lock Request	Conflicting Lock	Action
	Different item in set write-locked.	Block and wait *
	Different item in set read-locked.	Grant lock

* “Block” means that no more locks capable of impending the request will be granted.

A

Pack Statements

Introduction

The Pack statements provide a convenient means of transferring string and numeric data to and from a string variable. The UNPACK USING statement is particularly useful in conjunction with certain DBINFO modes, in which database information is returned in a string variable as a combination of ASCII characters and numeric integers.

Three Pack statements are available:

PACKFMT Specifies the data format for a PACK USING or UNPACK USING statement.

PACK USING Transfers data from variables in a pack list to a destination string.

UNPACK USING Transfers data from a source string to variables in a pack list.

The PACKFMT Statement

PACKFMT *pack list*

The parameter is as follows:

pack list A list of program variables, arrays, and/or skip fields separated by commas. This list contains an ordered set of variable names used by the PACK USING and UNPACK USING statements.

PACKFMT (pack format) defines a list of variables to be used in conjunction with a source or destination string referenced in an UNPACK USING or a PACK USING statement. Upon PACK USING execution, data is transferred from the PACK USING list variables to the destination string. Upon UNPACK USING execution, data is transferred from the source string to the pack list variables.

As the transfer occurs between the pack list variables and the string referenced in PACK USING or UNPACK USING, an internal pointer to the string's next position is updated. To skip character positions within the string, a skip indicator may be supplied in the appropriate position of the pack list (for example, **1X**=one byte, **2X**=two bytes). The PACK USING and UNPACK USING program examples, on the following pages, illustrate the use of the PACKFMT statement and the skip indicator.

The PACK USING Statement

PACK USING *line id*; *destination string*

The parameters are as follows:

line id A line number or line label referencing the pack list.

destination string A string variable that receives data contained in variables listed in a PACKFMT statement.

The PACK USING statement transfers data from each variable of the appropriate pack list to the destination string. The pack list is located on a separate program line. As the data is transferred to the destination string, its format is not altered in any way. Thus, a real-precision number requires eight bytes in the buffer; a short-precision number requires four bytes; and so on. When transferring a string from the pack list having a current length less than its dimensioned (or substring) length, the destination string is filled with blanks to equal the dimensioned (or substring) length.

The following example illustrates the use of the PACK USING statement:

```
10   INTEGER A
20   SHORT B
30   REAL C
40   DIM D$[10],E$[50]
50   A=47
60   B=89.5432
70   C=2.3456789
80   D$=”IMAGE”
90   PACK USING Here;E$
100  END
110  Here:  PACKFMT A,B,2X,D$,14X,C
```

After executing line 90, E$ contains the following:

Bytes	Value
1-2	Integer variable A
3-6	Short variable B
7-8	Skipped
9-18	String variable D$(Last 5 bytes are padded with spaces)
19-32	Skipped
33-40	Real variable C
41-50	Filled with spaces

Example with User Defined Types:

```
10 DIM T AS Type
 *
 *
 *
200  PACK USING Here;F$
 *
 *
230  Here:  PACKFMT STRUCT T
```

The UNPACK USING Statement

UNPACK USING *line id*; *source string*

The parameters are as follows:

line id A line number or line label referencing a PACKFMT statement.

source string A string expression that contains data to be unpacked into variables listed in a PACKFMT statement.

Through the UNPACK USING statement, data is transferred from the source string to the variables appearing in the pack list. A one-to-one transfer is done without altering the data format. When transferring data to short- or real-precision variables, UNPACK USING verifies that the data qualifies as a valid numeric value.

The following example illustrates the use of the UNPACK USING statement:

```
10   DIM A$[12],F$[40]
20   INTEGER X1,X2
 *
 *
 *
200  UNPACK USING Here;F$
210  DISP X1,A$,X2
220  END
230  Here:  PACKFMT 4X,X1,A$,X2
```

After executing line 200, variables X1, X2 and A$ contain the following information:

Variable	Bytes in F$
X1	5-6
X2	19-20
A$	7-18

Example with User Defined Types:

```
10 DIM T AS Type
 *
 *
 *
200  UNPACK USING Here;F$
210  PRINT STRUCT T
220  END
230  Here:  PACKFMT STRUCT T
```

B

DBML Syntax

Schema Definition

```
BEGIN DATABASE database name;
DEFAULT LANGUAGE collating sequence;
PASSWORDS:
   user-class number password;

ITEMS:
   item name,[sub-item count]specifier[(control no.)];
   . . .
IITEMS:
   iitem name = item name[:length][,item name...];
   . . .
SETS:
   list of data set definitions
END.
```

Schema Definition Parameters

database name 1 through 6 character database name, beginning with a letter and containing uppercase letters, digits and dashes.

collating sequence A language plus a modifier as your default collating sequence (optional).

user-class number An integer from 1 through 31.

password A string of from 1 through 8 characters not including the semi-colon, tabs, blanks. Embedded blanks are removed.

item name A 1 through 15 character item name, beginning with a letter and containing uppercase letters, digits and dashes.

sub-item count An integer which specifies the replication count for the item whose specifier it precedes.

specifier A specifier of the item type. It is either **L**, **R8**, **S**, **R4**, **I**, I2, or **X**. In the case of **X**, it is followed by an integer specifying the string length.

iitem name A 1 through 15 character iitem name, beginning with a letter and containing uppercase letters, digits and dashes.

length Integer specifying substring length of the item to be used as part of the index.

control no. An integer from 0 through 127. This number can be retrieved by DBINFO. It is used by Eloquence Query to determine the format used to print any data associated with that item.

Data Set Definition Syntax

Manual Master Data Set Definition

{NAME: / N:} *set name*, {MANUAL / M} (*read list/write list*);

{ENTRY: / E:} *item name* [(*path count*)] ,

```
[item name,]
[item name,]
   ...
[item name];
```

{CAPACITY: / C:} *maximum-entry count*;

Automatic Master Data Set Definition

{NAME: / N:} *set name*, {AUTOMATIC / A} (*read list/write list*);

{ENTRY: / E:} *item name* (*path count*);

{CAPACITY: / C:} *maximum-entry count*;

Detail Data Set Definition

{NAME: | N:} *set name*, {DETAIL | *D*} (*read list/write list*);
{ENTRY: | E:} *item name* [(*master-set name*)] ,

```
[item name[(master-set name)],]
[item name[(master-set name)],]
      . . .
[item name[(master-set name)]];
```

{CAPACITY: | C:} *maximum-entry count*;

Data Set Definition Parameters

set name A 1 through 15 character data set name, beginning with a letter and consisting of uppercase letters, digits and dashes.

read list A list of user-class numbers (including 0) separated by commas. The list can be null.

write list A list of user-class numbers (including 0) separated by commas. The list cannot be null.

path count An integer from 1 through 16 corresponding to the number of paths between this master and the associated detail sets.

maximum-entry count An integer.

master-set name The name of a previously listed master data set.

collating sequence Slash followed by language name and (optionally) a modifier:

```
/language[@modifier]
```

DBML Statements and Advanced Access

DBLOGON (*user$, pass$*)

The DBLOGON statement saves the authorization information which should be used subsequently when connecting a database serever.

DBOPEN (*base$, pass$, status(*)*)

The DBOPEN statement opens the database for access and defines the type of access allowed (for example, read-only, exclusive, or shared).

$$\text{DBCLOSE } (base\$, \left\{ \begin{matrix} set \\ set\$ \end{matrix} \right\}, mode, status\ (*))$$

The DBCLOSE statement closes the database and updates all information in the root file. The set parameter is ignored.

$$\text{DBGET } (base\$, \left\{ \begin{matrix} set \\ set\$ \end{matrix} \right\}, mode, status(*), list\$, buf\$, \left\{ \begin{matrix} arg \\ arg\$ \end{matrix} \right\})$$

The DBGET statement is used to retrieve information from the database. If an IN DATA SET is active on the specified set, buf$ will be unpacked into the appropriate variables.

$$\text{DBUPDATE } (base\$, \left\{ \begin{matrix} set \\ set\$ \end{matrix} \right\}, mode, status(*), list\$, buf\$)$$

The DBUPDATE statement is used to modify values in an existing record in the database (search item values cannot be modified). If an IN DATA SET is active on the specified data set, buf$ will be updated with the current values of the appropriate variables before the operation.

$$\text{DBPUT } (base\$, \left\{ \begin{matrix} set \\ set\$ \end{matrix} \right\}, mode, status(*), list\$, buf\$)$$

The DBPUT statement is used to add new entries to sets of type detail or manual. If an IN DATA SET is active on the specified set, buf$ will be updated with the current values of the appropriate variables before the operation.

DBDELETE (*base$*, {*set* / *set$*}, *mode*, *status*(*))

The DBDELETE statement is used to remove entries from sets of type detail or manual.

DBFIND (*base$*, {*set* / *set$*}, *mode*, *status*(*), {*item* / *item$*}, {*value* / *value$*})

The DBFIND statement is used to find the head of a chain in a detail data set.

DBINFO (*base$*, {*qual* / *qual$*}, *mode*, *status*(*), *buf$*)

The DBINFO statement provides general information about the database.

DBLOCK (*base$*, {*set* / *set$* / *predicate$*}, *mode*, *status*(*))

The DBLOCK statement locks the entire database or sections of the database so modifications can be performed when the database is open in shared mode.

DBUNLOCK (*base$*, {*set* / *set$*}, *mode*, *status*(*))

The DBUNLOCK statement unlocks a database or section of the database that was locked with a previous DBLOCK.

PREDICATE *predicate$* FROM set_1 [,$item_1$ [,*relop*, *value*]] [;set_2 . . .[;set_n]]

The PREDICATE statement defines a section of the database to be locked with the DBLOCK statement.

DBBEGIN (*comment$, mode, status(*)*)

The DBBEGIN statement begins a new (sub-) transaction. When this is the first transaction, it is called top level transaction. No modifications are permanently saved in the Eloquence A.06.00 data base until the top level transaction is committed. A subsequent DBBEGIN begins a new subtransaction, which can be controlled separately with the DBCOMMIT and DBROLLBACK statements.

DBCOMMIT (*mode, status(*)*)

The DBCOMMIT statement commits a transaction. If this is a top level transaction, modifications are made permanently in the data base. If a subtransaction is committed, it becomes part of its parent transaction.

DBROLLBACK (*id, mode, status(*)*)

The DBROLLBACK statement is used to undo a pending transaction. If this is a top level transaction, all pending modifications are reverted. If applied to a subtransaction all modifications including the enclosing DBBEGIN statement are reverted.

Msg$ = DBEXPLAIN$(*n*)
Msg$ = DBEXPLAIN$(*status(*)*)

The DBEXPLAIN$ function translates the given status into a descriptive message.

DBASE IS *base$*

The DBASE IS statement is used to specify the database before any IN DATA SETs.

IN DATA SET { *set* / *set $* } [IN COM] { USE ALL / USE *item list* / DIM ALL / USE REMOTE LISTS *line id list* / USE STRUCT *variable* / DEFINE TYPE *typename* }

IN DATA SET { *set* / *set$* } FREE

The IN DATA SET statement defines the automatic packing or unpacking procedure to be performed. Packing or unpacking of the buffer string is performed whenever a DBGET, DBUPDATE or DBPUT is executed on the specified data set (of the database specified by the last DBASE IS statement). The FREE option allows the automatic packing and unpacking to be turned off.

The IN DATA SET LIST statement is a non-executable statement which is referenced by an IN DATA SET with the USE REMOTE LISTS option. This option is used when the USE list is too long to store as one program line.

The IN DATA SET ... DEFINE TYPE statement can be used to define a type from a data set definition.

DBML Statement Parameters

base$	A string variable which contains the database name.
pass$	A string expression containing a left-justified string.
set	A numeric expression evaluating to a data set number.
set$	A string expression evaluating to a data set name.
mode	A numeric expression evaluating to a valid mode.
status	An integer array containing at least 10 elements in right-most dimension, used to return return codes on most DBML statements.
list$	A string expression evaluating to either “@ “, “@;” or “@”. In all but the first case, any arbitrary character sequence can also follow.
buf$	A string variable, without any substring specifiers, which is used to transfer information between an Eloquence program and a database.
qual	A numeric expression evaluating to a valid item, set, or volume number.
qual$	A string expression evaluating to a valid item, set, or volume label.
item list	A list of string or numeric variables (or arrays) which correspond to items in the data set specified in an IN DATA SET statement.
line list	A list of line numbers or labels which appears in an IN DATA SET…USE REMOTE LISTS statement. Each line id must refer to an IN DATA SET LIST statement.
predicate$	A string variable returned by PREDICATE and used as the qualifier parameter by DBLOCK.
value	A string or numeric expression giving the value of the item to be locked.
item	A string expression specifying the data item within the set to be locked.
comment	A string expression providing a comment for the transaction. This string is recorded in the transaction log, however there are currently no tools to review them.

Utility Statements

DBCREATE *base$* [;*maint$*] [,*set list$*] [,*return var*]

The DBCREATE statement creates the data sets associated with a root file. Options are available for creating either all sets or only specific sets. A maintenance password may be specified for the first time DBCREATE is used to define a password. This password must be used on all subsequent accesses to the database via utilities statements.

DBERASE *base$* [;*maint$*] [,*set list$*] [,*return var*]

The DBERASE statement erases all data sets or any group of data sets in the database.

DBPURGE *base$* [;*maint$*] [,*set list$*] [,*return var*]

The DBPURGE statement purges either all data sets or any group of data sets in the database.

Utility Statement Parameters

base$	A string expression evaluating to the database name. An optional volume label or unit specifier can be appended to the database name.
maint$	A string expression evaluating to the maintenance password.
set list$	A string expression identifying particular data sets. Data sets are specified by either name or number. Set identifiers are separated by commas.
vol spec$	A string expression evaluating to a volume label or unit specifier.
return var	A numeric variable in which an error number is returned (refer to page 197). 0 is returned if no error occurs.

Obsolete utility statements

The following statements are no longer supported as of Eloquence A.06.00. A runtime error 1004 is generated when they are encountered.

DBSTORE [*base$* [TO *vol spec$*]]

The DBSTORE statement calls the HP-UX script file “dbstore” which backs up all data sets or any group of data sets in the database.

DBRESTORE [*base$* [FROM *vol spec$*]]

The DBRESTORE statement calls the HP-UX script file “dbrestore” which restores the database using data stored previously by DBSTORE.

DBPASS *base$*, *user-class number*, *old password* TO *new password*

This statement changes the password for a stated user-class number.

DBMAINT *base$*, *old word* TO *new word*

This statement changes the maintenance password for a stated database. The old word and new word parameters are string expressions from 0 through 16 characters, excluding nulls and spaces. The old word specified must match the current maintenance word for the database. The maintenance word is established when the root file is created via the DBCREATE statement.

This statement reads all used passwords from the specified database into a string array.

READ DBPASSWORD *base$, maint$*; *string array variable*

This statement re-assigns all passwords in the specified root file with those in a specified string array.

READ DBPASSWORD *base$, maint$*; *string array variable*

C

Error Messages

This appendix describes all error numbers and associated messages for the following components of Eloquence DBMS:

- Database manipulation status errors.
- Pack and Eloquence DBMS execution errors.
- dbimport errors.

Eloquence DBMS Status Errors

The following list describes the condition word values for Eloquence DBMS programming statements:

0	Successful execution - no error.
-1	No such database. Database is currently opened in an incompatible mode. Bad root file reference. Database opened exclusively.
-2	Database in use (Eloquence library only)
-3	User name not recognized
-4	User password does not match
-7	Database lock request was already made in current environment.
-10	User may not open additional databases, ten are already opened.
-11	Bad database name or preceding blanks missing.
-12	DBPUT, DBDELETE, or DBUPDATE called with database not locked.
-14	DBPUT, DBDELETE, and DBUPDATE not allowed in access mode 8.
-21	Bad password - grants access to nothing. Data item nonexistent or inaccessible. Data set nonexistent or inaccessible. Data set volume nonexistent.
-23	User lacks write access to data set.
-24	DBPUT, DBDELETE, or DBUPDATE not allowed on automatic master.
-31	Bad mode. DBGET mode 5 - Specified data set lacks chains. DBGET mode 7 - Illegal for detail data set.
-52	Item specified is not an accessible search item in the specified set. Bad LIST variable - must be @$\triangle$ or @; or @. ($\triangle$ indicates blanks)

-91	Root file not compatible with current Eloquence DBMS statements.
-92	Database requires creation.
-94	Data or structure information lost. Database must be erased or re-created.
-95	No automatic master set entry for current detail. DBDELETE only.
-96	Corrupt pointer value detected in current data set.
-120	Not enough memory to perform DBLOCK.
-122	Descriptor list bad. Not within string limits.
-123	Illegal relational operator.
-124	Descriptor too short; must be greater than or equal to 9 words.
-125	Bad set name or number.
-126	Bad item name or number.
-127	Attempt to lock using a compound item.
-128	Bad descriptor length for numeric item.
-124	Two descriptors conflict.
-135	Second lock is not allowed in modes 1, 3, 5, 11, 13, and 15.
-136	Descriptor list exceeds 4092 bytes.
-137	Qualifier parameter is of wrong type.
- 800	ISAM error. Last element of status array contains ISAM error number.
- 801	Volume failure.
- 802	Node related failure.
- 803	FixRec related problem.
- 804	BTREE related problem.
- 805	SysCat related problem.
- 806	System call.
11	End-of-file.

12	Directed beginning of file.
13	Directed end of file.
15	End of chain.
16	The data set is full.
17	There is no chain for the search item value. There is no entry with the specified key value. No current record or the current record is empty. The selected record is empty.
18	Broken chain.
20	Database locked or contains locks. Status word 3: 0 - database locked. 1 - data set or entries locked.
22	Data set locked by another process.
23	Entries locked within set.
24	Item conflicts with current locks.
25	Entry or entries already locked.
27	Relational operator type conflict.
32	Transaction nesting exceeds maximum.
33	No transaction active.
41	DBUPDATE will not alter a search item.
43	Duplicate key value in Master.
44	Cannot delete a Master entry with non-empty Detail chains.
50	User's buffer is too small for requested data.
53	ARGUMENT field type incompatible with search field type (DBGET, mode 7, or DBFIND). ARGUMENT's current string length is less than the string length of the search field.
80	Data set volume is not on-line, or user has not the requested access rights.
90	Root file volume is not on-line.
94	Corrupt database opened successfully in mode 8.
1xx	There is no chain head for path xx.
3xx	The automatic master for path xx is full.

4xx The master data set for path xx is not currently mounted (applies to DBPUT and DBDELETE for detail data sets).

Status Array Contents Following an Error

The contents of the status array following an Eloquence DBMS error (a non-zero condition word) are listed below.

Table 11 **Eloquence DBMS status array**

Word	Description
1	Condition Word
2 through 4	Unchanged
5	0
6	Bits 0 through 11: an id number from 401 through 410 (see table below) Bits 12 through 15: 0 or the mode value used to open the database
7	Program line number
8	0
9	The mode parameter value
10	Reserved

An identification number is associated with each DBML statement as shown below.

Table 12 **Eloquence DBMS identification numbers**

ID Number	DBML Statement
401	DBOPEN
402	DBINFO
403	DBCLOSE
404	DBFIND
405	DBGET
406	DBUPDATE

Table 12 **Eloquence DBMS identification numbers**

ID Number	DBML Statement
407	DBPUT
408	DBDELETE
409	DBLOCK
410	DBUNLOCK
411	DBCREATE
412	DBERASE
413	DBPURGE
420	DBLOGON
421	DBBEGIN
422	DBROLLBACK
423	DBCOMMIT

Pack and Eloquence DBMS Error Codes

Error Code	Error Description
200	Referenced line not a PACKFMT.
202	Insufficient dimension length in PACK USING statement, or insufficient current length in UNPACK USING statement.
204	Conversion error.
205	UNPACK USING requires a source string of greater length.
210	Bad status array.
211	No DBASE IS statement active; improper database specified or database is not open.
212	Data set not found.
213	Excessive variables in list.
214	IN DATA SET already active for data set.
215	Number of elements does not match.
216	Variable type does not match with associated field in set.
217	String length in list insufficient, or length of list array greater than 255 bytes.
218	Variables not in common.
219	Line referenced is not an IN DATA SET LIST statement.
220	Improper or illegal use of maintenance word.
221	Data set not created.*
225	Improper utility version number in root file.
226	Corrupt database - must recreate it.
227	Corrupt database - must erase it.
320	Set or item specifier is out of range or is an invalid set or item name.
321	Relational operator is invalid.

322 The predicate specified is not a valid form.

* When executing DBCREATE, DBERASE, or DBPURGE from the keyboard without a return variable, errors 54, 56, 64, 77, and 221 are not fatal. They are logged on the CRT along with information on the set to which the error pertains.

dbimport Error Messages

This is a subset of the dbimport error messages:

illegal arguments

bad option(s) specified

no database specified

could be missing option parameter

unable to setup database system

unable to connect to eloquence daemon

illegal volume specification

illegal device specification

illegal database name

bad format: # DATABASE = expected

single file export files must start with the sequence above

importing from database

if importing from single file, dbimport will notify you if the database name in import file differs from actual one.

Illegal set name

you entered an improper set name on commandline

Messages during data base loading

INTERRUPT

you interrupted dbimport

Fatal error # *number* while *operation*

a database status error (number) occured during given operation

Secondary status

this line will be present if an ISAM call failed

bad format: # SET *no* = *name* expected

bad format: # *item index item name item type* expected

Lines starting with a '#' character will be used by dbimport to retrieve field names from input file or to delimit between import sets.

Number of fields exceeds maximum

There is a limit of 2048 field names per set.

set not in backup file

Specified data set name or number not in backup file.

No field names for this set in import file

you specified field names in restructure file, but no field names are present in import file for this data set

Field name not found

field name specified in restructure file not found in import file

Too many fields to process

import rules too complex. There is a limit of 2048 rules per set.

Illegal type conversion

item type conflicts with value found in import file. Values (from import file) are converted automaticly if processing in restructure mode. If not in restructure mode this is considered an error.

Number of values exceeds maximum

Unexpected end-of-line

Value buffer overflow

Missing quote

Illegal numeric value

probably bad input data

Messages during parsing of restructuring specification

Duplicate set specification

duplicate data set restructuring specification

from set number must be specified unless importing from single file

dbimport could only resolve data set names from import file if importing from a single import file

Illegal *from* specification

improper backup set number

Set name too long

Illegal set name

Undefined set name

Improper set number

improper or undefined data set name or number

Item name too long

Illegal item name

improper field or item name

Undefined item name

Improper item number

improper or undefined item name or number

Number of elements do not match

Index exceeds subitem count

Improper index

Improper index range

improper item element

ISAM Errors

The following values are specific to the previous Eloquence database implementation and can only happen when using eloqdb5. The Eloquence A.06.00 uses different values to indicate internal problems.

If you receive a database status -800, this indicates an internal failure, usually related to the internal ISAM subsystem. In this case the last element of the status array provides additional information.

Any value below 100 indicates an error returned by a HP-UX system call. Please refer to /usr/include/sys/errno.h for more information, Numbers above 100 indicate an ISAM related problem.

Error Number	Description
100	An attempt was made to add a duplicate value to an index via iswrite, isrewrite, isrewcurr, or isaddindex.
101	An attempt was made to perform some operation on an ISAM file that was not previously opened using the isopen call.
102	One of the arguments of the ISAM call is not within the range of acceptable values for that argument.
103	One or more of the elements that make up the key description is outside of the range of acceptable values for that element.
104	The maximum number of files that may be open at one time would be exceeded if this request were processed.
105	The format of the ISAM file has been corrupted.
106	In order to add or delete an index, the file must have been opened with exclusive access.
107	The record or file requested by this call cannot be accessed because it has been locked by another user.
108	An attempt was made to add an index that has been defined previously.
109	An attempt was made to delete the primary key value. The primary key may not be deleted by the isdelindex call.
110	The beginning or end of file was reached.

111	No record could be found that contained the requested value in the specified position.
112	This call must operate on the current record. The current record is not defined.
113	The file is exclusively locked by another user.
114	The filename is too long.
115	The lock file cannot be created.
116	Adequate memory cannot be allocated.
126	Bad record number.
127	No primary key.
131	No free disk space.
132	Record too long.
	All other numbers indicate an internal error, and should be reported to HP. Examples:
201	Bad request.
202	Too far.
203	No info.
204	Not set.
205	Bad info.

D

Eloquence Library

This appendix describes how you can integrate Eloquence DBMS functions with your own programs.

You should have had previous experience with Eloquence DBMS and with the C programming language.

Compilation and Linking

Compilation

Each program in which you want to include programs from the Eloquence libraries must have header-file /opt/eloquence6/include/eloqdb.h inserted at the beginning.

```
...
#include <eloqdb.h>
...
```

Linking

When linking the program you must enter **`-l eloq`** to include access to the Eloquence DBMS library /opt/eloquence6/lib/libeloq.a.

On the Windows platform, you need the to install the eloqdb.dll and should link with the eloqdb.lib import library.

ELOQLIB functions

Below you will find the reference specifications for the Eloqlib functions arranged by function name in the following order:

Function	Description
INIT	Initialize/configure database subsystem
EXIT	Terminate database subsystem
ERROR	Return error description
LOGON	Provide authorization information
OPEN	Open database
CLOSE	Terminate database access
DELETE	Delete entry from data set
FIND	Set up chain or index
GET	Retrieve data from data set
INFO	Return structural information
LOCK	Lock database, data set or data item
UNLOCK	Unlock database
PUT	Add entry to data set
UPDATE	Update entry in data set
BEGIN	Begin (sub-) transaction
COMMIT	Commit Transaction
ROLLBACK	Rollback transaction.

The INIT Function

The INIT function initializes and configures a database subsystem. The syntax is as follows:

```
int cc = idb_init(max_db, max_path, max_buf, max_open);

int max_db;
int max_path;
int max_buf;
int max_open;
```

All arguments are ignored and are only present for compatibility with the previous implementation of the database library.

Description

The Idb_init function will initialize the database subsystem. It is called is implicitly when the first database is opened.

All function arguments are ignored and are only present to achieve source and binary (in case of eloqdb.dll) compatibility with the previous implementation. You should pass a zero value for all arguments.

Return value

Returns 0 if successful, or -1 if an error was encountered.

Example

```
if(idb_init(0,0,0,0)) {
   printf(”unable to setup database subsystem”);
   exit(1);
}
```

The EXIT Function

The EXIT function terminates a database subsystem. The syntax is as follows:

```
int cc = idb_exit()
```

The parameters are: None

Description

The idb_exit function call will terminate access to a database subsystem. All databases currently opened are closed. All resources set up by idb_init() are reset.

Return value

Returns 0 if successful, or -1 if an error was encountered.

Example

```
if(idb_exit()) {
   printf("unable to terminate database subsystem");
   exit(1);
}
```

This will terminate access to a database subsystem.

The ERROR Function

The ERROR function interprets the status array and returns the corresponding error or warning message. The syntax is as follows:

```
int cc = idb_error(status,buffer,length)

int status[10];
char buffer[80];
int *length;
```

The parameters are:

status	The status parameter of any failed function call (see below).
buffer	A pointer to a character array of at least 80 bytes. Used to return the message. The buffer will terminate with backslash 0 (‘\0’).
length	A NULL pointer, or a pointer to an integer to return the byte length of the message returned in buffer (except ‘\0’).

Description

The idb_error function call returns a message explaining the specified status code. The message is returned to the buffer specified in the idb_error call.

Return value

Returns 0 if successful, or -1 if an error was encountered.

Example

```
char buf[80];
int status[10];

...

if(idb_error(status, buf, NULL)) {
   printf(”unable to analyze status”);
   exit(1);
}
```

The above will get a descriptive message for a failed idb function.

The LOGON Function

The DBLOGON is used to provide authorization information which is used when connecting the database server.

The syntax is as follows:

```
int cc = idb_logon(user, passwd)

char *user;
void *passwd;
```

The parameters are:

user A pointer to a character array which specifies the user id which should be used to authorize database access. The user id must be terminated with a ‘\0’ character. When a NULL pointer or an empty user id is passed, the server will default to the user name "default".

passwd A pointer to a character array containing the password for the user id. The password must be terminated with a ‘\0’ character. When a NULL pointer or an empty password is passed, the server will assume no password.

The authorization information is transmitted to a server when a database is opened. This has no effect when used with the eloqdb5 server.

Description

The idb_logon function call is used to provide authorization data which are transmitted to the data base server, when a data base is openend. For Eloquence A.05.xx data bases (accessed through the eloqdb5 server), the logon information is ignored.

Return value

Returns 0 if successful, or -1 if an error was encountered.

Example

```
char *user = "mike";
char *pswd = "secret";
if(log_on(user, pswd)) {
   printf(”unable to save authorization information”);
   exit(1);
}
```

The OPEN Function

The OPEN function initiates access to a database.

Status Codes

```
int dbid = idb_open(base,passwd,mode,status)

char *base;
void *passwd;
int mode;
int status[10]
```

The parameters are:

base Specifies the database to be opened. *Base* is a pointer to a character array containing up to 256 characters (bytes). The database name must be terminated with a '\0' character.

passwd A pointer to a character array containing the password. If the password is less than 16 characters long, it must be terminated by a semicolon, a blank or a '0' character. The password establishes the security class number which defines the data sets and data items to which the calling program has read, update and/or write access.

The password is only used when connecting an eloqdb5 server. With eloqdb6, the password is ignored and authorization information is passwd with idb_logon().

mode Specifies type of database access. See below.

status A pointer to an array of a least 10 elements used to indicate the success or failure of the function call (see below).

IDBOPEN Modes

Mode 1: Modify shared

Data entries can be read, updated, added or deleted. This mode can be used if all concurrent users have opened the database in either mode 1 or mode 9.

Mode 3: Modify exclusive

Data entries can be read, updated, added or deleted. This mode can be used only if the database is not already opened by another user.

Mode 8: Read shared, only concurrent read allowed

Data entries can only be read. Updating, adding or deleting are not permitted by this user or any other user. This mode can be used only if all concurrent users have opened the database in either mode 8 or mode 9.

This mode allows you to access the database even if the database is recognized to be inconsistent.

Mode 9: Read shared

Data entries can only be read. Updating, adding or deleting are not permitted by this user. This mode can be used only, if all concurrent users have opened the database in mode 1, mode 8 or mode 9.

Description

idb_open initiates access to a database, establishes the security class number of the calling program, and establishes the type of access for all subsequent operations on the database.

The *password* parameter is not checked if user is the superuser (root).

Return value

Returns 0 if successful, or error number if an error was encounterd.

Status codes

If idb-open was successfully executed, the status array will contain the following values:

Table 13 **Status codes**

Element	Meaning
0	S_OK
1	user class
2	0
3	0
4	0
5	DB_OPEN | (open_mode << 12)
6	0

Table 13 **Status codes**

Element	Meaning
7	0
8	mode
9	database id

Example

```
if((dbid = idb_open("SAD","MANAGER",1,status)) %< 0) {
   printf("unable to open SAD database\\n");
   exit(1);
}
```

This will open database SAD.

The CLOSE Function

The CLOSE funtion terminates access to a database or resets the current record for a data set. The syntax is as follows:

```
int cc = idb_close(base,dset,mode,status)

int base;
void *dset;
int mode;
int status[10];
```

The parameters are:

base Identifies the database. This must be the return value from the idb_open call.

dset This parameter is ignored if mode is not equal to 3. Identifies the data set to be rewound. Dset is one of the following:

- a pointer to an integer variable that specifies the data set number;
- a pointer to a character array containing up to 16 characters (bytes) that specifies the data set name. The data set name must be terminated with a semicolon, a blank or \0 character if it is less than 16 characters.

mode See below.

status A pointer to an array of a least 10 elements used to indicate the success or failure of the function call.

IDBCLOSE Modes

Mode 1. Close database

Terminate access to the database and deallocate resources.

Mode 3. Rewind data set

The current record of the given data set is reset to zero.

Description

Idb_close terminates access to a database or resets current record for a data set. If a program has issued multiple calls to idb_open for the same database, only the access path of the given base parameter is affected.

Idb_close does the following:

- If idb_close is called with mode 1, access to all data sets in the specified database is terminated. Resources allocated for the database are deallocated.
- If idb_close is called with mode 3, the current record for the given data set is reset to zero.

Return value

Returns 0 if successful, or an error number if an error was encountered.

Status codes

If idb_close was successfully executed, the status array will contain the following values:

Table 14 **Status codes**

Element	Meaning
0	S_OK
1	0
2	0
3	0
4	0
5	`DB_CLOSE | (open_mode << 12)`
6	0
7	`0`
8	mode
9	`0`

Example

```
if(idb_close(dbid,"CUSTOMER",3,status))
   error_handler();
```

This will rewind CUSTOMER data set.

The DELETE Function

The DELETE function deletes an entry from the database. The syntax is as follows:

```
idb_delete(base,dset,mode,status)

int base;
void *dset;
int mode;
int status[10]
```

The parameters are:

base Identifies the database. This must be the return value from the idb_open call.

dset Identifies the data set in which the entries are to be located. Dset is one of the following:

- a pointer to an integer variable that specifies the data set number;
- a pointer to a character array containing up to 16 characters (bytes) that specifies the data set name. The data set name must be terminated with a semicolon, a blank or 0 character if it is less than 16 characters.

mode Must be 1.

status A pointer to an array of a least 10 elements used to indicate the success or failure of the function call (see below).

Description

Idb_delete deletes the current record from the specified data set. The database must be open in either mode 1 or mode 3. The security class number must have write access to the data set specified in dset.

If the entry being deleted is part of a chain, all links are automatically maintained. If the entry is the last entry in a chain and the chain is linked to an automatic master, the entry in the automatic master is deleted unless it is linked to any other chains. An entry in a manual master can only be deleted if all linked child entries are deleted first.

Idb_delete does not affect the current chain or the chain information in the status parameter.

Return value

Returns 0 if successful, or error number if an error was encountered.

Status codes

If idb_delete was successfully executed, the status array will contain the following values:

```
[0] = S_OK    [6] = 0
[1] = record length   [7] = backward address
[2] = 0     [8] = 0
[3] = record number   [9] = forward address
[4] = 0
[5] = 0 if a detail set, 1 if master set
```

Example

```
if(idb_delete(dbid,"CUSTOMER",1,status))
   error_handler();
```

This will delete the current record from the CUSTOMER data set.

The FIND Function

The FIND function establishes the current chain or index in preparation for access. The syntax is as follows:

```
idb_find(base,dset,mode,status,item,arg)

int base;
void *dset;
int mode;
int status[10]
void *item;
void *arg;
```

The parameters are:

base Identifies the database. This must be the return value from the idb_open call.

dset Identifies the data set in which the entries are to be located. Dset is one of the following:

- a pointer to an integer variable that specifies the data set number;
- a pointer to an character array containing up to 16 characters (bytes) that specifies the data set name. The data set name must be terminated with a semicolon, a blank or 0 character if it is less than 16 characters.

mode One of the access modes. See below.

status A pointer to an array of a least 10 elements used to indicate the success or failure of the function call.

item If the mode is 1, *item* identifies the search item to be used. It can be one of the following:

- a pointer to an integer variable that specifies the search item number;
- a pointer to a character array containing up to 16 characters (bytes) that specifies the search item name. The search item name must be terminated with a semicolon, a blank or 0 character if it is less than 16 characters.

If the mode is 2 or 3, *item* identifies the index to be used. It can be one of the following:

- a pointer to an integer variable that specifies the index item number;

- a pointer to a character array containing up to 16 characters (bytes) that specifies the index item name. The index item name must be terminated with a semicolon, a blank or 0 character if it is less than 16 characters.

If the mode is 4 or 5, *item* identifies the index to be used. It can be one of the following:

- a pointer to an integer variable that specifies the index item number;

- a pointer to a character array containing up to 16 characters (bytes) that specifies the index item name. The index item name must be terminated with a semicolon, a blank or 0 character if it is less than 16 characters.

arg

If the mode is 1 and *item* specifies a search item, *arg* contains a pointer to search for the item value to be used to identify the chain.

If mode is 1 and *item* specifies an index, or the mode is 2 or 3, *arg* contains a pointer to index lookup value.

If mode is 4 or 5, *arg* contains a pointer to a regular expression. This is set up in the same way as index lookup values (Modes 2, 3).

Example:

```
struct {
 int len;        /*number of bytes in value area */
 char value[...]; /*the actual key value         */
} arg;
```

IDBFIND Modes

Mode 1: Find chain head/match subset

Idb_find will locate chain head or match subset depending on the *item* parameter. The *item* parameter may be either a search item or an index item.

If the *item* parameter specifies a search item:

The *arg* parameter is a pointer to search for the item value to be used to identify the chain. The length and type of the buffer specified by the *arg* parameter must match the specified item.

If the *item* parameter specifies an index item:

The *arg* parameter is a pointer to the index value to be used to identify the subset. The type of buffer specified by the *arg* parameter must match the specified index in the given length.

NOTE: For performance considerations, accessing indexes using mode 1 is not recommended. In order to return first/last record pointer and number of records, all matching records are read.

Mode 2: Find first entry

Locates the first matching entry in a given index.

The *item* parameter specifies which index to use. The *arg* parameter is a pointer to the index value to be used to identify the subset. The type of buffer specified by the *arg* parameter must match the specified index in given length.

Mode 3: Find last entry

Locates the last matching entry in given index.

The *item* parameter specifies which index to use. The *arg* parameter is a pointer to the index value to be used to identify the subset. The type of buffer specified by the *arg* parameter must match the specified index in given length.

Mode 4: Find first entry with matching regular expression

Locates the first entry matching the regular expression in index order.

The *index* parameter must refer to an index item. The index item must contain at least one leading string segment.

The *arg* parameter is a pointer to the regular expression to be used to identify the subset.

The given regular expression must exactly describe the leading string segments. There is no implicit '*' at the end (as DBFIND Modes 2/3). If you store "AAA " (trailing space) in an entry, you won't find it using a search value of "AAA", but you will find it using "AAA?" or "AAA*".

The entries will be retrieved using DBGET Modes 5/6 in index order.

Mode 5: Find last entry with matching regular expression

Locates the last entry matching the regular expression in index order.

DBFND Mode 5 operates in the same way as Mode 4, except that it locates the last entry.

NOTE: Access time depends on the regular expression given. We do not recommend specifying a character class or a wildcard character at the beginning of the regular expression because this would result in a serial access to the dataset specified.

NOTE: *Status* may return a 0 in the first status array element although there is no matching entry in the dataset specified. A subsequent DBGET will return 15 (end-of-chain) in the first status array element.

Description

idb_find is used with detail data sets to establish a current chain for the specified data set. This is done in preparation for using idb_get with mode 5 or 6, which retrieve entries that belong to the current chain. The current chain is determined by the search item specified in the *item* parameter. Each current chain is associated with a specific data set in the database and is also associated with the relationship defined by the base parameter.

idb_find tries to locate the corresponding entry in the parent set (identified by search item). There may not actually be any entries in the specified data set with a matching search item value. In this case, the idb_find call will execute successfully, but the subsequent idb_get call will return an “end-of-chain” status code.

idb_find is used with indexed access to locate the first or last entry with a matching index value. This is done in preparation for using idb_get with modes 5,6,15 or 16, which retrieve entries in index order. The current index is determined by the index item specified in the *item* parameter.

idb_find verifies that the *item* parameter references an index item for the specified data set. It then locates the argument value in the appropriate index. If the argument paramater has a length value of zero, this is simply the first or last record (in index order). If a matching value cannot be found, the record pointer will be located at the position in the index where the requested value would be inserted.

If the index item segment is of type character (’X’) you can just use a part of it in the index access. However if the index item segment is a numeric one, you have to specify the whole segment or it will be ignored (what’s the first byte of 123.456E-2 ?).

Beware of alignment trap. For example: if you specify a construction like the following:

```
{
   char x[2];
   long d;
}
```

there will be a 2 byte gap between the character array and the long value (longs must be 4 byte aligned). Idb_find will assume that there is no gap between data.

For idb_find to execute successfully, the security class must have at least read access to the data set specified by dset and to the search item specified by item. It is not necessary to have read access to the parent of that data set. Idb_find does not retrieve any data.

Return value

Returns 0 if successful, or error code if an error was encountered.

Status codes

If idb-find was successfully executed, the status array will contain the following values:

Table 15 **Status codes**

Element	Meaning
0	S_OK
1	0
2	0
3	0
4	0
5	number of entries (mode 1 only)
6	0
7	backward address (mode 1)
8	0
9	forward address (mode 1)

Example

```
int itemno;
int product_no;

itemno = 5;           /* PRODUCT-NO item */
product_no = 4711;    /* product number   */

if(idb_find(dbid,"ORDER",1,status,&itemno, &product_no))
   error_handler();
```

This will find the chain head for PRODUCT-NO in ORDER data set.

The following example assumes that an index has been defined for the data set “SAMPLE-SET”, consisting of the following items:

```
ITEM:
   CODE, X2;
   GROUP, I;

IITEM:
   SAMPLE-INDEX = CODE,GROUP;
```

Now let’s try to locate all entries with a code starting with ‘A’ in ascending order.

```
struct {
   int len;
   char code[2];
} sample_key;

sample_key.len = 1;
sample_key.code[0] = 'A';

          if(idb_find(dbid,"SAMPLE-SET",2,status,"SAMPLE-INDEX",
&sample_key))
{
   if(status[0] == S_NOREC) {
      printf("No entry with selected type code");
      ...
   }
   else error_handler();
}

 while(idb_get(dbid, "SAMPLE-SET, 5, status, "@", buffer, 0) ==
S_OK) {
   /* ... got the next entry ... */
}
if(status[0] != S_ENDCHAIN)
   error_handler();
```

The GET Function

The GET funtion retrieves an entry from a data set. The syntax is as follows:

```
idb_get(base,dset,mode,status,list,buffer,arg)

int base;
void *dset;
int mode;
void status[10]
void *list;
char *buffer;
void *arg;
```

The parameters are:

base Identifies the database. This must be the return value from the idb_open call.

dset Identifies the data set in which the entries are to be located. *Dset* is one of the following:

- a pointer to an integer variable that specifies the data set number;
- a pointer to a character array containing up to 16 characters (bytes) that specifies the data set name. The data set name must be terminated with a semicolon, a blank or '\0' character if it is less than 16 characters.

mode See access modes below.

status A pointer to an array of a least 10 elements used to indicate the success or failure of the function call (see below).

list Usually specifies the data items for which values are to be retrieved and stored in the buffer parameter. With idb_get, only @ is supported, i.e. a value for every data item in the entry will be returned (in the order defined in the data set). *List* must be a pointer to a character array.

buffer A pointer to a character array used to return the values of the items specified in the list parameter. The values are returned in the order specified in the list.

arg Ignored, except in mode 4 and mode 7. If the mode is 4, *arg* contains a pointer to the record number of the record to be retrieved.

If the mode is 7, *arg* is a pointer to the value of the key item

selected for calculated access.

IDBGET Modes

Mode 1: Reread

idb_get retrieves the current record.

Mode 2: Serial read, forward

idb_get serially retrieves the record after the current record. The retrieved record becomes the current record.

Mode 3: Serial read, backward

idb_get serially retrieves the record before the current record. The retrieved record becomes the current record.

Mode 4: Directed read

idb_get retrieves the entry with the record number specified in the *arg* parameter.

Mode 5: Chained read, forward

idb_get serially retrieves the next entry in the current chain. Successive calls to idb_get can be executed to retrieve all the entries in the chain. A previous call to idb_find can be used to establish the current chain.

Indexed read, forward

idb_get retrieves the first or next entry in the index order. idb_find on index item is used to define current index. idb_get will fail with end-of-chain condition if no more entries with search value as specified by idb_find can be found.

Mode 6: Chained read, backward

idb_get serially retrieves the previous entry in the current chain.

Indexed read, backward

idb_get retrieves the last/previous entry in index order. idb_find on index item is used to define current index. idb_get will fail with end-of-chain condition if no more entries with search value as specified by idb_find can be found.

Mode 7: Calculated read

idb_get retrieves the entry with the key item value specified in the *arg* parameter. This mode is only allowed on manual or automatic data sets.

Mode 15: Indexed read forward, ignore “end-of-chain” condition

idb_get reads the first or next entry in current index order. idb_find on index item is used to establish current index. An “end-of-chain” condition will be ignored. idb_get will just retrieve the next entry (in index order) without displaying a message until end-of-file condition.

Mode 16: Indexed read backward, ignore “end-of-chain” condition

idb_get reads the last or previous entry in current index order. idb_find on index item is used to establish current index. An “end-of-chain” condition will be ignored. idb_get will just retrieve previous entry (in index order) without displaying a message until end-of-file condition.

Description

Idb_get retrieves an entry using the access methods specified by the mode parameter. Depending on the *list* parameter, all or part of the entry may be returned. The values of the data items are placed in the buffer in the same order as they appear in the list parameter. The data returned in the buffer is byte-aligned.

For idb_get to execute successfully, the security class must be in the read class list of the data set specified by *dset* and in the read class list of the requested items.

The record retrieved by idb_get is the current record for this data set. A data set does not have a current record in the following cases:

- The data set has not yet been accessed by idb_get;
- The data set was rewound using idb_close mode 3.

To obtain a record number for a directed read (mode 4), call idb_get in another mode and save the record number returned in the status array. The record number can be used for a subsequent directed read. Because each idb function reinitializes the status array, you must copy the record number from the status array into a variable, and then pass the record number back to the idb_get mode 4 later.

NOTE: Entries are not added sequentially, so the record number incremented by 1 is not necessarily the record number of the next entry.

Return value

Returns 0 if successful, or error code if an error was encountered.

Status codes

If idb-get was successfully executed, the status array will contain the following values:

Table 16 **Status codes**

Element	Meaning
0	S_OK
1	record length
2	0
3	record number
4	0
5	0 if detail set, 1 if master set
6	0
7	backward address (0 if retrieving in index order)
8	0
9	forward address (0 if retrieving in index order)

Example

```
if(idb_get(dbid,"ORDER",5,status, "@", NULL))
   error_handler();
```

This will retrieve the next entry from current chain in the ORDER data set.

The INFO Function

The INFO funtion provides structural information about the database currently being accessed. The syntax is as follows:

```
idb_info(base,qual,mode,status,buffer)

int base;
void *qual;
int mode;
int status[10];
void *buffer;
```

The parameters are:

base	Identifies the database. This must be the return value from the idb_open call.
dset	Identifies a data set or data item, depending on the mode used. Refer to description below for information on qualifier and mode. Refer to the idb_find parameters *dset* and *item* for information on specifying the data set or data item.
mode	The available modes and information returned by each are explained below.
status	A pointer to an array of a least 10 elements used to indicate the success or failure of the function call.
buffer	A pointer to a character array containing the values of all data items in the data set.

Description

The idb_info function allows you to programmatically retrieve database structure and access information about data items and data sets. Access to structural information is restricted by the security class number specified when the database is opened. Any data sets which are inaccessible to the specified security class are considered nonexistent. For example, if your password is paired with security class number 20, idb_info will not show any data sets to which security class 20 has no access.

Idb_info can be used to make application programs independent of the database structure. If this procedure is used to retrieve all structural information, including item and set numbers, new data items and data sets can be added to the database without affecting existing programs.

IDBINFO Modes

Mode 101: Item number

Mode 101 identifies the data item number.

Qualifier Identifies the data item for which the information is requested.

Buffer Contains the following:

```
struct m101 {
   int item_number;
}
```

Mode 102: Item name, type and length

Mode 102 describes a specific data item, including its name, data type and length.

Qualifier Identifies the data item for which the information is requested.

Buffer Contains the following:

```
struct m102 {
   char item_name[16];
   char item_type;
   char blanks[3];
   int  subitem_length;
   int  subitem_count;
}
```

Mode 103: Items in database

Mode 103 identifies all data items defined in the database.

Qualifier Ignored

Buffer Contains the following:

```
struct m103 {
   int item_count;
   int item_numbers[];
}
```

The data items are listed in the order in which they are defined in the item part of the schema.

Mode 104: Fields in data set entry

Mode 104 identifies all data items in an entry of a specific data set.

Qualifier Identifies the data set for which the information is requested.

Buffer Contains the following:

```
struct m104 {
   int field_count;
```

```
   int item_numbers[];
}
```

A field is the occurrence of an item within an entry. The fields are listed in the order in which they are defined for the set entry in the schema.

Mode 201: Set number and access

Mode 201 identifies the data set number and the type of access allowed.

Qualifier Identifies the data set for which the information is requested.

Buffer Contains the following:

```
typedef struct m201 {
   int set_number;
}
```

The data set number is positive if the security class has only read access to the data set. The number is negative if the security class has both read and write access.

Mode 202: Set name, type and length

Mode 202 describes a specific data set, including its name, data set type and capacity.

Qualifier Identifies the data item for which the information is requested.

Buffer Contains the following:

```
typedef struct m202 {
   char set_name[16];
   char set_type;
   char blanks[3];
   int  entry_length;
   char blanks2[4];
   int  entry_count;
   int  capacity;
}
```

Entry count is the number of records used in the data set. *Capacity* is the maximum capacity as defined in the schema text file and the maximum entry count.

Mode -202: Set name, type and length (local)

Mode -202 describes a specific data set, including its name, data set type and capacity. This is the same as mode 202. The difference is that this is performed locally (without contacting the database server). The entry_cnt element of the return structure is always zero.

Mode 203: Sets in database

Mode 203 identifies all data sets defined in the database and the type of access allowed.

Qualifier Ignored

Buffer Contains the following

```
struct m203 {
   int set_count;
   int set_numbers[];
}
```

The data set number is positive if the security class has only read access to the data item. The number is negative if the security class has both read and write access.

Mode 204: Sets with item

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

Qualifier Identifies the data item for which the information is requested.

Buffer Contains the following:

```
typedef struct m204 {
   int set_count;
   int set_numbers[];
}
```

The data set number is positive if the security class has only read access to the data item. The number is negative if the security class has both read and write access.

Mode 301: Paths and Indexes

Mode 301 identifies the search item and sort items defined for or related to the specified data set.

Qualifier Identifies the data set for which the information is requested.

Buffer Contains the following:

```
struct m301 {
   int search_item_count;
   struct {
      int set_number;
      int search_item_number;
      int reserved;
   } path_list[];
}
```

If the qualifier specifies a master data set, the returned set numbers identify the linked detail sets. The corresponding search item number identifies the search item in the linked detail set. If the master has no detail sets, the search item count is zero.

If the qualifier specifies a detail data set, the returned set numbers identify the linked master sets. The corresponding search item number identifies the item in the detail data set. If the detail set has no master sets, the search item count is zero.

Mode 302: Primary search item

Mode 302 identifies the primary search item for a specified data set.

Qualifier Identifies the detail or master data set for which the information is requested.

Buffer Contains the following:

```
struct m302 {
   int item_number;
   int parent_set_number;
}
```

If the qualifier is a detail data set, the item number is the primary search item number, and the set number is the related master data set number.

If the qualifier is a master data set, the item number is the key item number, and the set number is zero. If you do not have access to the key item, the item number is zero.

Mode 501: Index item number

Mode 501 identifies the index item number.

Qualifier Identifies the index item for which the information is requested.

Buffer Contains the following:

```
struct m501 {
   int index_item_number;
}
```

Mode 502: Index item name and segments

Mode 502 describes a specific index item, including its name and segments.

Qualifier Identifies the index item for which the information is requested.

Buffer Contains the following:

```
struct m502 {
   char iitem_name[16];
   int  iitem_seg_cnt;
   struct {
      int item_number;
      int item_length;
   } seg[8];
}
```

Mode 503: Index items in database

Mode 503 identifies all index items defined in the database.

Qualifier Ignored

Buffer Contains the following:

```
struct m503 {
   int iitem_count;
   int iitem_numbers[];
}
```

The index items are listed in the order in which they are defined in the index item part of the schema.

Mode 504: Index items in data set entry

Mode 504 identifies all index items in an entry of a specific data set.

Qualifier Identifies the data set for which the information is requested.

Buffer Contains the following:

```
struct m504 {
   int iitem_count;
   int iitem_numbers[];
}
```

The index items are listed in the order in which they are defined for the set index in the schema.

Return value

Returns 0 if successful, or error number if an error was encountered.

Status codes

If idb-info was successfully executed, the status array will contain the following values:

Table 17 **Status codes**

Element	Meaning
0	S_OK
1	number of bytes transferred into buffer
2	0
3	unchanged
4	0
5	DB_INFO | (open_mode <<12)
6	0
7	0
8	mode
9	0

Example

```
union info info;

if(idb_info(dbid,"ORDER",202,status,&info))
   error_handler();

printf("Set name: %16.16s\\n", info.info_202.name);
printf("Set type: %c\\n",      info.info_202.type);
...
```

This will describe the ORDER data set.

The LOCK Function

The LOCK function locks a database, a data set or an item. The syntax is as follows:

```
idb_lock(base,qual,mode,status)

int base;
void *qual;
int mode;
int status[10];
```

The parameters are:

base Identifies the database. This must be the return value from the idb_open call

qual Identifies a data set, data item or lock qualifier, depending on the mode used. Refer to the description below for information on qualifier and Mode. Refer to the idb_find parameters *dset* and *item* for information on specifying the data set or data item.

mode The modes available listed below.

status A pointer to an array of at least 10 elements used to indicate the success or failure of the function call (see below).

Description

The idb_lock function locks the whole database, a data set or data items depending on mode.

Table 18 **idb_lock modes**

Mode	Target	Wait	Description
1	database	Write Wait	Lock database for writing
2	database	Write	
3	data set	Write Wait	Lock data set for writing
4	data set	Write	
5	predicate	Write Wait	Lock using for writing
6	predicate	Write	using predicate spec

Table 18 **idb_lock modes**

Mode	Target	Wait	Description
11	database	Read Wait	Lock database for reading
12	database	Read	
13	data set	Read Wait	Lock data set for reading
14	data set	Read	
15	predicate	Read Wait	Lock for reading
16	predicate	Read	using predicate spec

A *read* lock allows concurrent read locks. A *write* lock does not allow any concurrent lock (it is a lock for WRITE).

A *wait* lock will block execution until lock is available. A lock without *wait* will return status error if lock is not available.

A lock with wait will fail and return status error if another lock has already been granted and the requested lock is not available.

The Lock Descriptor Format (next page) will explain how to set up predicate specification. Using predicate specification it is possible to lock database, data set(s) or (a group of) data item(s) to be locked.

Return value

Returns 0 if successful, otherwise gives error code.

Status codes

If idb-lock was successfully executed, the status array will contain the following values:

Table 19 **Status codes**

Element	Meaning
0	S_OK
1	1
2	0
3	1

Table 19 **Status codes**

Element	Meaning
4	0
5	DB_LOCK | (open_mode << 12)
6	0
7	0
8	mode
9	0

Example

```
if(idb_lock(dbid,"ORDER",4,status))
   error_handler();
```

This will lock ORDER data set.

LOCK DESCRIPTOR FORMAT

```
   A packed block of lock descriptors.
   See programming manuals for details.

   Set Item Op Parameters
   --- ---- -- ----------------------------------------------
   @   @       set_cnt = 0
   x   @       set_cnt = 1, setno, item = 0
   @   x    x  set_cnt, set_list, item, op, vtype, value
   x   x    x  set_cnt = 1, setno, item, op, vtype, value


lock descriptor buffer :

   +-----------------+  Total size of buffer EXCLUDING this
   | total_size      |  in byte units (int)
   +-----------------+
   | descriptor #1   |
   +-----------------+
   | ...             |
   +-----------------+

lock descriptor :

   +-----------------+  Size of descriptor INCLUDING this
   | length          |  in byte units (int)
   +-----------------+
   | set             |  Set name (16 bytes) or number (int)
   +-----------------+
   | item            |  Item name (16 bytes) or number (int)
   +-----------------+
   | op              |  OP (2 bytes)
   +-----------------+
   | value           |  VALUE as defined in ROOT file
   +-----------------+  padded to next 4 byte boundary
```

The UNLOCK Function

The UNLOCK function unlocks a database. The syntax is as follows:

```
int cc = idb_unlock(base,qual,mode,status)

int base;
void *qual;
int mode;
int status[10];
```

The parameters are:

base	Identifies the database. This must be the return value from the idb_open call
qual	This parameter is ignored.
mode	The modes available listed below.
status	A pointer to an array of a least 10 elements used to indicate the success or failure of the function call (see below).

Description:

Idb_unlock is used to release data base locks. If a program has issued multiple calls to idb_open for the same database, only the access path of the given base parameter is affected.

Table 20 **idb_unlock modes**

Mode	Target	Description
1	database	Unlock database. All locks for the database are released.
3	data set	Unlock data set. A lock mode 3/4/13/14 for the specified data set is released.
5	predicate	Unlock predicate. A lock mode 5/6/15/16 is released.. The qualifier must match the idb_lock qualifier argument.

In addition to the "official" modes above, idb_unlock also accepts and translates the following mode values:

Mode 2/11/12 is mapped to 1
Mode 4/13/14 is mapped to 3

Mode 6/15/16 is mapped to 5

This makes it possible to use the same idb_lock and idb_unlock modes.

Return value

Returns 0 if successful, otherwise error code.

Status codes

If idb_unlock was successfully executed, the status array will contain the following values:

Table 21 **Status codes**

Element	Meaning
0	S_OK
1	unchanged
2	unchanged
3	unchanged
4	0
5	DB_UNLOCK | (open_mode << 12)
6	0
7	0
8	mode
9	0

Example

```
if(idb_unlock(dbid,"",1,status))
   error_handler();
```

This will release all locks for given database.

The PUT Function

The PUT function adds an entry to the database. The syntax is as follows:

```
idb_put(base,dset,mode,status,list,buf)

int base;
void *dset;
int mode;
int status[10]
void *list;
void *buffer;
```

The parameters are:

base Identifies the database. This must be the return value from the idb_open call

dset Identifies the data set in which the entries are to be located. Dset is one of the following:

- a pointer to an integer variable that specifies the data set number
- a pointer to a character array containing up to 16 characters (bytes) that specifies the data set name. The data set name must be terminated with a semicolon, a blank or \0 character if it is less than 16 characters.

mode Mode must be 1

status A pointer to an array of a least 10 elements used to indicate the success or failure of the function call (see below).

list Usually specifies the data items for which values are to be retrieved from the buffer parameter and stored in the dataset. @ is the only character supported as *list* parameter, i.e. a value for every data item in the entry will be stored (in the order defined in the data set). *list* must be a pointer to a character array.

buffer A pointer to a character array containing the values of all data items in the data set.

Description

Idb_put adds an entry to the specified data set. The database must be open in either mode 1 or mode 3. The security class number must have write access to the data set specified in dset.

If the entry being written is part of a chain, then all links are automatically maintained. If the entry is the first entry in a chain and the chain is linked to an automatic master, the entry in the automatic master is added if it is not already linked to any other chains.

If index items are defined for the specified data sets, the index entries are automatically maintained.

An entry in a manual master may not have a duplicate search item value.

Return value

Returns 0 if successful, otherwise error code.

Status codes

If idb-put was successfully executed, the status array will contain the following values:

Table 22 **Status codes**

Element	Meaning
0	S_OK
1	record length
2	0
3	record number
4	0
5	`0 if detail set, 1 if master set`
6	0
7	`backward address`
8	0
9	`forward address`

Example

```
if(idb_put(dbid,"CUSTOMER",1,status,"@",buf))
   error_handler();
```

This will add a record to CUSTOMER data set.

The UPDATE Function

The UPDATE function updates an entry in the database. The syntax is as follows:

```
idb_update(base,dset,mode,status,list,buf)

int base;
void *dset;
int mode;
int status[10]
void *list;
void *buffer;
```

The parameters are:

base Identifies the database. This must be the return value from the idb_open call

dset Identifies the data set in which the entries are to be located. Dset is one of the following:

- a pointer to an integer variable that specifies the data set number
- a pointer to an character array containing up to 16 characters (bytes) that specifies the data set name. The data set name must be terminated with a semicolon, a blank or \0 character if it is less than 16 characters.

mode Mode must be 1

status A pointer to an array of a least 10 elements used to indicate the success or failure of the function call (see below).

list Usually specifies the data items for which values are to be retrieved from the buffer parameter and stored in the dataset. @ is the only character supported as *list* parameter, i.e. a value for every data item in the entry will be returned (in the order defined in the data set). *list* must be a pointer to a character array.

buffer A pointer to a character array containing new values for all data items in the data set.

Description

Idb_update updates current entry of the specified data set. The database must be open in either mode 1 or mode 3. The security class number must have write access to the data set specified in dset.

Search item values may not be changed.

If index items are defined for the specified data sets, the index entries are automatically maintained.

Return value

Returns 0 if successful, otherwise error code.

Status codes

If idb-update was successfully executed, the status array will contain the following values:

Table 23 **Status codes**

Element	Meaning
0	S_OK
1	record length
2	0
3	record number
4	0
5	`0 if detail set, 1 if master set`
6	0
7	`backward address`
8	0
9	`forward address`

Example

```
if(idb_update(dbid,"CUSTOMER",1,status,"@",buf))
   error_handler();
```

This will update current record in the CUSTOMER data set.

The BEGIN Function

The BEGIN function begins a new (sub-) transaction . The syntax is as follows:

```
idb_begin(comment,mode,status)

char *comment;
int mode;
int status[10]
```

The parameters are:

comment	A pointer to a character array providing a comment for the transaction. It must be terminated with a 0 character. This comment is stored by the database server in the transaction log. When a NULL pointer is passed or the comment is empty the value is ignored.
mode	Must be 1.
status	A pointer to an array of a least 10 elements used to indicate the success or failure of the function call (see below).

Description

The idb_begin function begins a new (sub-) transaction. When this is the first transaction, it is called top level transaction. No modifications are permanently saved in the Eloquence A.06.00 data base until the top level transaction is committed. A subsequent idb_begin function call begins a new subtransaction, which can be controlled separately with idb_commit and idb_rollback functions.

Each transaction gets a transaction identifier assigned, which is unique during the database session. This is returned in the status. If a comment is passed to idb_begin, the string is written to the database transaction log..

Return value

Returns 0 if successful, or error number if an error was encountered.

Status codes

If idb_begin was successfully executed, the status array will contain the following values:

Table 24 **Status codes**

Element	Meaning
0	S_OK
1	Transaction ID
2	Transaction nesting level
3	0
4	0
5	0
6	0
7	0
8	0
9	0

Example

```
if(idb_begin(NULL,1,status))
   error_handler();
```

This will begin a new transaction.

The COMMIT Function

The COMMIT function commits (sub-) transaction . The syntax is as follows:

```
idb_commit(mode,status)

int mode;
int status[10]
```

The parameters are:

mode	Must be 1 or 2, see below.
status	A pointer to an array of a least 10 elements used to indicate the success or failure of the function call (see below).

IDBCOMMIT Modes

Mode 1: Commit current (sub-)transaction

idb_commit commits the current transaction. If this is the toplevel transaction, modifications are made permanent to the database. A commit on a subtransaction does only cause all modifications to become part of the parent transaction.

Mode 2: Commit top level transaction

idb_commit can be directed to commit the toplevel transaction and any currently active subtransaction.

Description

Database modifications made in a transaction are not saved in the database until the enclosing is commited. When the commit function succeeds it is guaranteed that all modifications succeeded. In case the commit fails the database it not modified at all.

As you probably have noticed, idb_commit does not take a database argument. Transactions are global for all databases even if they are executed on different servers.

Return value

Returns 0 if successful, or error number if an error was encountered.

Status codes

If idb_commit was successfully executed, the status array will contain the following values:

Table 25 **Status codes**

Element	Meaning
0	S_OK
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0

Example

```
if(idb_begin(NULL, 1,status))
   error_handler();
...
if(idb_commit(1,status))
   error_handler();
```

This will commit any modifications made in the transaction and finish the transaction.

The ROLLBACK Function

The ROLLBACK function undoes any pending (sub-) transaction . The syntax is as follows:

```
idb_rollback(id,mode,status)

int id;
int mode;
int status[10]
```

The parameters are:

id	Transaction id, used with mode 2.
mode	See below.
status	A pointer to an array of a least 10 elements used to indicate the success or failure of the function call (see below).

IDBROLLBACK Modes

Mode 1: Rollback current (sub-)transaction

The idb_rollback function is used to undo a pending transaction. If this is a top level transaction, all pending database modifications are reverted. If applied to a subtransaction all modifications including the enclosing idb_begin are reverted.

Mode 2: Rollback specified transaction

idb_rollback can be directed to rollback all transaction until the specified one. The transaction id is returned by the idb_begin function.

Mode 3: Rolback top level transaction

idb_rollback can be directed to rollback the toplevel transaction and any currently active subtransaction.

Description

Database modifications made in a transaction are not saved in the database until the enclosing is commited. The idb_rollback reverts any pending modification.

As you probably have noticed, idb_rollback does not take a database argument. Transactions are global for all databases even if they are executed on different servers.

Return value

Returns 0 if successful, or error number if an error was encountered.

Status codes

If idb_rollback was successfully executed, the status array will contain the following values:

Table 26 **Status codes**

Element	Meaning
0	S_OK
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0

Example

```
/* begin transaction */
if(idb_begin(NULL, 1,status))
   error_handler();

if ( modify_database() ) {
   /* something went wrong, revert changes */
   if(idb_rollback(0, 1,status))
      error_handler();
   return -1;
}

/* commit changes */
if(idb_commit(1,status))
   error_handler();
```

This will rollback any modifications in case the function modify_database indicates a failure. Otherwise the transaction is commited.

ERROR HANDLING

A nonsuccessful attempt to execute an idb function call will set the status array as follows:

Table 27 **Status codes**

Element	Meaning
0	error code
1	unchanged
2	unchanged
3	unchanged
4	0
5	`dbml id | (open_mode << 12)`
6	0
7	`0`
8	mode
9	`secondary status`

If error code is S_DAEMON or S_ISAM then secondary code will specify the reason of failure.

SAMPLE PROGRAM

```
/*
   sample.c

   compile: cc sample.c -o sample -l eloq
   usage  : sample database password
*/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include <eloqdb.h>

#define ABS(a) ((a)<0?-(a):(a))

int dbid;
int status[10];

main(argc, argv)
int argc;
char *argv[];
{
   int i;
   struct {
      int count;
      int setno[MAX_SET_CNT];
   } set_list;

   if(argc < 3) {
      fprintf(stderr, “usage: %s data_base password\\n”, argv[0]);
      exit(2);
   }

   dbid = idb_open(argv[1], argv[2], 9, status);
   errorhandler(”opening database”);

   idb_info(dbid, 0, 203, status, &set_list);
   errorhandler(”info 203”);
   for (i = 0 ; i < set_list.count; i++)
      set_info(set_list[i]);

   wrapup(0);
}

set_info(setno)
int setno;
{
   union info info;

   setno = ABS(setno);
   idb_info(dbid, &setno, 202, status, &info);
   errorhandler(”info 202”);
   printf(”%16.16s %02d %c %6d %8d %8d\\n”,
            info.set.name, setno, info.set.type, info.set.rec_len,
            info.set.capacity, info.set.entries);
```

```
}

wrapup(cond)
int cond;
{
   idb_exit();
   exit(cond);
}

errorhandler(action)
char *action;
{
   char tmp[80];

   if(status[0] != S_OK)
   {
       fprintf(stderr, “Status error #%d while %s\\n”, status[0],
action);

      if(idb_error(status, tmp, NULL) == S_OK)
         fprintf(stderr, “%s\\n”, tmp);
      wrapup(1);
   }
}
```

Sample Program

```
/*
   sample.c

   This is a sample program showing usage of eloquence database
   library.

   compile: cc sample.c -o sample -l eloq
   usage  : sample

   This sample will assume the existence of a database SAD
   with the following structure:

   ITEMS:
       ...
       PRODUCT-NO,          I;
       PROD-DESC,           X30;
       ...

    SETS:
       ...
    N: PRODUCT, M (/0);
    E: PRODUCT-NO,
       PROD-DESC;
       ...

*/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <signal.h>

#include <eloqdb.h>

int dbid;                   /* data base id */
int status[10];             /* data base status */

char buf[100];              /* data abse buffer */

struct Product {
   short no;
   char desc[30];
} product;

typedef struct Packlist {
   char *item;              /* item name */
   void *ptr;               /* pointer to item value */
   char type;               /* item type */
   int  size;               /* item size */
   int  count;              /* subitem count */
};

struct Packlist product_packfmt[] =
{
   { "PRODUCT-NO", &product.no  },
```

```
   { "PROD-DESC",  product.desc },
   { "" }
};

signal_handler()
{
   printf("Interrupt\n");
   wrapup(3);
}

main(argc, argv)
int argc;
char *argv[];
{
   char op;

   signal(SIGINT,  signal_handler);
   signal(SIGTERM, signal_handler);

   printf("Opening database ...\n");
   dbid = idb_open("../sad/SADN", "MANAGER", 3, status);
   errorhandler("opening database");

   setup_pack_list(product_packfmt);

   do {
      printf("Add, Update, Delete, Quit ?");
      fflush(stdout);
      scanf("%1s", buf);
      op = *buf;
      gets(buf);

      switch(op) {
      case 'A':
         add_product();
         break;
      case 'U':
         update_product();
         break;
      case 'D':
         delete_product();
         break;
      case 'Q':
         break;
      default:
         printf("Illegal - reenter\n");
      }
   } while(op != 'Q');
   wrapup(0);
}

wrapup(cond)
int cond;
{
   if(dbid >= 0) {
      printf("closing database ...\n");
      idb_close(dbid, NULL, 1, status);
      errorhandler("closing database");
   }
   idb_exit();
   exit(cond);
```

```
}

errorhandler(action)
char *action;
{
   char tmp[80];

   if(status[0] != S_OK)
   {
      fprintf(stderr, "Status error #%d while %s\n", status[0], ac-
tion);
      if(idb_error(status, tmp, NULL) == S_OK)
         fprintf(stderr, "%s\n", tmp);
      wrapup(1);
   }
}

/*
   handle product
*/

get_product()
{
   printf("Product no  ? ");
   fflush(stdout);
   scanf("%hd", &product.no);

   idb_get(dbid, "PRODUCT", 7, status, "@", buf, &product.no);
   if(status[0] == S_NOREC) {
      printf("Product no %d found\n", product.no);
      return(0);
   }
   errorhandler("get product");
   unpack_buffer(product_packfmt);
   return(1);
}

add_product()
{
   printf("Product no  ? ");
   fflush(stdout);
   scanf("%hd", &product.no);

   printf("Description ? ");
   fflush(stdout);
   scanf("%s", product.desc);

   pack_buffer(product_packfmt);
   idb_put(dbid, "PRODUCT", 1, status, "@", buf);
   if(status[0] == S_DUPL) {
      printf("Duplicate product\n");
      return(0);
   }
   errorhandler("adding product");
   printf("Product added\n");
   return(1);
}

update_product()
{
   if(!get_product())
```

```
      return(0);
   printf("Product no  : %d\n", product.no);
   printf("Description : %.30s\n", product.desc);

   printf("Description ? ");
   fflush(stdout);
   scanf("%s", product.desc);

   pack_buffer(product_packfmt);
   idb_update(dbid, "PRODUCT", 1, status, "@", buf);
   errorhandler("updating product");
   printf("Product updated\n");
   return(1);
}

delete_product()
{
   if(!get_product())
      return(0);
   printf("Product no  : %d\n", product.no);
   printf("Description : %.30s\n", product.desc);

   idb_delete(dbid, "PRODUCT", 1, status);
   errorhandler("deleting product");
   printf("Product deleted\n");
   return(1);
}

/*
   buffer pack/unack utilities
*/

setup_pack_list(listp)
struct Packlist *listp;
{
   union info info;

   while(*listp->item) {
      idb_info(dbid, listp->item, 102, status, &info);
      errorhandler("info 102");
      listp->type = info.item.type;
      listp->size = info.item.size;
      listp->count = info.item.count;
      listp++;
   }
}

pack_buffer(listp)
struct Packlist *listp;
{
   int i, ofs;

   ofs = 0;
   while(*listp->item) {
      for(i = 0; i < listp->count; i++) {
         if(listp->type == 'X')
         {
            char *to = (char *)&buf[ofs];
            char *from = (char *)listp->ptr + i*listp->size;
            int len = listp->size;
            while(*from && len) {
```

```
                *to++ = *from++;
                len--;
             }
             memset(to, ' ', len);
          }
          else
              memcpy(&buf[ofs], (char *)listp->ptr + i*listp->size,
listp->size);
          ofs += listp->size;
       }
       listp++;
   }
}

unpack_buffer(listp)
struct Packlist *listp;
{
   int i, ofs;

   ofs = 0;
   while(*listp->item) {
      for(i = 0; i < listp->count; i++) {
        memcpy((char *)listp->ptr + i*listp->size, &buf[ofs], listp-
>size);
          ofs += listp->size;
       }
       listp++;
   }
}
```

E

Obsolete Database Utilities

The database utilities documented in this chapter are no longer used as of Eloquence revision A.06.00. The documentation has been retained so this manual is still usable if you are using a previous Eloquence revision.

Introduction

The Eloquence DBMS utilities create, initialize, and purge database files and perform various maintenance operations. The utilities consist of Eloquence commands, programs, and statements, in addition to HP-UX programs and script files. The database utilities are summarized below:

Table 28 **Obsolete database utilities**

Eloquence Statements	HP-UX Programs and Script Files	Description
	dbutil	Database modification tool.
DBSTORE	dbstore	Copies either the database or selected data sets to a backup device. dbstore is a customizable script file. DBSTORE calls dbstore.
DBRESTORE	dbrestore	Restores the database from the backup created by DBSTORE or dbstore. dbrestore is a customizable script file. DBRESTORE calls dbrestore.
	dbexport	Copies data entries from all or selected data sets to ASCII files. Database structural information is *not* saved.
	dbimport	Copies data entries from ASCII files into data sets of a database.
	dbmods	Allows changing various database structural information without unloading and reloading stored data.
DBPASS		Changes the password for a stated user-class number.
DBMAINT		Changes the maintenance password for a stated database.
READ DBPASSWORD		Reads all user passwords from a specified database.

Table 28 Obsolete database utilities

Eloquence Statements	HP-UX Programs and Script Files	Description
WRITE DBPASSWORD		Writes all user passwords from a specified database.
	dstatus	Diagnostic tool for reporting the Eloquence system status. Its functionalitycould be altered as new features are developed. The parameters are:-t = list of tasks; -d = list of opened databases; -help.

DBPATCH utility

If the database ROOT file contains different information than the data sets (e.g. number of records), the database is considered corrupted and a status word -94 is returned by DBOPEN. This may be the result of a power failure or a kill -9 of the daemon in the "right" moment.

The recommended procedure to recover from a corrupted database is to reload the database from a backup, because this will guarantee a fast and consistent recovery. However, there are situations, where the only supported way to repair your database is to export it using dbexport, erase it using dberase and re-fill it using dbimport utility programs.

The dbpatch utility can be used for two different purposes:

- If started without any options, it indicates which datasets are corrupted.

  Sample output of the dbpatch utility. The mark at the end of a line indicates a corrupted dataset:

```
                                    EXPECTED        REAL
DATA SET              CAPACITY RECLEN ENTRIES  RECLEN ENTRIES
---------------- -- - -------- ------ -------- ------ --------
INTERPRET        01 A      100     32       68     32       69 <
NAME              02 A      100     64       50     64       50
INHABER           03 M      200     76      161     76      161
DISC             04 D      500    128     2429    128     2430 <
```

- If started with the **`-w`** option it patches the ROOT file.

  THIS DOES NOT FIX ANY CORRUPTED DATA, it will simply suppress the status error.
  However, this may be necessary, if time does not permit you to export/import the database.

NOTE:

The use of the dbpatch utility with the write option is UNSUPPORTED and may result in unpredictable behaviour, due to errors in the "links" between data records or data sets (Master/Detail). You should repair a corrupted database using dbexport/dbimport as soon as possible.

Database Restructuring

Certain changes to an existing structure can be made without having to transfer data from the old database to the new one. The dbmods utility allows users to change many structural items, as described on the following pages.

More extensive changes are possible by first unloading the database and recreating the root file. The general sequence is:

1. Run the dbexport to back up all data set entries.
2. Purge the old database using DBPURGE or dbpurge.
3. Redefine the database and use an editor to modify the schema.
4. Run the schema program to create the new root file.
5. Use DBCREATE or dbcreate to create and initialize the new data sets.
6. Run the dbimport to load data entries from the backup into the new database.

The following are examples of changes that can be made to the database structure using the database utility programs.

Table 29 **Type of Access**

Database Structural Change	Program To Use
Adding, changing or deleting passwords and user-class numbers.	dbmods
Changing data set read- and write-class lists.	dbmods
Adding new data item definitions.	dbexport/dbimport *
Removing data items not used as search items.	dbexport/dbimport *
Rearranging the item order of a data set entry.	dbexport/dbimport
Changing the length of string items. **	dbexport/dbimport
Changing a data item or data set name and all references to it.	dbmods
Changing numeric type items to another numeric type. ***	dbexport/dbimport
Changing numeric type items to string items and vice versa ****	dbexport/dbimport

* If a data item is added or deleted within a data set, the ASCII export file created by dbexport must be edited to conform with the new structure, or dbexport and dbimport must be used in -r (restructure) mode.

** Certain changes cannot be made to strings used as search items. For example, decreasing the string length may cause a duplicate search item value in a master data set. Increasing the length of a search item value causes no problems.

*** Numeric value conversions are performed between integer, dinteger, short, and real values. The resulting item values are as close to the starting values as the destination item type permits.

**** Requires dbexport and dbimport to be used in restructure mode.

The DBUTIL program

Introduction

Dbutil provides all functionality of the former DBMODS utility and additionally provides the ability to change some aspects of the database structure without having to export/import the database.

Database changes can either be defined interactive or by a script file (called control script). This makes it possible for a software vendor to provide a control script to a customer to perform database changes without manual interaction. The dbutil may even be used to interactively create a control script.

The following actions can be performed:

- ❒ Set the maintenance password
- ❒ Define access classes
- ❒ Grant or revoke access to datasets
- ❒ Create a new data item
- ❒ Change data item definition
- ❒ Create a new index item
- ❒ Change index item definition
- ❒ Create a new data set
- ❒ Add new data items to a data set
- ❒ Add new indices to a data set
- ❒ Additionally, in interactive mode, it is possible to print a schema definition of the database.

DBUTIL commandline arguments

Synopsis:

```
usage: dbutil [options] [file]

options:

 -help   = show usage (this list)
 -i      = interactive mode (batch mode only)
 -n      = pretend          (batch mode only)
 -v      = verbose          (batch mode only)
 -e cnt  = abort processing after encountering cnt errors
 -t tmp  = where temporary files are stored
 -d      = debug
```

If a file argument is present, dbutil will process in batch mode unless the **-i** option is present. If the **-n** option is present, no changes will be made to the database. Processing will end after checking the input file and the analysis phase.

Arguments:

-i After processing the supplied batch file, dbutil will change into interactive mode, so you can verify the database changes made so far or you can add additional changes. Interactive mode is the default, if no control file is specified on the command line.

-n dbutil will analyze the control file but will not make any changes to the database. This option can be used to check the syntax of the control file or to get an overview which changes would be applied to the database.

-v[v] Specifying the -v option will cause dbutil to output a summary of changes after analyzing the control file and some descriptive text during the database restructuring.

Specifying two -v options will cause dbutil to echo the control file to stdout as it is analyzed.

-e cnt Abort processing the control file after encountering the given number of syntax or validation errors.

-t tmp This option makes it possible to specify where temporary files will be created. As temporary files can become quite huge, it may overflow the default location.

If the -t argument is not specified, dbutil will allocate temporary files at the same location as the database.

-d This is used internally to debug dbutil itself. You should not use it.

If a file argument is present, dbutil will process in batch mode unless the **-i** option is present.

Dbutil execution

Dbutil will execute in different phases:

1. If a control file is given on the command line, it will analyze the control file and all changes imposed by the control file.
2. If the interactive mode is requested by either specifying the -i argument or by omitting the control file, it will change into interactive mode.

3 Finally it will analyze all changes and list the restructure operations it needs to perform. If the -n argument is present execution will stop here.

4 If stdout and stdin are connected to a tty device, dbutil will prompt for a confirmation to apply the database changes. This is the last chance to change your mind. If you interrupt the program beyond this point, your database will be lost and you have to restore it from your backup.

5 Finally dbutil will start to restructure the database.

Example

```
$ dbutil test/du5

B1368B DBUTIL (C) COPYRIGHT MARXMEIER SOFTWARE AG 2002 (A.05.10)

Analyzing restructure specification ...
Analysis completed successfully

Checking database consistency ...
Consistency check completed successfully

Database restructure analysis:

Analyzing changes:
 NEW-SET-M
 * this is a new data set
   Estimated temporary disc space required: 1 MB

 NEW-SET-D
  * this is a new data set
  Estimated temporary disc space required: 1 MB

Estimated temporary disc space required: 1 MB

Temporary files will be created at
 /disc/project/eloq/db2/ndbmods/test/db/

Available disc space: 147 MB

Data restructure process required.


------------------------------------------------------------
PLEASE NOTE:
If the restructure process fails or is interrupted, YOUR
DATA BASE IS LOST and can only be recovered from a backup.
If you don't have a current backup, you should NOT continue.
------------------------------------------------------------

Continue with database restructure (y/n) ? y

CAUTION: DO NOT INTERRUPT THIS PROGRAM!

Removing current ROOT file

Restructuring database ...
NEW-SET-M
NEW-SET-D

Dumping new ROOT file /disc/project/eloq/db2/ndbmods/test/db/DB
```

```
End of Database Maintenance Utility
```

Control file syntax

The dbutil control file is a plain text file. The following general rules apply:

- Everything after a hash character (#) is considered a comment and will be ignored.
- dbutil does recognize keywords in either upper or lower case (but not mixed).
- Each statement must be delimited by a semicolon (;)
- Identifiers such as Item, Index and Set-names may be specified in either case (upper/lower case).
- Strings must be enclosed in double quotes. To include a quote character in a string, the quote character must be preceded by a backslash (\) character.
- A statement may span multiple lines.
- Some statements allow repeating a list of definitions by enclosing it in curly braces.

The syntax description below use the following conventions:

- All keywords are given in upper case.
- All Identifiers, such as Item, Index and Set names are given in upper/lower case.
- Optional syntax elements are enclosed in brackets.

Opening the database

```
OPEN DATABASE "database" [PASSWORD "password"] ;
```

database The database argument specifies the database path and name. The database path may be specified either by giving a HP-UX path or a Eloquence volume.

password The password argument specifies the database maintenance password. The PASSWORD clause is not required if the database has no maintenance password or you are the superuser (root).

For example:

```
OPEN DATABASE "/opt/sqlr/db.g/db";
```

This opens the database "db" located in directory **/opt/sqlr/db.g**.

```
OPEN DATABASE "test,DB" PASSWORD "secret";
```

This opens the database "TEST" at the path described by the Eloquence volume "DB".

Changing access passwords

The Eloquence database has 32 access profiles. The access profile is selected during database open, depending on the given password. If no password is given, or there is no password at all, access profile 0 (PUBLIC) is selected.

ALTER PASSWORD
acl_no ["password"];

```
ALTER PASSWORD {
   acl_no ["password"];
   ...
}
```

Access profiles 1 to 31 are associated with a password. Omitting the password (and specifying only the access class) will cause the password for this class to be removed and the access class become disabled. The password must not exceed 8 characters.

If enclosed in curly braces, multiple passwords can be defined.

For example:

```
ALTER PASSWORD
   1 "GuessMe";
```

This defines the password "GuessMe" for access profile 1.

```
ALTER PASSWORD {
    3;
    4 "Manager";
}
```

This statement disables the access profile 3 (by removing the password) and defines the password "Manager" for access profile 4.

Granting access

```
GRANT privilege ACCESS
  ON target_list
  TO acl_list;
```

Where privilege is one of:

READ	Allow read access
WRITE	Allow read and write access
ALL	Same as WRITE

The **target_list** is either **ALL** or a list of data set identifiers separated by a comma (,). If **ALL** is specified, the privilege will be granted to all data sets.

The **acl_list** is a list of access profile numbers separated by a comma (,). The keyword **PUBLIC** may be used as a synonym for access class 0.

For example:

```
GRANT READ ACCESS
ON Customers,Parts,Orders,Order-Entries
TO PUBLIC;
```

This will allow read access to the data sets Customers,Parts,Orders and Order-Entries to everyone.

```
GRANT WRITE ACCESS
ON Customers,Parts,Orders,Order-Entries
TO 1,2,3;
```

Write access to the given sets is enabled for members of the access class 1, 2 or 3.

Revoking access

```
REVOKE privilege ACCESS
  OF target_list
  FROM acl_list;
```

Where privilege is one of:

READ	Deny read access
WRITE	Deny read and write access
ALL	Same as WRITE

The **target_list** is either **ALL** or a list of data set identifiers separated by a comma (,). If **ALL** is specified, the privilege will be revoked from all data sets.

The **acl_list** is a list of access profile numbers separated by a comma (,). The keyword **PUBLIC** may be used as a synonym for access class 0.

For example:

```
REVOKE ALL ACCESS
OF Budget
FROM PUBLIC;
```

This revokes any access to the data set Budget for members of the access profile PUBLIC.

```
REVOKE WRITE ACCESS
OF Parts
TO 1,2;
```

This statement revokes write access on data set Parts for members of access pro files 1 and 2.

Creating a new data item

```
CREATE ITEM
   Ident, [count] type [ (format) ];

CREATE ITEM {
   Ident, [count] type [ (format) ];
   ...
}
```

Ident is the item name, **count** is the optional subitem count, **type** is the item type, **format** is an optional format number.

If the statement is enclosed in curly braces, multiple items can be defined.

Item type is the same as defined by the schema processor and must be one of:

I, I2	Allow read access
I4, D	Allow read and write access
R4, S	Short REAL
R8, L	REAL
Xn	String, size specified by n must be even

The format number may be an arbitrary number. It may be retrieved using DBINFO and is used by the QUERY program to define the item output format.

The QUERY format number is composed by adding the various values, which are given after the plus sign.

Table 30 **Query format numbers**

Bits	Field description	Field value	Description	add value
0	Item Protection	0	Item value may be changed in QUERY	0
		1	Item is write protected	1
1-2	Item type	0	default	0
		1	date type (days since 1972)	2
		2	Currency	4
		3	Undefined	6

Table 30 **Query format numbers**

Bits	Field description	Field value	Description	add value
0	Item Protection	0	Item value may be changed in QUERY	0
3	Item spacing	0	default	0
		1	Comma every 3 digits	8
4-6	Post decimals	0	default	0
		1	FIXED 0	16
		2	FIXED 1	32
		3	FIXED 2	48
		...		
		7	FIXED 6	112

For example:

```
CREATE ITEM
  Call-Id, D (2);
```

This defines the data item Call-Id of type 'D' with format number 2.

```
CREATE ITEM {
  Call-Id,   D;
  Call-Date, D (2);
  Call-Desc, 2X40;
}
```

This defines the data items **`Call-Id`**, **`Call-Date`** and **`Call-Desc`**.

Changing data item definition

```
ALTER ITEM Ident {
  [NAME  = Ident;]
  [TYPE  = [count] type [(fmt)]; ]
  [TYPE  = (fmt); ]
}
```

Ident is the item name, **count** is the optional subitem count, **type** is the item type, **format** is an optional format number.

The subitem count cannot be changed, if the data item is used as a search item or is included in an index item. The length of a string item cannot be reduced if it is used as a search item.

Changing the item type does will convert all values to the new item type.

Multiple item attributes must be enclosed in curly braces.

For example:

```
ALTER ITEM Phone-No
   TYPE  = X12;
```

This does change the type of the item **Phone-No** to an eight character string.

```
ALTER ITEM Phone-No {
  NAME = Phone;
  TYPE = 2 X20;
}
```

This does change the item name from **Phone-No** to **Phone**, the subitem count to two and the item type to a twenty character string.

Creating a new index item

```
CREATE IITEM
  Iident = Ident[:length ] ...;

CREATE IITEM {
  Iident = Ident[:length ] ...;
  ...
}
```

Iident is a new index item name, **Ident** is an existing item name and the optional **length** clause specifies a different index segment length. The index item name may not be defined as an item. Up to 7 index segments can be specified, separated by a comma (,).

If enclosed in curly braces, multiple index items can be defined.

For example:

```
CREATE IITEM
  ICall-Id = Call-Id;
```

This defines a new index item named ICall-Id using the value of the data item Call-Id.

```
CREATE IITEM
   ICall-Id = Call-Date, Call-Id;
```

This defines a new index item named **ICall-Id** combining the items **Call-Date** and **Call-Id**.

Changing data item definition

```
ALTER IITEM Ident {
   [NAME = Iident;]
   [TYPE = Ident[:length] ...;]
}
```

Iident is a new index item name, **Ident** is an existing item name and the optional **length** clause specifies an different index segment length. The index item name may not be defined as an item.

Up to 7 index segments can be specified, separated by a comma (,).

For example:

```
ALTER IITEM ICall-Id {
  NAME = IX-Call-Id;
  TYPE = Call-Id;
}
```

This changes the index item name from **ICall-ID** to **IX-Call-Id** and defines that the index is built using the value of the data item **Call-Id**.

Creating a new data set

```
CREATE SET SetIdent type {
   Ident [(MasterSetIdent)];
   ...
}
```

SetIdent is the name of a new data set, **type** is the data set type.

The optional **MasterSetIdent** is the name of a master data set.

Type must be one of:

M or MASTER	Master data set
A or AUTOMATIC	Automatic data set
D or DETAIL	Detail data set

For example:

```
CREATE SET Calls, D {
  Call-Id(Id);
  Call-Date;
  Call-Desc;
}
```

This creates the new detail data set **Calls** containing the data items **Call-Id**, **Call-Date** and **Call-Desc**.

The data item **Call-Id** is used as a search item with the master set **Id**.

Changing a data set:

The ALTER SET statement can be used to

- change data set properties
- add new data items to an existing data set
- add a new index to an existing data set

```
ALTER SET SetIdent
{
   [NAME = SetIdent;              ]
   [CAPACITY = number;            ]

   [ADD ITEM {
     [Ident [(MasterSetIdent)];   ]
   }                              ]

   [ADD INDEX
    Iident [/"collate"];          ]
}
```

SetIdent is the name of an existing data set, capacity is the capacity value.

Ident is an existing item name, the optional MasterSetIdent is the name of a master data set. A path can only be specified on new data sets.

Iident is an existing index item name, collate is the name of a collating sequence. The collating sequence name consists of a locale name and optional the modifier fold or nofold separated by an at (@) character. Please refer to the section "Collating sequences" for a description on collating sequences.

Changing Set properties

For example:

```
ALTER SET Calls {
  NAME = TheCalls;
  CAPACITY = 0;
}
```

This changes the data set name from Calls to TheCalls and the data set Capacity to zero.

Adding a data item

For example:

```
ALTER SET Calls {
  ADD ITEM {
     Call-Id(Id);
     Date;
     Code;
     Call-Desc;
  }
}
```

This adds four data items to the data set Calls. The Item Call-Id is used as a search item with an associated Master data set Id.

Adding an index

For example:

```
ALTER SET Calls {
  ADD INDEX ICall-Id;
  ADD INDEX ICode / "german@fold";
}
```

This will add two indices to data set Calls. The index ICode does use the collating sequence "german@fold".

Example script

```
### open database

DATABASE "db,DB" PASSWORD "Secret";

### passwords

ALTER PASSWORD {
 1 "READ";
 2 "WRITE";
}

### permissions

REVOKE ALL
OF ALL
FROM ALL;

GRANT READ
ON ALL
TO 1;

GRANT ALL
ON ALL
TO 2;

### items

CREATE ITEM {
 Call-Id,     D;
 Date,        D (2);
 Code,        X20;
 Description, 4X40;
}

### iitems

CREATE IITEM
 ICall-Id = Call-Id;

CREATE IITEM
 ICode = Code:10;

### set

CREATE SET Calls, D {
  Call-Id;
  Date;
  Code;
  Call-Desc;
}
ALTER SET Calls
{
   ADD INDEX ICall-Id;
   ADD INDEX ICode / "german@fold";
}
```

Interactive usage

The following general rules apply:

- If a Push button starts with a number, you can press the equivalent function key as an accelerator. For example **`[8. Exit]`** can be triggered with the function key F8.
- An underlined character indicates that the associated function can be reached by pressing function key F2 and the underlined character.
- An entry in a selection list can be selected by pressing the RETURN key or by pressing the space bar.
- There is currently no functionality associated with the help push button.

The Entry dialog

This dialog appears if dbutil is executed in interactive mode. The following dialog elements are present:

Database	The database name and path
Maintenance Password	The database maintenance password
Log	Shows the log window
Accept	Open the database
Exit	Exit the program

The Main dialog

The following dialog elements are present:

Log	Shows the log window
Exit	Exit the program. If there are unsaved changes, the exit dialog will appear.

The following functions can be selected from the selection list:

- Change maintenance password
- Change password or access
- Data Items
- Index Items
- Data sets
- Add Data Item to Set
- Add Index Item to Set
- Print Schema definition
- Analyze changes
- Apply changes

The Log dialog

The log windows holds a log of the most recent performed operation. If an error is encountered, the log dialog pops up automatically. The following dialog elements are present:

Close Close the log window

The Exit dialog

The exit dialog pops up, if you are about to leave the program while while your changes have not been saved. The following dialog elements are present:

Save changes If you select this checkbox and enter a file name in the following field, your changes are saved in a session script.

Accept Exit the program. If selected, your changes are saved to a session script file.

Cancel Don't exit the program, return to main dialog.

The Maintenance Password dialog

The following dialog elements are present:

Accept Save changes.

Cancel Discard changes, return to main dialog.

The Password dialog

This shows a list of all database access classes along with the password and an access summary. The first access class is PUBLIC which is used, if no password is provided to DBOPEN or no password is defined for the database. The R gives the number of data sets, this access class has READONLY access t, the W gives the number of data sets, this access class has READ/WRITE access to.

By selecting an access class, the access class properties dialog is opened. The following dialog elements are present:

Cancel Discard changes, return to main dialog.

The Access class properties dialog

In the Access Class properties dialog, you can change the password or the access rights associated with an access class. The following dialog elements are present:

Password Change the password associated with access class. If the password is empty, the access class becomes disabled. You cannot assign a password for the PUBLIC access class.

Read/Write Select this to select or modify the data sets, the current access class has READ/WRITE access to.

Read only Select this to select or modify the data sets, the current

	access class has READ ONLY access to.
Log	Shows the log window
Accept	Save changes, return to password dialog.
Cancel	Discard changes, return to password dialog.

To change the access rights for the current access class, select the Read/Write or Read Only push buttons. This opens the Data Set Selection dialog. It will display the list of data sets. In front of the data set name is a checkbox. If selected, the specified access is allowed for the current access class. If you select a data set, the access is changed.

The Data Item List dialog

This dialog shows all or a subset of the data items in your data base. By selecting a data item, the Data Item Properties dialog is opened. The following dialog elements are present:

Filter	The value entered in this field is used to select, which data items should be displayed. If this field is not empty, only data matching items are displayed
By Set	If selected, the value of the Filter field is considered a data set name. Only data items present in the selected data set are displayed.
Log	Shows the log window
Create	Opens the Item properties dialog to create a new data item.
Close	Close window, return to main dialog.

The Item Properties dialog

The following dialog elements are present:

Item Name	The data item name
Item Count	Number of elements
Item Type	Select an item type
Size	This is only used if the item is of type String. This is the maximum number of characters this item can hold. Size must be an even value.
Item Format	The item Format. This information can be used by QUERY or other programs to format their output. Please refer to the table "Query Format Numbers" for more information.
Log	Shows the log window
Accept	Save changes.
Cancel	Discard changes, return to previous dialog.

The Index Item List dialog

This dialog shows all or a subset of the index items in your data base. By selecting a data item, the Index Item Properties dialog is opened. The following dialog elements are present:

Filter	The value entered in this field is used to select, which index items should be displayed. If this field is not empty, only data matching items are displayed
By Set	If selected, the value of the Filter field is considered a data set name. Only data items present in the selected data set are displayed.
Log	Shows the log window
Create	Opens the Item properties dialog to create a new index item.
Close	Close window, return to main dialog.

The Index Item Properties dialog

The following dialog elements are present:

Name	The index item name
Segment / Item	Data item name for this index segment
Segment / Length	If this is a string item, only a leading part of the item may be used in the index.
Log	Shows the log window
Accept	Save changes.
Cancel	Discard changes, return to previous dialog.

The Data Set List dialog

This dialog shows all or a subset of the data sets in your data base. By selecting a data set, the Data Set Properties dialog is opened. The following dialog elements are present:

Filter	The value entered in this field is used to select, which data sets should be displayed. If this field is not empty, only matching data sets are displayed
Log	Shows the log window
Create	Opens the data set properties dialog to create a new data set.
Close	Close window, return to main dialog.

The Data Set Properties dialog

The following dialog elements are present:

Set Name	The data set name
Set Type	Select a set type

Capacity	The data set capacity.
Log	Shows the log window
Accept	Save changes.
Cancel	Discard changes, return to previous dialog.

The Data Set Entry dialog

This dialog is opened by selecting one of the following entries from the main dialog after selecting an Data Set from the Data Set Selection Dialog.

- Add Data Item to Set
- Add Index Item to Set

The dialog display the data or index items for the selected data set. The following dialog elements are present:

Log	Shows the log window
Add Item	Opens the Data Item or Index Item dialog.
Close	Close window, return to main dialog.

The Print Schema Definition dialog

The following dialog elements are present:

Write to file	Output Schema definition to file if selected.
File	File name and path.
Print to HP-UX printer	Output Schema definition to printer if selected
Printer	Printer name (and additional options).
Lines/Page	Number of lines / Page. Use zero to avoid no pagination.
Log	Shows the log window
Accept	Create Schema definition.
Cancel	Return to main dialog.

The Analyze dialog

The log windows holds a summary of all pending operations. The following dialog elements are present:

Close	Close the window

The Apply dialog

The exit dialog pops up, if you are about apply your changes to your database. The following dialog elements are present:

Save changes	If you select this checkbox and enter a file name in the following field, your changes are saved in a session script.
Analyze	This pops up the Analyze dialog. If summarizes the pend-

	ing changes.
Accept	Apply changes to your database. If selected, your changes are saved to a session script file.
Cancel	Don’t apply changes, return to main dialog.

The dbmods Program

The database modification (dbmods) utility program allows the user to make certain changes in the database structure without the need to unload and load data stored in the database. dbmods can be used to modify database passwords, user-class accesses, item names, item format numbers, set names, set capacities, and database volume labels. These changes do not affect data stored in the database. However, modifications made by dbmods may require minor changes to any application programs that access the database.

The dbmods utility program maintains two modification counts associated with the root file. One count, the password modification count, is increased when any passwords (including the maintenance word) are modified. A second count, the database modification count, is increased when any other changes are made to the root file. No changes are made to the modification count unless the root file is actually modified by the program. These counts enable the user to detect unauthorized database modifications made via the utility program.

To run the dbmods utility program, execute the following from the HP-UX prompt:

```
dbmods
```

The initial menu requests the database name, root file volume specifier, and the maintenance word (if used), as shown by the following screen:

```
A.03.00                  DATABASE MODIFICATION UTILITY
                              DATABASE ENTRY FORM


          DATABASE NAME _____________________________________________

          ROOT FILE VOLUME ________

          DATABASE MAINTENANCE WORD _______________


_____________________________________________________________________________
Please complete this form.

_____________________________________________________________________________
 ACCEPT |        |        |        |        |        |        | EXIT
 DATA   |        |        |        |        |        |        | PROGRAM
```

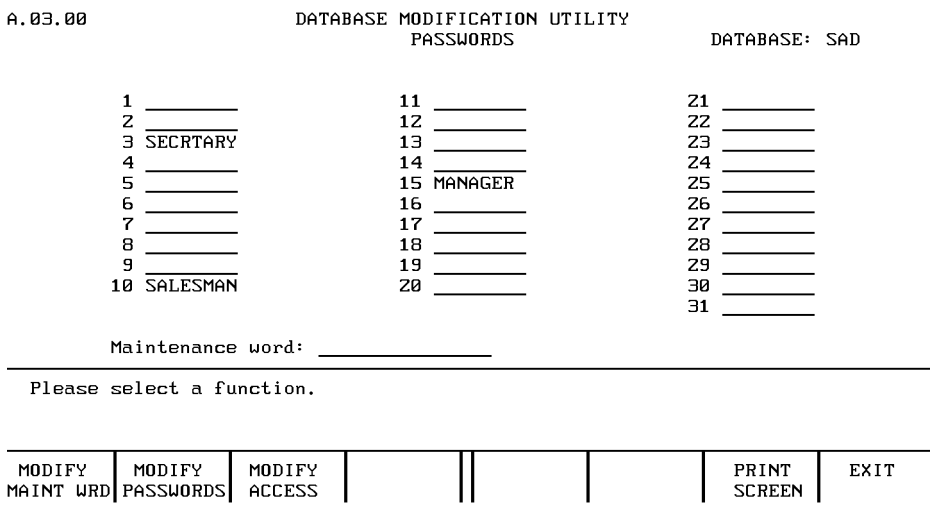
After entering all items and pressing ACCEPT DATA, the utility opens the database and displays the menu shown next. An error message indicates if the root file cannot be found or the database has already been opened by another user. If an error occurs, either re-enter another database name or press EXIT PROGRAM.

The main selection menu allows you to run any of the list and modification routines. Each is explained in the following pages. The example screens used here show information obtained from the SAD (sales analysis) database.

```
A.03.00                  DATABASE MODIFICATION UTILITY
                               MAIN SELECTION MENU            DATABASE: SAD

    PASSWORD       - List and modify database passwords.

    ITEM           - List and modify item names and format numbers.

    IITEM          - List and modify index item names.

    SET            - List and modify set names, volumes, capacities and access.

    PRINT SCHEMA   - Print schema definition of database.

    SELECT PRINTER - Select default printer for print screen functions.

    RESTART        - Run program again using another database.

Please select a function.
Schema created:  Th, 15.11.90, 20:03:02 by mike
Database modification count:  0    Password modification count:  0

PASSWORD | ITEM | IITEM | SET | PRINT SCHEMA | SELECT PRINTER | RESTART | EXIT PROGRAM
```

List and Modify Passwords

Press the PASSWORD softkey from the main selection menu to list and modify all database passwords and to modify user-class access capabilities. The Passwords menu displays all user passwords and the maintenance word. Here is a sample Passwords screen:



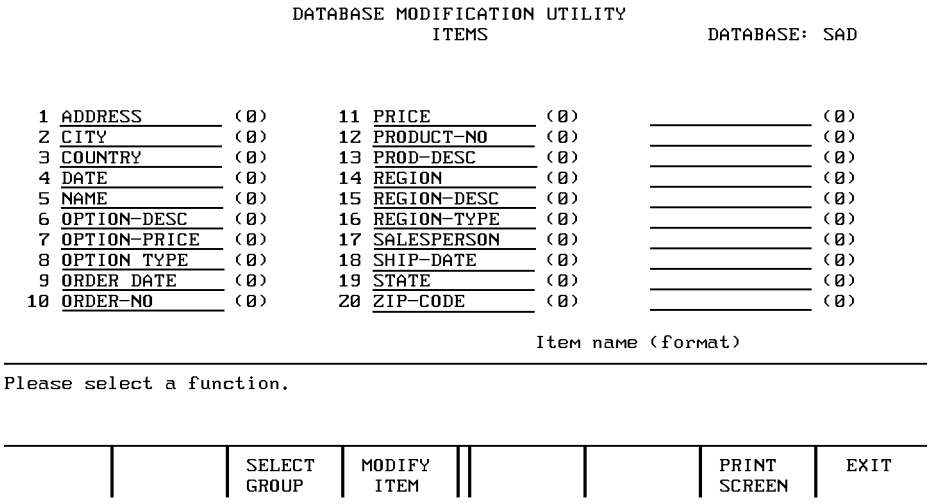
To modify the maintenance word, passwords, or user-class access, press the appropriate softkey and enter the requested items.

Press PRINT SCREEN to obtain a hard-copy output of the password list on the currently selected printer (see page 302 later in this chapter).

Press EXIT to return to the previous menu.

List and Modify Items

Press the ITEM softkey from the main selection menu to list and modify data item names and format numbers. The Items menu displays all data item names and numbers in the database. Here is a sample Items screen:



The SELECT GROUP softkey allows you to select alternate ways of displaying data items—by schema number or by data set order. When items are listed in data set order, all items for a particular data set are listed in schema definition order. If more than 30 items have been selected, additional items may be displayed using the NEXT GROUP and PREVIOUS GROUP softkeys.

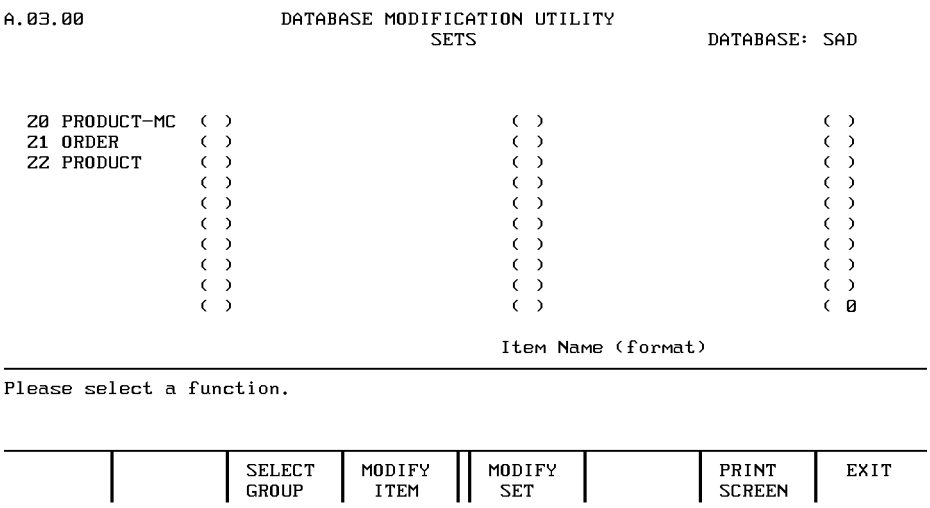
The MODIFY ITEM softkey allows you to change item names and format numbers. Any item in the database can be modified; it need not be currently displayed.

Press PRINT SCREEN to obtain a hard copy of the current item list on the currently selected printer (see page 302 later in this chapter).

Press EXIT to return to the previous menu.

List and Modify Index Items

Press the IITEM softkey from the selection menu to list and modify index item names. The Index Item menu displays all the index item names in the data base. Here is a sample Index items screen:



The SELECT GROUP softkey allows you to select alternative ways of displaying index items - by schema number or by data set order. When index items are listed in data set order, all index items for a particular data set are listed in schema definition order. If more than 30 index items have been selected, additional index items may be displayed using the NEXT GROUP and PREVIOUS GROUP softkeys.

The MODIFY IITEM softkey allows you to change index names. Any index item in the database can be modified. It need not be currently displayed.

Press PRINT SCREEN to obtain a hard copy of the current index item list on the currently selected printer (see “Select Printer” later in this chapter.)

Press EXIT to return to the previous menu.

List and Modify Data Sets

Press the SET softkey from the main selection menu to list and modify data set names, volumes, capacities, and user-class access. The Sets menu displays the first 20 data set names, along with their associated volume labels and capacities. If more than 20 sets are found, NEXT GROUP and PREVIOUS GROUP softkeys appear to allow reviewing all data sets. Here is a sample Sets screen:

```
A.03.00                  DATABASE MODIFICATION UTILITY
                                    SETS                    DATABASE: SAD

  1 DATE,SALES_____________ (     50)   _____________________________ (     )
  2 ORDER__________________ (    100)   _____________________________ (     )
  3 PRODUCT,SALES__________ (     11)   _____________________________ (     )
  4 LOCATION,SALES_________ (     17)   _____________________________ (     )
  5 OPTION_________________ (    300)   _____________________________ (     )
  6 CUSTOMER_______________ (    100)   _____________________________ (     )
    _____________________________ (     )   _____________________________ (     )
    _____________________________ (     )   _____________________________ (     )
    _____________________________ (     )   _____________________________ (     )
    _____________________________ (     )   _____________________________ (     )
    _____________________________ (     )   _____________________________ (     )

                                                Set Name, Volume (Capacity)

Please select a function.

          |          | SELECT  | MODIFY  | MODIFY  |          | PRINT   | EXIT
          |          | GROUP   | ACCESS  |  SET    |          | SCREEN  |
```

The SELECT GROUP softkey allows you to select alternate ways data sets are listed. You can list sets in schema order (ascending set number order), by data set volume order, or by user-class number. When sets are listed in data set volume order, all sets stored on a particular volume are listed in schema definition order. When sets are listed by user-class number, all sets accessible by a particular user-class number are listed in schema definition order.

The MODIFY SET softkey allows you to change data set names, set volume labels, and set capacities. When a volume label is changed on a created data set, the entries in the existing data set are copied to the new volume. Any data set in the database can be modified regardless of which sets are currently displayed. The set capacity field is for informational purposes only. It increases automatically as new entries are added to a data set. If entries are deleted, the number displayed does not automatically decrease. This provides you with information on the maximum number of entries ever in the data set. This field can be reset.

Print Schema Listing

Press the PRINT SCHEMA softkey from the main selection menu to generate a schema definition of the database. A screen similar to the following is displayed:

```
A.03.00                  DATABASE MODIFICATION UTILITY
                                SCHEMA LISTER              DATABASE: SAD


    PRINT SCHEMA    - Print the schema definition of the database
                      to the printer.

    COPY TO FILE    - Copy the schema definition of the database
                      to a new or existing DATA file.

    EXIT            - Return to main selection menu.
```

PRINT SCHEMA	PRINT TO FILE						EXIT

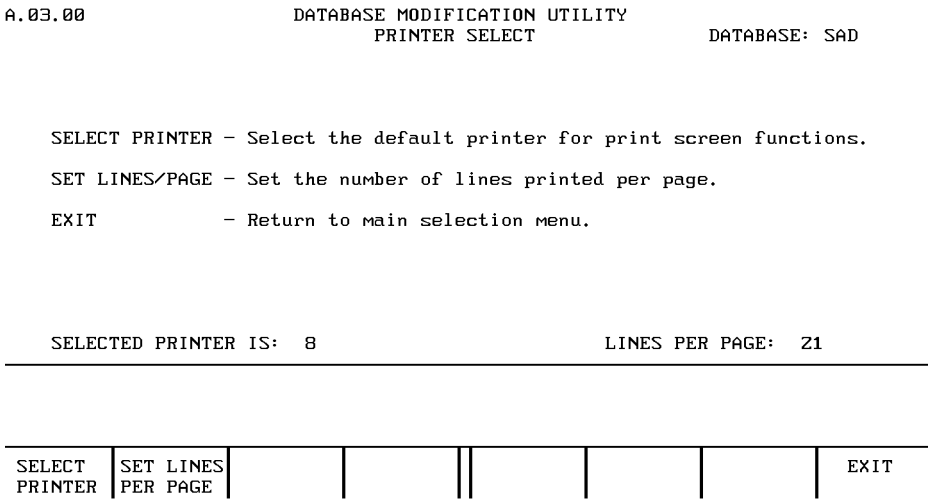
Press PRINT SCHEMA to obtain a hard copy of the schema listing on the currently set printer (see page 302).

Press PRINT TO FILE to copy the schema listing to a DATA file. If the specified DATA file does not already exist, the utility creates a new file. After the schema definition is copied to a DATA file, the file can be edited and used by the schema program.

Press EXIT to return to the main selection menu.

Select Printer

Press the SELECT PRINTER softkey from the main selection menu to specify the output device used for dbmods print operations. A screen similar to the following is displayed:



Press SELECT PRINTER to change the default device used for dbmods print operations. An error indicates if the selected device is not on line or not ready.

Press SET LINES PER PAGE to change the default page length. The range is from 20 through 200 lines per page.

Press EXIT to return to the main selection menu.

The DBPASS Statement

The DBPASS statement allows you to change the password for a stated user class number. Syntax for this statement is as follows:

DBPASS *root file spec*, *user-class number*, *old password* TO *new password*

The root file spec is a string expression containing the root file name and, optionally, its volume specifier. The user-class number is a numeric expression ranging from 1 through 31. The old password and new password parameters are string expressions from 0 through 8 characters in length and may be terminated by a space or semicolon. Longer strings are automatically truncated. Allowable characters are A through Z, 0 through 9, and the underscore character "_". The old password specified must match the corresponding password on the root file. For example, the following statement changes the password FRED to FRIEDA for user class 1:

```
DBPASS "LEDGER",1,"FRED" TO "FRIEDA"
```

The DBMAINT Statement

The DBMAINT statement allows you to change the maintenance password for a specified database. Syntax for this statement is as follows:

DBMAINT *root file spec, old word* TO *new word*

The root file spec is a string expression containing the root file specifier and, optionally, its volume specifier. The old word and new word parameters are string expressions from 0 through 16 characters. Allowable characters are A through Z, 0 through 9, and the underscore character "_". The old word specified must match the current maintenance word for the database. The maintenance word is established when the root file is created via the DBCREATE statement. For example, the following statement changes the maintenance password for the LEDGER database from SECRET to MANAGER:

```
DBMAINT "LEDGER", "SECRET" TO "MANAGER"
```

The READ DBPASSWORD Statement

The READ DBPASSWORD reads all user passwords from the specified database into a string array. Syntax for this statement is as follows:

READ DBPASSWORD *root file spec, maintenance word*; *string array variable*

The string array must have at least 31 elements, with a dimensioned length of at least eight characters long. Passwords are read into array elements in numerical order, beginning with the password for user-class number 1. If no password exists for a user-class number, that array element is filled with eight spaces. Here is a program sequence which reads and displays the user passwords from a root file named PAYROL.

```
10 OPTION BASE 1
20 DIM Pass$(31) [8]
30 READ DBPASSWORD “PAYROL”,”MANAGER”;PASS$(*)
40 DISP “User-class Numbers and Passwords on the PAYROL Database.”
50 DISP
60 FOR Num=1 TO 31
70   IF Pass$(Num)<>”        “ THEN DISP Num;Pass$(Num)
80 NEXT Num
90 END
```

The WRITE DBPASSWORD Statement

The WRITE DBPASSWORD statement replaces all passwords in the specified root file with those in a specified string array. Syntax for this statement is as follows:

WRITE DBPASSWORD root *file spec*, *maintenance word*; *string array variable*

The string must contain at least 31 elements, with a dimensioned length of a least eight characters. The string in the first element becomes the password for user-class number 1; the second string becomes the password for user-class number 2; and so on. Passwords are between 0 through 8 characters and may be terminated by a space or semicolon. Longer strings are automatically truncated. If an element in the string array is null or if the first eight characters are blanks, the corresponding user-class number is not assigned a password.

Here is a program which lists, and then allows the user to change, passwords for a specified database:

```
10    OPTION BASE 1
20    ON HALT GOTO Exit
30    DIM Pass$(31)[8],File$[13],Maint$[8]
40    INTEGER Num,Unc
50    DISP “    “,SPA(24); “    EDIT DATABASE PASSWORD  “
60 START: !
70    ON ERROR GOTO Trapperr
80    CURSOR (1,20),UL(80),(1,21)     !Draw prompt line.
90    INPUT “Enter root file name (and volume spec):”;File$
100   INPUT “    Enter maintenance password:”;Maint$[1;8]
110   CURSOR (1,21)
120   DISP “    Reading passwords.”
130   READ DBPASSWORD File$,Maint$;Pass$(*)
140 Disp_list: ! Display password list.
150   CURSOR (14,4)
160   DISP “User-class Numbers and Passwords on “;TRIM$(File$) ;”Da-
tabase.”
170   DISP
180   FOR num=1 TO 28 STEP 4
190      DISP SPA(6);Num;Pass$(Num),Num+1;Pass$(Num+1),Num+2,
Pass$(Num+2),Num+3;Pass$(Num+3)
200   NEXT Num
210   DISP SPA(6);29;Pass$(29),30;Pass$(30),31;Pass$(31)
220   Edit: !
230   CURSOR (1,21)
240   Ucn=0
250  INPUT “  Enter password number to edit :”;Ucn
260  IF NOT Ucn THEN Done            ! Exit if no number entered.
270  IF (Ucn<1) OR (Ucn>31) THEN 240  ! Loop if ucn out of range.
280  EDIT “Edit password:”,Pass$(Ucn)[1;8]
290  GOTO Disp_list                  ! Re-display passwordlist.
300  Done: ! Write passwords on root file.
310  CURSOR (1,21)
```

```
320  DISP "'‘Writing passwords."
330  WRITE PASSWORD File$,Maint$;Pass$(*)
340  Exit:   !
350  DISP "    End of program."
360  STOP
370  Trapperr: !
380  BEEP
390  CURSOR (1,19)                    ! Position cursor above line.
400  DISP "~"
410  SELECT ERRN
420  CASE 56
430  DISP "FILE NAME NOT FOUND. TRY AGAIN."
440  CASE 81 TO 99
450  DISP "FILE SYSTEM (DIRECTORY) PROBLEM";ERRN;"."
460  WAIT 2000
470  GOTO Exit
480  CASE 220
490  DISP "INCORRECT PASSWORD. TRY AGAIN."
500  CASE ELSE                      ! Display any other error here.
510  DISP ERRMS$
520  WAIT 2000
530  GOTO Exit
540  END SELECT
550  WAIT 2000
560  DISP "    "
570  GOTO Start
580  END
```

Index