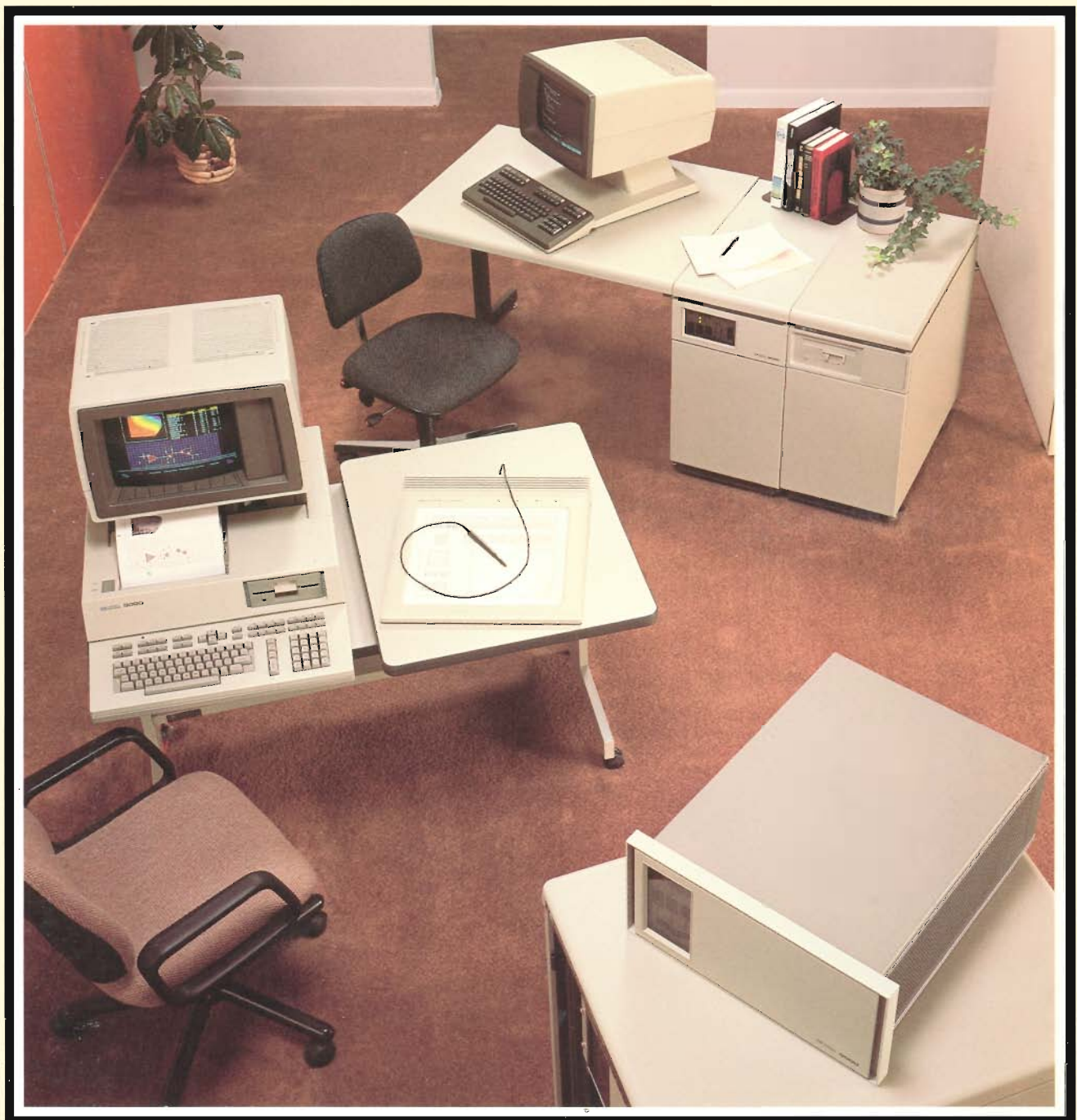


IMAGE/Data Base Programming Techniques



HP Computer Museum
www.hpmuseum.net

For research and education purposes only.



IMAGE/Data Base Programming Techniques

for the HP 9000 Series 500

Manual Part No. 97053-90000

© Copyright 1982, 1983, Hewlett-Packard Company.

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

Use of this manual and flexible disc(s) or tape cartridge(s) supplied for this pack is restricted to this product only. Additional copies of the programs can be made for security and back-up purposes only. Resale of the programs in their present form or with alterations, is expressly prohibited.

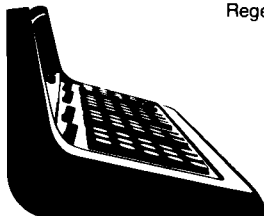
Restricted Rights Legend

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in paragraph (b)(3)(B) of the Rights in Technical Data and Software clause in DAR 7-104.9(a).

© Copyright 1980, Bell Telephone Laboratories, Inc.

© Copyright 1979, 1980, The Regents of the University of California.

This software and documentation is based in part on the Fourth Berkeley Software Distribution under license from the Regents of the University of California.



Hewlett-Packard Company
3404 East Harmony Road, Fort Collins, Colorado 80525

Printing History

New editions of this manual will incorporate all material updated since the previous edition. Update packages may be issued between editions and contain replacement and additional pages to be merged into the manual by the user. Each updated page will be indicated by a revision date at the bottom of the page. A vertical bar in the margin indicates the changes on each page. Note that pages which are rearranged due to changes on a previous page are not considered revised.

The manual printing date and part number indicate its current edition. The printing date changes when a new edition is printed. (Minor corrections and updates which are incorporated at reprint do not cause the date to change.) The manual part number changes when extensive technical changes are incorporated.

December 1982...First Edition

November 1983...Second Edition

Warranty Statement

Hewlett-Packard products are warranted against defects in materials and workmanship. For Hewlett-Packard Fort Collins Systems Division products sold in the U.S.A. and Canada, this warranty applies for ninety (90) days from the date of delivery.* Hewlett-Packard will, at its option, repair or replace equipment which proves to be defective during the warranty period. This warranty includes labor, parts, and surface travel costs, if any. Equipment returned to Hewlett-Packard for repair must be shipped freight prepaid. Repairs necessitated by misuse of the equipment, or by hardware, software, or interfacing not provided by Hewlett-Packard are not covered by this warranty.

HP warrants that its software and firmware designated by HP for use with a CPU will execute its programming instructions when properly installed on that CPU. HP does not warrant that the operation of the CPU, software, or firmware will be uninterrupted or error free.

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

* For other countries, contact your local Sales and Support Office to determine warranty terms.

Table of Contents

Chapter 1: Introduction and Data Base Concepts

General Information.	1-1
Data Base Design.	1-2
Programming Syntax.	1-2
What is a Data Base and Why Use One?	1-2
Data Base Management Steps	1-3
1. Understand Data Base Concepts and Terms	1-3
2. Design the Data Base	1-3
3. Define and Create the Data Base	1-3
4. Access the Data Base	1-3
5. Maintain the Data Base.	1-3
Advanced Data Base Techniques.	1-3
Data Base Overview.	1-4
Data Base Terms	1-5
The Sample Data Base	1-5
The Scenario	1-5
General Information	1-6
Getting Started in BASIC	1-7
Preparing for Daily Use	1-7
Installing QUERY and the LIBRARY Data Base.	1-8
Getting Started in HP-UX	1-8
Data Base Concepts.	1-8
File Management	1-9
Records.	1-9
Serial Access.	1-9
Random Access	1-10
Keyed (Indexed) Access	1-10
Multiple-keyed Access.	1-11
Hashing.	1-11
Collisions	1-12
Representing and Relating Data.	1-13
Entity.	1-13
Attributes	1-13
Attribute Values.	1-14
Representing Data in a Data Base	1-14
Data Item	1-15
Data Types	1-15
Integer Data, Type I2	1-15
Double Integer Data, Type I4	1-16
Shortreal Data, Type R4.	1-16
Longreal Data, Type R8.	1-16
String Data, Type X.	1-16
CODE Data, Type C2 (BASIC Only)	1-17
Array Data Types.	1-17
Data Storage	1-17

Type Extensions (BASIC Only)	1-17
NULL Extension	1-17
PSEUDONYM Extension	1-18
RANGE Extension	1-18
PICTURE Extension	1-18
Data Set	1-19
Data Entry	1-19
Key Items	1-19
Search Items	1-20
Data Chain	1-21
Other Items	1-21
SORT Items	1-22
Storing a Data Set	1-22
Data Base	1-23
Types of Data Sets	1-23
Data Path	1-24
Types of Master Sets	1-25
Master Set Examples	1-25
Master Data Set Summary	1-26
Diagramming a Data Base	1-27
Storing Data Entries	1-27
Accessing Data Entries	1-28
Serial Access	1-28
Serial Access Example	1-28
Directed Access	1-28
Directed Access Example	1-29
Rewinding a Data Set	1-29
Chained Access	1-29
Chained Access Example	1-29
Calculated Access	1-30
Calculated Access Example	1-30
Data Base Files	1-30
Data Base Directory	1-30
Data Base Root File	1-30
Data Set Files	1-31
Passwords and File Names (BASIC Only)	1-31
Permission Bits (HP-UX Only)	1-31
Shared Data Base Access	1-32
Data Security	1-32
Corrupt Data Base	1-33
Data Base Backup	1-33
BASIC and HP-UX Systems	1-34

Chapter 2: Creating the Schema and the Data Base

Introduction	2-1
The Schema	2-1
Creating a Schema	2-2

Schema Struture	2-2
Schema Language Conventions	2-2
Case	2-2
Comments	2-2
Readability	2-2
Syntax Drawings	2-3
Tables	2-3
Restrictions	2-3
Reserved Words	2-3
Schema Syntax	2-4
Data Base Part	2-5
Directory Part	2-6
Password Part	2-7
Item Part	2-8
Set Part	2-11
Schema Instructions	2-15
The BASIC SCHEMA Process	2-18
Creating the ROOT File	2-19
SCHEMA Errors	2-19
Example	2-19
The HP-UX Schema Process	2-23
Creating the ROOT File	2-23
Creating the Data Sets	2-23
Example	2-23

Chapter 3: Data Base Access

Introduction	3-1
Host Language Considerations	3-2
Parameter Data Types	3-2
The Data Base Variable	3-2
The Status Array	3-2
Opening the Data Base - DBOPEN	3-3
Data Base Number	3-3
Specifying the Data Base	3-4
Path Specifiers	3-4
Directories	3-4
Access Capability	3-4
DBOPEN Password	3-4
Passwords and Automatic Masters	3-4
DBOPEN Mode	3-5
Read vs Read/Write	3-5
Sharing of a Data Base	3-5
Buffering and Posting (BASIC Only)	3-6
Closing the Data Base - DBCLOSE	3-6
Ordinary Close - Mode 1	3-7
Forced Close	3-7
Partial Close - Mode 2	3-8
Rewind a Data Set - Mode 3	3-8
Force Buffer Posting - DBCLOSE mode 4	3-9

Reading and Writing Data	3-9
Identifying the Data Items - the List Specifier	3-9
The Form of the List Specifier	3-9
Full Record List Mode (@)	3-10
Item List Mode	3-10
Current List Mode	3-12
Program Variables for Item Values	3-12
Item Data in a Packing Variable	3-12
Item Data in a Variable List	3-13
Adding Data to the Data Base - DBPUT	3-14
DBPUT in a Manual Master	3-14
DBPUT in a Detail Data Set	3-15
Locating a Chain - DBFIND	3-16
Normal DBFINDs	3-16
Shortreal and Longreal Search Items	3-16
Changing the Current Path and Chain	3-17
The Primary Path	3-17
Retrieving Data - DBGET	3-18
CHAINED Reads - DBGET Modes 5 and 6	3-18
Chained Reads in Detail Data Sets	3-19
Chained Reads in Master Data Sets	3-19
Reading a Key Item's Entry - DBGET Mode 7	3-19
Locating a Synonym Chain	3-20
Reading Every Entry - DBGET modes 2 and 3	3-20
Rewind or Select - DBGET mode 4	3-20
Rewinding a Data Set	3-20
Selecting an Entry	3-21
Re-read Current Record - DBGET mode 1	3-21
Deleting a Data Entry - DBDELETE	3-22
Deleting in a Master Data Set	3-22
Deleting in a Detail Data Set	3-22
Modifying Data in the Data Base - DBUPDATE	3-23
Coordinating Access - DBLOCK/DBUNLOCK	3-24
How Not to Use Locking	3-24
The Correct Way to Use Locking	3-25
Locking With and Without Wait	3-25
Locking Portion of Data Base	3-26
What Should You Lock?	3-26
IMAGE Lock Allocation	3-27
Reading the ROOT File - DBINFO	3-28
Why do you need DBINFO?	3-28
Using CODEs and Type Extensions (BASIC Only)	3-28
Using CODEs - DBINFO mode 108 (BASIC Only)	3-28
Using the PICTURE Extension - DBINFO mode 109 (BASIC Only)	3-30
Using the PSEUDONYM Extension - DBINFO mode 101 (BASIC Only)	3-30
Using the RANGE Extension - DBINFO mode 109 (BASIC Only)	3-31

Chapter 4: Data Base Maintenance

Introduction	4-1
The Utility Intrinsic	4-1
Data Base Validation	4-1
The Snapshot - DBSTATUS	4-2
Exhaustive Validation	4-2
Data Base Backup	4-3
Re-entry of Data	4-3
Manual Data Re-entry	4-3
Programmatic Data Re-entry	4-4
Backup Techniques	4-5
Entry Level Backups	4-5
File Level Backups	4-5
Directory and Volume Level Backups	4-6
Partial Backup	4-6
Frequency of Backup	4-6
Backup After Session	4-7
Backup on Modification Count	4-7
Scheduled Backups	4-7
Backup Retention	4-7
Media Selection	4-8
Media Labels	4-8
Media Rotation	4-9
Corrupt Data Base Recovery	4-9
Recovery From a Backup	4-9
Bad Data Recovery	4-9
Entry-level Recovery	4-10
Set-level Recovery	4-10
Volume-level Recovery	4-10
Recovery Without a Backup	4-10
Bad Data Only	4-10
Bad IMAGE Pointers	4-10
Corrupt Root File or File System	4-11
Purged Data Set Files	4-11
Data Base Restructure	4-11
Increasing Data Set Capacity - DBEXTEND	4-11
Changing the Schema	4-12
Cosmetic Changes	4-12
Data Structure Changes	4-12
Set Structure Changes	4-13
Data Base to Data Base	4-13

Chapter 5: Advanced Data Base Operations

Introduction	5-1
Data Base Definition Considerations	5-1
Locating the Data Base	5-1
Multiple Volume Data Base	5-2

Master Data Set Capacities	5-2
Minimizing Synonyms	5-3
Minimizing Secondary Collisions	5-3
High Performance Access Methods	5-4
Minimizing Processing	5-4
Minimizing ROOT File I/O	5-4
DBCONTROL Modes 3 and 4 (BASIC Only)	5-4
Minimizing Data Set I/O (BASIC Only)	5-4
Buffer Use in Reads (BASIC Only)	5-4
Buffer Use in Writes	5-5
Controlling Buffer Posting	5-5
Controlling Buffer Size (BASIC Only)	5-6

Appendix B: BASIC Programming Language Considerations and Examples

Getting Started	B-1
Data Base Item Types	B-1
Invoking the Statements	B-2
Parameters	B-2
List Specifier	B-2
Software Revision Codes and Dates	B-2
Software Installation	B-3
Preparing for Daily use	B-3
Installing QUERY and the LIBRARY Data Base	B-4
Accessing the Packing String	B-4
The PACKFMT Statement	B-5
The PACK USING and UNPACK USING Statements	B-6
BASIC Language Examples	B-7
BASIC Main Program	B-7
DBOPEN Example	B-9
DBCLOSE Example	B-10
DBPUT Examples	B-11
CHAINED DBGET Example	B-12
DBUPDATE Example	B-13
DBDELETE Example	B-14
DBINFO Example	B-15
Error Reporting Subprogram	B-16

Appendix C: C Programming Language Considerations and Examples

Data Base Item Types	C-1
C Library Calls	C-1
Parameters	C-1
C Language Example	C-2
Dbopen Example	C-2
Dbinfo Example	C-3
Serial Dbget	C-3
Buffer Unpacking Routine	C-4
Print Error Message Routine	C-4
Buffer Pack Routine	C-4

Print Element of Buffer Routine	C-5
Terminal Conversion Routine	C-6
Print Data Base Entry Routine	C-7

Appendix E: Errors

IMAGE Status Errors	E-1
IMAGE-Related Errors	E-2
SCHEMA and QUERY/DEFINE Errors	E-3
General Error Messages	E-3
Specific Error Messages	E-3

Appendix F: Fortran Programming Language Consideration and Examples

Data Base Item Types	F-1
Fortran Library Calls	F-1
Fortran Parameters	F-2
Fortran Language Example	F-3
Set Up Arrays	F-3
Dbopen Example	F-3
Serial Dbget	F-4
Print Results Routind	F-4
Dbclose Example	F-4
Error Return Routine	F-4

Appendix G: Glossary

Appendix I: HP IMAGE Data Base Compatibility

Introduction	I-1
Specifications	I-1
Data Bases	I-1
Passwords	I-1
Data Sets	I-2
Data Paths	I-2
Data Entries	I-2
Data Items	I-3
Data Item Extensions	I-3
IMAGE Intrinsic	I-4
Implementation	I-4
Host Languages	I-4
Syntax	I-4
DBBEGIN	I-4
DBBUFFER	I-5
DBCLOSE	I-5
DBCNTOL	I-5
DBCREATE	I-5
DBDELETE	I-5
DBEND	I-5
DBERASE	I-6

DBERROR	I-6
DBEXPLAIN	I-6
DBEXTEND	I-6
DBFIND	I-6
DBGET	I-6
DBINFO	I-7
DBLOCK	I-8
DBMEMO	I-8
DBOPEN	I-8
DBPURGE	I-8
DBPUT	I-9
DBSTATUS	I-9
DBUNDO	I-9
DBUNLOCK	I-9
DBUPDATE	I-9
Data Base Backup	I-10
Miscellaneous	I-10
QUERY	I-11

Appendix P: Pascal Programming Language Considerations and Examples

Data Base Item Types	P-1
Pascal Library Calls	P-1
Pascal Aliasing	P-2
Pascal Parameters	P-3
Pascal Language Example	P-3
Type Declarations	P-3
Variable Declarations	P-4
External Declarations	P-4
Error Message Routine	P-4
Print Entry Routine	P-5
Main Program	P-5
Dbopen Example	P-5
Dbfind Example	P-6
Chained Dbget Example	P-6
Dbclose Example	P-6

Appendix Q: QUERY Type Extensions

Introduction	Q-1
PICTURE Interpretations and IMAGE Data Types	Q-1
Default PICTURE Word	Q-2
PICTURE Words TIMEDATE, DATE and TIME	Q-2
PICTURE Word NAME	Q-3
PICTURE Word Numeric	Q-3
User-Defined PICTURE Word	Q-6

Appendix T: Useful Tables

HP 9000 Character Codes	T-1
HP 9000 ASCII Control Character Codes	T-1
HP 9000 ASCII Displayable Character Codes	T-2
HP 9000 BASIC Extension Control Character Codes	T-4
HP 9000 BASIC Extension Displayable Character Codes	T-5
Prime Numbers	T-7
PRIME-FINDER Program	T-7
Determining File Sizes	T-8
ROOT File Directory	T-8
ROOT File	T-8
Data Set File Directory	T-8
Master Data Set	T-9
Linked Detail Data Sets	T-9
Stand Alone Detail Data Sets	T-9
Media Records	T-10
Master Data Set Records	T-10
Detail Data Set Records, Chained	T-11
Empty Records	T-11
Occupied Records	T-12
Detail Data Set Records (stand_alone)	T-12
Empty Records	T-12
Occupied Records	T-13

Appendix U: Upgrading From IMAGE/45 to IMAGE/9000

Introduction	U-1
The Export/Import Utilities	U-1
DBXPRT Instructions	U-2
DBIMPORT Instructions	U-2

Chapter 1

Introduction and Data Base Concepts

General Information

IMAGE/9000 is a set of programs and procedures which you can use to define, create, access and maintain a data base. The first chapter of this manual is intended for those who use and write data base programs. The balance of the manual is designed for those who intend to write programs to access an IMAGE/9000 data base in the following host environments:

- HP 9000 BASIC Language System;
- HP 9000 HP-UX Operating System.



Note

Throughout this manual IMAGE commands and statements are written in all capital letters. This is the way you would see them in the BASIC Language System. In the HP-UX System, they are in all lowercase letters. The commands and statements are the same in both systems except where noted.

The purpose of this manual is to:

- introduce the concepts and terms of data base management;
- introduce the basics of data base programming and maintenance;
- discuss advanced topics primarily concerned with data base performance.

Completion of a Hewlett-Packard Data Base programming course is strongly recommended, but not required. This manual assumes that you are familiar with the following material:

- programming in the desired host language (BASIC, FORTRAN, Pascal or C);
- file I/O programming in the host operating system (BASIC, HP-UX).

Data Base Design

Data base design is an essential step in using a data base management system. A data base should be designed after you are familiar with the terms and concepts presented in this chapter. A Data Base Design Kit, consisting of a manual and worksheets, is provided with IMAGE to assist you in the creation of data bases.

Programming Syntax

This manual is intended to be used in conjunction with a language or library reference manual supplied with your system. The reference manual contains an alphabetical listing of the data base intrinsics (statements or library subprograms) applicable to your system and programming language. The Reference manual provides full details of the syntax and limitations of each statement. Once you have read and understood this techniques manual, the reference manual should provide all the information necessary for program development.

What is a Data Base and Why Use One?

A data base is a collection of logically-related files. The files contain data and information about the data. Related data items are linked by pointers, and all data items are identified by type and size.

The principle benefit of a data base management system is productivity. When compared to a program which accesses conventional files, a data base application program can be developed in less time and requires less on-going maintenance. There are other benefits which are difficult or impossible to obtain with conventional files. Some examples are:

- Programs can be largely (or completely) independent of the data base structure. Changes to the data base do not automatically require changes to the applications programs.
- Data storage can be consolidated. Two different applications which require one or more of the same data items need not have different, dedicated, files with duplicated information. All applications can use the same data base.
- Data is more secure. A user who has the capability to access an ordinary file can access **all** of the data in that file. A data base user can be restricted to a subset of the data in the data base.
- A data base supports a variety of methods of accessing data. Data in a conventional file normally has a limited number of access methods and sometimes only one (serial). You can access data sets (the data base equivalent of files) in seven different ways.
- Data entries (the equivalent of records) are associated in a data base. Entries with related data are linked together. Exhaustive searches are usually not required to locate an item of interest. Information which is frequently sorted when retrieved can be transparently stored in a pre-sorted order in a data base.
- Support and maintenance tools are provided with a data base. You can create, back-up and restore a data base without writing any programs. If QUERY is available, other data base maintenance and ad-hoc reporting can be performed without programming.

Data Base Management Steps

There are five basic tasks to be performed before using your data base:

1. Understand Data Base Concepts and Terms

It is essential to understand the fundamental concepts and terms in order to use IMAGE effectively. The remainder of this chapter is devoted to these ideas.

2. Design the Data Base

Determining how to organize your data is the most important step in using your data base management system. The **Data Base Design Kit**, HP part number 97053-90002, provides procedures and tools to make the conversion from application needs to a data base solution as straightforward as possible.

3. Define and Create the Data Base

Once you have designed the data base, it is a simple process to convert your logical design (or Design Kit worksheets) into a **schema**, or formal definition of the data base. The SCHEMA utility program uses the schema text to create the data base **ROOT file**, which records the internal structure of the data base. With BASIC QUERY/9000, you can also create the entire data base interactively using DEFINE. The next chapter discusses the schema and the creation of the data set files.

4. Access the Data Base

Any time you use your data base you must **open** it. Then you can add, retrieve, modify, or delete data. When access is complete, the data base must be **closed**. This process is discussed in the "Data Base Access" chapter.

5. Maintain the Data Base

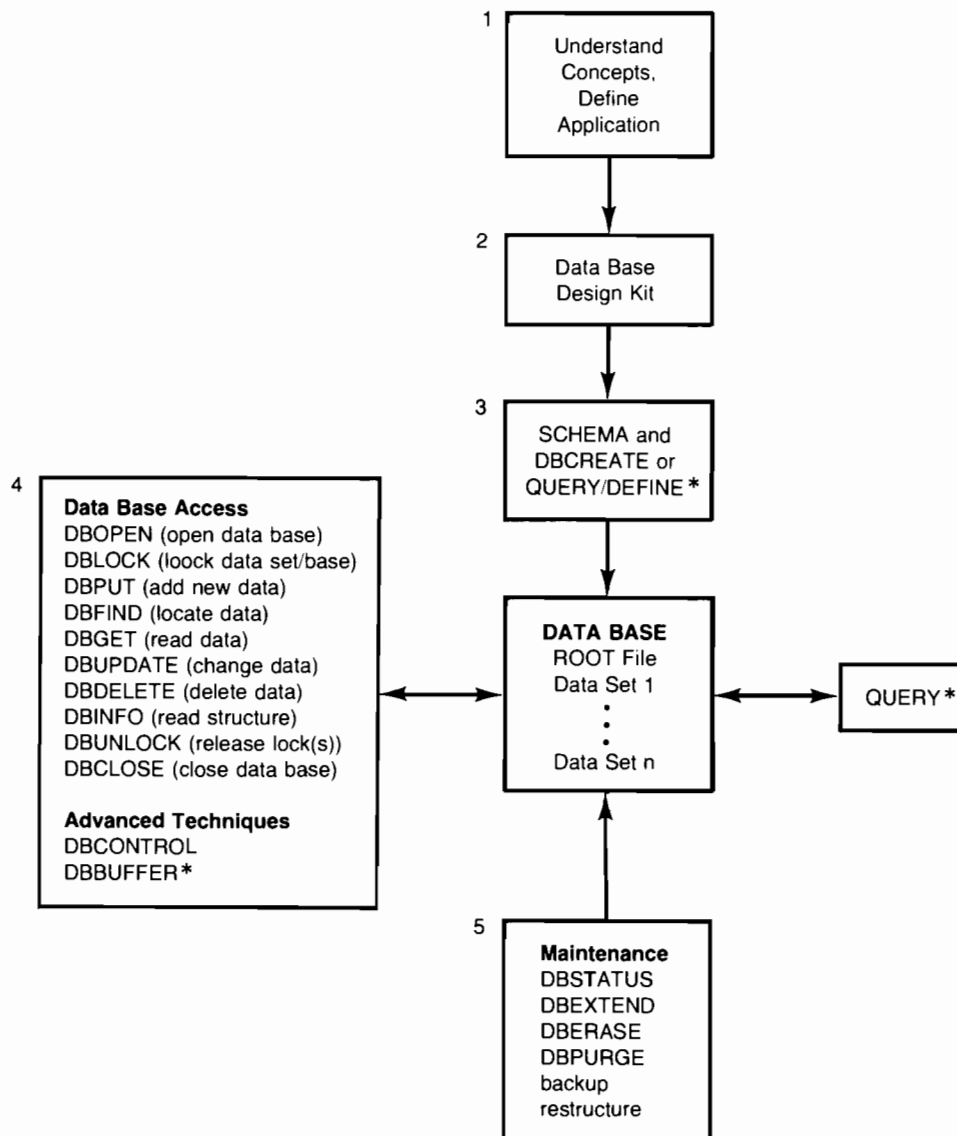
Backing up should be a regular part of your data base operations. Restructuring the data base or performing large-scale erasure may also be needed from time to time. The "Data Base Maintenance" chapter discusses these operations.

Advanced Data Base Techniques

The "Advanced Data Base Operations" chapter discusses issues primarily related to performance, such as memory allocation, multiple volumes, and access efficiency.

Data Base Overview

The following block diagram shows the relationship of the IMAGE software modules and the data base management steps just described.



* These modules are not currently implemented in HP-UX.

Data Base Terms

Most data base terms are defined as they are introduced in this manual. Many appear in the Glossary of this manual. Several are not related to any specific topic and need to be introduced here.

The term IMAGE refers to the software which you add to your system to access a data base. In the BASIC Language System, IMAGE is an optional BIN file which adds data base statements to the BASIC language. In HP-UX, IMAGE is a set of commands and a library of system software which your programs call as subroutines. The names of these IMAGE intrinsics, for example, DBOPEN, are the same in both operating systems.

In both host environments, the term password has several meanings. In the BASIC Language System, you can restrict access to volumes and files (such as directories) with volume passwords and file passwords. In HP-UX your login password determines your file access capabilities. There are also passwords within an IMAGE data base which restrict access within data sets. When this manual uses the term password by itself or in the expression DBOPEN password, it refers to the internal data base passwords. All other uses of password are prefixed with the term volume, file or login.

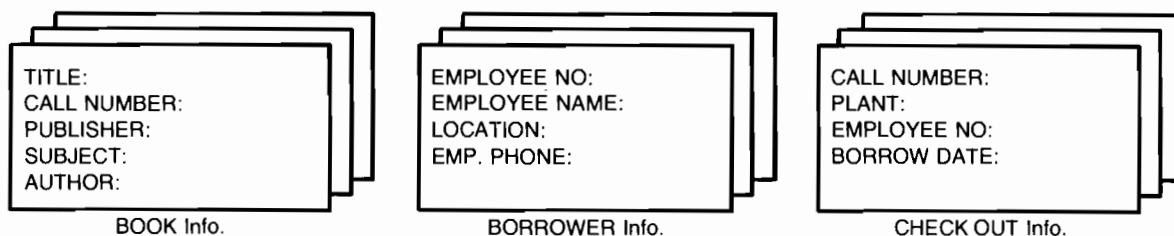
In addition to DBOPEN passwords, each data base can have a single maintenance word. This is a special string required only for certain "utility" data base intrinsics.

The Sample Data Base

Throughout this manual, a sample data base application is used. A copy of this data base containing sample data is supplied with your IMAGE product. It is a simple example and allows you to become familiar with and perform all the data base steps, from concepts to maintenance, before you create your own.

The Scenario

The NOP Software Company has eight facilities. Each has a small library of reference books. The books may be checked out by any employee from any facility and kept as long as needed. Information concerning books and borrowers used to be maintained on cards like the ones in the following illustration.



1-6 Introduction and Data Base Concepts

The card system is no longer satisfactory. NOP cannot determine easily what books are in or out, and the system does not lend itself to simple analysis such as: What is the value of the inventory?

NOP would like to record or track the following information:

- card catalog: Book title, author, call number¹, publication date, publisher, cost, and subject.
- the entire book list, identifying the owning facility, or current location;
- checkout information: employee name, number, department, and extension;
- branch information: Facility name and address, librarian name and extension;
- classification of each book by subject.

Using IMAGE, books can be found by title, author, subject or call number. The call number can be used to determine whether or not a book is checked out (and to whom) and if there are any other copies available. Additionally, all books on a particular subject can be found.

General Information

You need to know this information to access the data base:

Data Base Name: LIBRARY

Passwords: LIBRMGR
ENGINEER

Maintenance Word: BOOKS

¹ The call number used is not related to any particular library numbering system.

Getting Started in BASIC

Flexible discs wear out after approximately 250 hours of disc rotation. The flexible discs supplied with your IMAGE product, the distribution media, should be used only as the backup or archival copies of your software. You should make a backup copy of your IMAGE software before using it. There are two ways to accomplish this:

- If you have an internal “sealed” or an external disc, you can copy the distribution media to it. You can copy the QUERY files and the example LIBRARY data base with the BUILD_QUERY program. BUILD_QUERY requires that the IMAGE option BIN file IMAGE_DBM and the BASIC option BIN file MS be loaded in your system. You can copy the remaining files with the BASIC COPY statement. Instructions for the BUILD_QUERY program are in one of the following sections.
- If the internal flexible disc is your only mass storage media, you can use the “DISC_BACKUP” utility supplied with your BASIC Language System software to copy each IMAGE disc to another flexible disc.

Note

You should copy the IMAGE software files only to a disc having the Structured Directory Format (SDF). Although you can copy QUERY and the utility programs to any format disc, you cannot copy the LIBRARY data base to a LIF or 9845 format disc.

Preparing for Daily use

Once you have an on-line copy of your IMAGE software, you need to consider how often you will be running data base programs, and how to most conveniently load the IMAGE_DBM file into the BASIC Language System.

To execute the QUERY, SCHEMA, DBXPRT, DBIMPORT, BUILD_QUERY programs and your own data base programs, IMAGE_DBM must be loaded in your system. There are four ways to accomplish this:

- If you run data base programs infrequently, you can load file IMAGE_DBM each time you require it. You use the BASIC LOADBIN command, for example:

```
LOADBIN "IMAGE_DBM:CS80,7"
```

- If your data base programs require other option BIN files, you can load the required files once with individual LOADBIN commands and then store the entire collection in a single file with a STOREBIN command, for example:

```
STOREBIN "MY_NAME<Password>/MY_BINS:CS80,7"
```

When you run your applications, you need LOADBIN only the composite file.

- If you run data base programs frequently, and have an adequate quantity of memory in your computer, consider storing the frequently required BINs in the system's “AUTOBN” file. That way, the BASIC system automatically loads the required BINs at power-up.
- As an alternative to using the “AUTOBN” file, you can also use the “BUILD_BOOT” utility program supplied with your BASIC system to configure the required BIN files in the “BOOT” area of the system disc. The system then powers-up with the required BIN options already present.

Installing QUERY and the LIBRARY Data Base

Once you have the IMAGE_DBM and MS BINs loaded, refer to the QUERY/9000 User's Guide for instructions for installing QUERY.

Getting Started in HP-UX

The IMAGE product is supplied on a magnetic tape cartridge. You need to transfer the files from the tape to your HP-UX system. The HP-UX System Administrator Manual (p/n 97089-90045) describes the programs and procedures necessary to do this.

Use the following steps to do this.

1. Log onto the system as the superuser.
2. Place magnetic tape into the tape drive. Wait for the tape's busy light to go out.
3. Use *optinstall* to transfer the files from tape to disc. The IMAGE product number is 97083A for multi-user IMAGE and 97073A for single-user IMAGE.
4. Use *revck* on file */system/97083A/revlist* (or */system/97073A/revlist*) to check that all files have been transferred.
5. Remove tape from tape drive using the *tcio* program.
6. Re-boot the system using *stopsys -r*.

This procedure should be done by the system administrator as it requires that the system be shut down and stopped.

A demo data base for IMAGE-UX is included in */usr/lib/image/demo*. It consists of:

- the sample LIBRARY data base used throughout this manual;
- sample Fortran, Pascal and C source programs;
- a set of useful commands to do simple operations on a data base.

Data Base Concepts

A data base management system lets you represent, relate and store data without worrying about how it is organized and maintained. You can store information about things and how they are related, and retrieve it later. If you are familiar with data base management systems (DBMS), you know that there are three primary ways to organize data: hierarchical, relational, and network. IMAGE uses a simple network scheme.

File Management

Data can be stored on a mass storage medium for later use. To use a data base effectively, you should be familiar with conventional mass storage operations. A data base is fundamentally a highly structured file system. This section covers various file types and access methods. This material is presented to help you understand how data is stored in a data base.

Records

Files are composed of records. Each record may contain one or more data values. Each value occupies a position in the record, and values of the same type usually occupy the same position in each record. In conventional files, this position is called a **field**. In a data base all values of the same type are called **items**. The following is a sample record from a conventional file:

Record data	03050969	Hooper, Grace M	40	4321	9132
-------------	----------	-----------------	----	------	------

As you can see, the raw data is of little value unless a good deal more is known about it, for example:

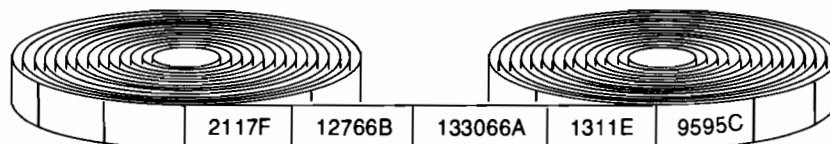
Data Identity	employee number	employee name	facility	dept	phone extension
Data Type	double integer	character	integer	int.	integer
Data Size in Bytes	4 bytes	30	2	2	2

With a conventional file it is the responsibility of each application program to “know” this information about the record. A record change may require changing all the programs which access the file. In a data base the program code does not need to reflect the actual location, type, or size of the item. An item is accessed simply by its name (or number).

Serial Access

The most basic method of storing and retrieving records is serial access. A magnetic tape is an example of a medium which allows only serial access. Even if you know which record you want, serial access has several disadvantages:

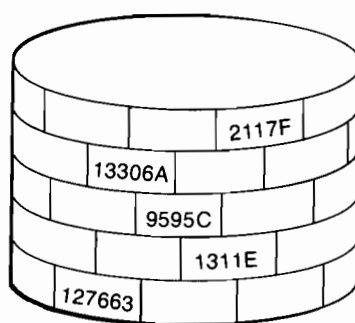
- all records between the current position and the desired record must be accessed (and perhaps read and examined) to obtain the desired data;
- new records can only be added at or after the last existing record;
- backspacing of records may not be available, requiring a “rewind” of the entire file to access records prior to the current record;
- it may not be possible to re-write (update) existing records.



Serial Access

Random Access

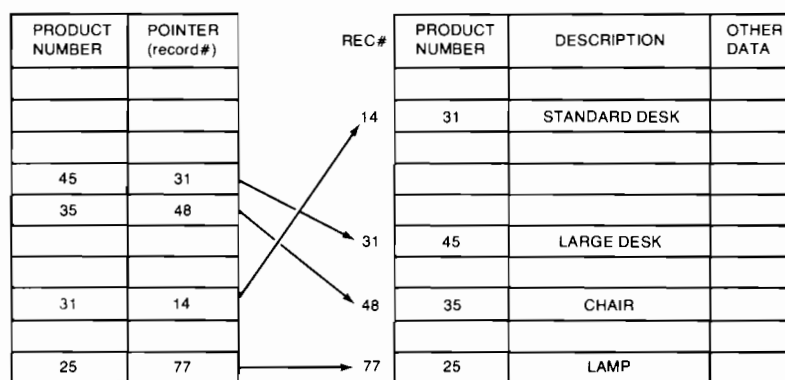
With random access, any specific record (selected at random) can be accessed as quickly as any other in the file. Your computer's disc drive is an example of a random access device. Random access files eliminate most of the disadvantages of serial files. However, unless a structure is imposed on the file, it may still be necessary to serially search the entire file for a given item value.



Random Access

Keyed (Indexed) Access

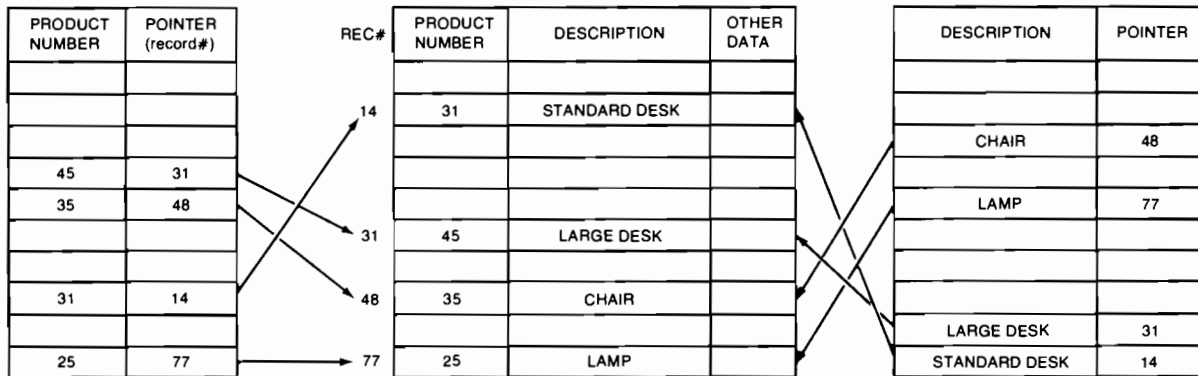
Random access allows fast access to any record, but unless the record number is known, a search may be required. One way to determine the record number is to set up a separate "look-up" or **index** file. This file contains the value of items of interest (called keys), and the corresponding record number (called a pointer) in the main data file. Finding the key item in this index file can be relatively fast if it is small enough to be in sorted order, or have a record for every possible key item value.



Keyed Access

Multiple-keyed Access

If it is necessary to access more than one item in a file quickly, an index file can be maintained for each key item.

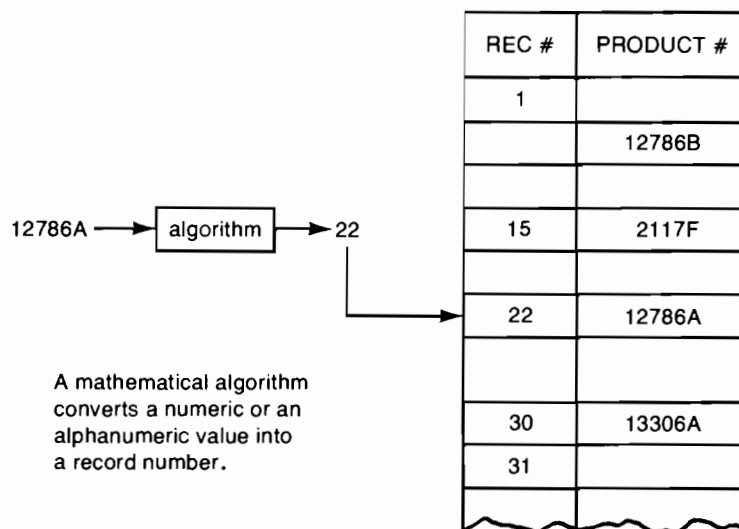


Multiple-keyed Access

Hashing

Keyed file access is only as fast as the process of locating the key item value of interest in the index file. Even if the index file is sorted and the technique known as a binary search can be used, it may be necessary to examine $\log_2 N$ records, where N = the number of records in the file.

A more efficient method of organizing the index is to compute the record number from the key value. The same algorithm is used on all accesses to the same file. Because the record number generated bears no apparent relationship to the key value, the key is said to have been “hashed”. The hashing is not random because the same key value always produces the same record number.



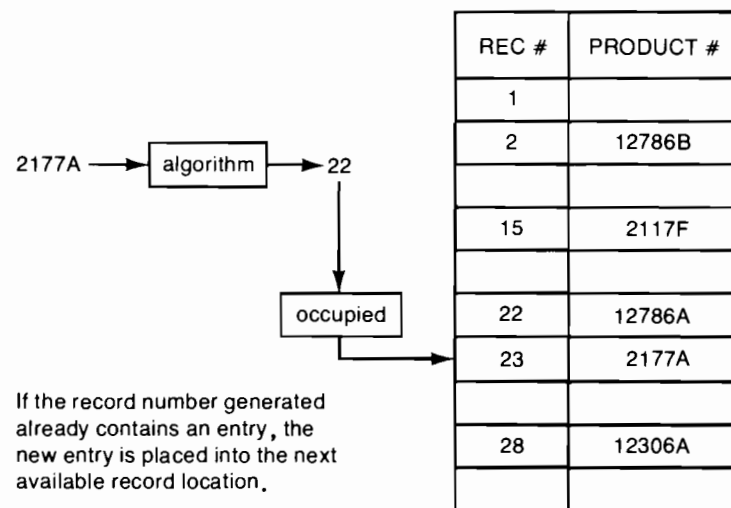
Hashing

Collisions

Hashing algorithms are chosen so that they generate an even, nearly uniform, distribution of record numbers for all key values. However, a different key item value can hash to a record already occupied; this is called a collision. A colliding key value is called a **synonym**.

The record to which a key value hashes is known as its **primary address**. If a key value collides with an occupied record, it is placed at a **secondary** location.

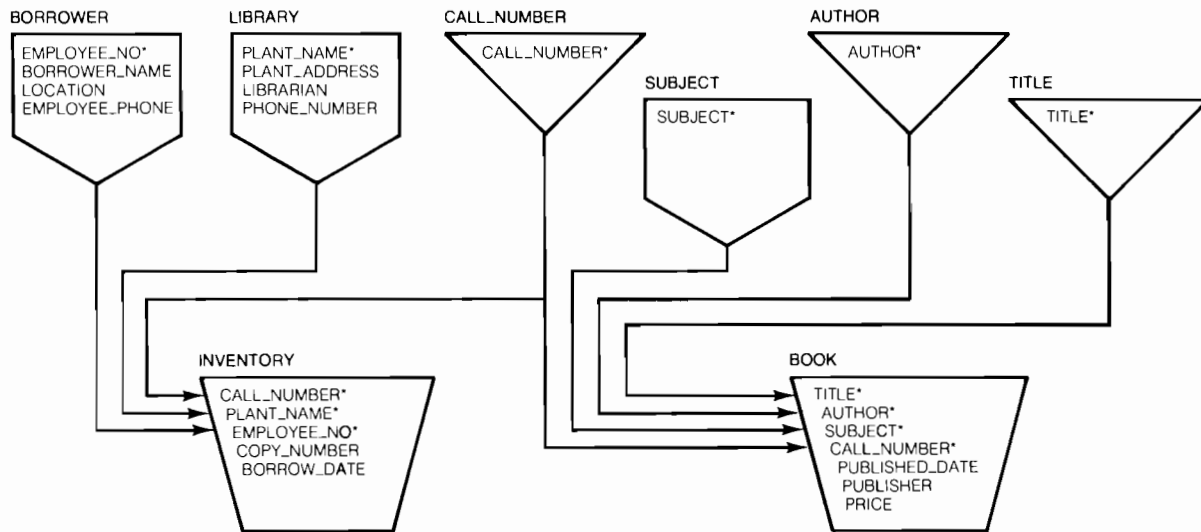
When collisions occur in IMAGE, the synonym is placed in the next available record in a chain of free records. In some other HP IMAGE data bases, a secondary (synonym) record migrates (is relocated) if the primary record becomes available, or if another new key hashes to the secondary record. In IMAGE/9000 there are no "migrating secondaries". See the "Advanced Data Base Operations" chapter for a discussion of secondaries and performance.



Collision

Representing and Relating Data

The sample data base of the NOP Library is shown in the following diagram. The diagram shows how the data is represented and related, and is used in the following pages to explain how IMAGE represents and relates data.



Entity

A data base is used to keep information about things, or **entities**, and relationships between things. For example, the sample library data base keeps information about three entities: books, borrowers and library branches. It also keeps information about the relationship between things: the inventory is defined in terms of books, their borrowers and the library branch where they are kept.

Attributes

Each entity is described and defined by its various **attributes**, which are identifying characteristics of an entity. The attributes of book, borrower and library branch are shown in the following table.

Entity	Attribute
Book	Title Author Call Number Subject Publisher Publication date Cost
Borrower	Name Employee Number Location Phone number
Library branch	Plant name Plant address Librarian Phone number

1-14 Introduction and Data Base Concepts

Attribute Values

Each book in the library is identified by giving values to its attributes. The following example shows the attribute values for two books.

Attribute	Attribute Values	
Title	Programming Proverbs	Thin Films
Author	Henry Legard	J.M.Poate
Call Number	74122058	77125348
Subject	Programming	Semiconductors
Publisher	Hayden	Wiley
Publication date	February 17,1975	July 17,1978
Cost	12.95	11.85

Representing Data in a Data Base

A data base management system provides you with a way to represent entities, relationships between entities, attributes and attribute values. These means are covered in the following sections.

Data Item

Attributes are represented in IMAGE by data items, which are the smallest accessible data elements. Data items are given values and can correspond to program variables. Each data base can have up to 1023 data items, each of which is represented by a unique data item name, and up to four pseudonym names.¹

Data Types

IMAGE supports six data types in BASIC and five data types in HP-UX. Each can be a scalar or an array. Your program can further restrict the range or use of an item if one or more type extensions are specified for the item.

Refer to the appropriate language appendix for a discussion of which language variable type designator to declare for the various IMAGE item types.

The following table summarizes the item types, element ranges, and type extensions:

Type	Des ²	Element Range	Type Extensions
integer, integer array	I2	– 32 768 thru 32 767	NULL, PICTURE, PSEUDONYM, RANGE
double integer, double array	I4	– 2 147 483 648 thru 2 147 483 647	NULL, PICTURE, PSEUDONYM, RANGE
shortreal, short array	R4	±3.402 823 E + 38 thru ±1.175 494 E – 38 and 0	NULL, PICTURE, PSEUDONYM, RANGE
longreal, long array	R8	±1.797 693 134 862 315 E + 308 thru ±2.225 073 858 507 202 E – 308 and 0	NULL, PICTURE, PSEUDONYM, RANGE
String, String array	Xn	CHR\$(0) thru CHR\$(255) length 1 thru 2 147 483 647; length must be an even number in HP-UX	PICTURE, PSEUDONYM
CODE, CODE array	C2	1 thru 32, each number represents a user- defined string	PICTURE, PSEUDONYM

CODE type and Type Extensions apply to the BASIC Language system only. They are not supported in the HP-UX system.

Integer Data, Type I2

An integer item requires two bytes of data storage per data element. Chose the integer type when four significant digits are adequate to represent the item, and you want to conserve storage space.

Integers are stored in 16-bit two's complement integer binary form and are useful in representing search item values.

¹ Pseudonym in BASIC only.

² DEFINE/SCHEMA designator.

Double Integer Data, Type I4

A double integer item requires four bytes of data storage per data element. Chose the double integer type when more than four and less than ten significant digits are adequate to represent the item.

Double integers are stored in 32-bit two's complement integer binary form and are useful in representing search item values.

Shortreal Data, Type R4

A shortreal item requires four bytes of data storage per data element. Chose the shortreal type when seven significant digits are adequate to represent the item's mantissa but its magnitude cannot be represented by a double integer.

Shortreals are stored in 32-bit IEEE floating point binary form. This form cannot precisely represent all rational values within its range. You should be cautious in using shortreals as search items or in performing arithmetic comparisons of shortreal values. To assure proper hashing and comparisons of shortreals, use rounding to limit the significant digits stored or compared.

Longreal Data, Type R8

A longreal item requires eight bytes of data storage per data element. Chose the longreal type when more than seven but less than 16 significant digits are adequate to represent the item's mantissa, or when the range of the exponent of the shortreal type is inadequate to represent the magnitude of the item.

Longreals are stored in 64-bit IEEE floating point binary form. This form cannot precisely represent all rational values within its range. You should be cautious in using longreals as search items or in performing arithmetic comparisons of longreal values. To assure proper hashing and comparisons of longreals, use rounding to limit the significant digits stored or compared.

Items treated by QUERY as TIMEDATE, DATE, and TIME (according to their PICTURE words) must be longreal.

String Data, Type X

Strings can be used to store any items which can be represented as a concatenation of 8-bit bytes. Normally, strings are used to store HP 9000 character data.

Strings require one byte for each position in the string.

Strings can be used to store numeric values in two ways:

1. Each individual byte can represent a value in the range 0 thru 255. In BASIC the CHR\$ function can store a numeric value in a string, and the NUM function can retrieve it.
2. The entire string can represent a numeric value with any number of digits. For example, an integer of 15 digits can be stored in a string. You can use a string numeric as a search item with much more confidence of an exact match than if a longreal is used. This is at the expense of some speed in the performance of the compare operation.

CODE Data, Type C2 (BASIC Only)

CODE is an enumerated data type. When a CODE item is defined, a relationship is established between CODE values and CODE numbers. A CODE value is one of a maximum of 32 user-defined 16-character strings. A CODE number is a 16-bit integer in the range 1 thru 32. The CODE type is provided so that an item can have meaningful string data values without requiring a large amount of string storage.

The following lines are an example from a SCHEMA ITEM PART. This declaration assigns the code numbers one through eight to the code values (strings) WHITE through BLACK. When data values of "WHITE" through "BLACK" are entered in applications programs, the program can obtain the code number with DBINFO mode 108. The code number is stored in the data base, not the string.

```
COLOR,C2 CODE="WHITE","RED","YELLOW","GREEN","CYAN",
          "BLUE","MAGENTA","BLACK"
```

The CODE type requires two bytes of data storage per element.

To use a CODE value in your program requires that you "look-up" the code number. IMAGE does not automatically perform this correlation. DBINFO mode 108 returns the table of code values.

Array Data Types

Each item type can be an entire array of that type. An array can have one to six dimensions, and up to 2 147 483 647 total elements.

Arrays are stored as a list of elements. Only the actual data elements of the array are stored in the data base. You can use a program array of any order when storing and retrieving an array from a data base as long as the program array is at least as large as the data base array in total number of elements.

Data Storage

The figures given above for data storage requirements apply only to the data. Additional room is required for pointers linking the data entry to other entries in the data set. See Appendix T for detailed information about data entry formats.

Type Extensions (BASIC Only)

A type extension consists of information (in the form of tables) stored in the root file for the item when it is created with DEFINE or SCHEMA.

IMAGE does not automatically test data values against extension information. With the exception of the NULL type extension, it is the responsibility of the application program to validate each data item against its type extensions. You can check data values against this information with DBINFO.

NULL Extension

In a form of DBPUT, specific values for items to be added or changed can be omitted. This signifies that the "null" or default value for each item is to be used. Null values exist for all item types and can be explicitly specified for all numeric types.

A numeric item's currently defined NULL value can be obtained via DBINFO mode 106 or 109.

If the NULL type extension is not specified during DEFINE or SCHEMA, the default null value is zero for item types I2, I4, R4, R8. For item type X (string) the null value is always an ASCII space-filled (CHR\$(32)) string. For CODE (C2) it is always zero. If you are also using the RANGE extension, you may want to choose a NULL value which is outside the RANGE.

PSEUDONYM Extension

Each item in a data base has a name which uniquely identifies the item. You can also designate up to four alias, or **pseudonym** names by which the item is known. Pseudonyms are recommended when the principal item name is time-consuming to type or is easily forgotten. For example, in our sample data base, the item AUTHOR can also be accessed via the pseudonyms NOVELIST, WRITER, POET and AUTHOR_NAME.

IMAGE automatically correlates the principal item name with the item number used to access the data base. IMAGE does not correlate pseudonym names. If your program is to support the use of pseudonyms, it must execute a DBINFO mode 101 request to determine the principal item name or number. You can also use DBINFO mode 107 to return the entire PSEUDONYM table.

RANGE Extension

Your application may require that the value of a numeric item be within a sub-set of the range allowed by the data type. That range can be stored in the root file when the data base is defined, and retrieved by your program via DBINFO mode 109. IMAGE does not automatically range-check values added to the data base.

If no RANGE is specified, the full range of the data type is stored in the root file.

PICTURE Extension

The PICTURE extension is a 32-bit double integer value stored in the root file. You can use it to assign an interpretation to a data item type.

Typical interpretations might be:

- names of people are converted by your program to lastname,firstname middlename format before being stored in the data base;
- the item value is the time of day, in seconds since midnight;
- the item value is a date. When displayed, the month is to be spelled, but abbreviated.

The value of the PICTURE extension has interpretations which are defined in QUERY as sub-types DATE, NAME, numeric format, TIME and TIMEDATE. These sub-types can be specified in QUERY/DEFINE.

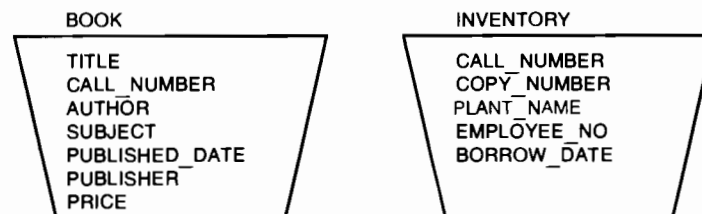
SCHEMA supports the PICTURE extension only as the specification of a double integer constant. You can use the constants defined for the QUERY types. They are listed in Appendix Q. IMAGE ignores the picture word during data base operations.

You can retrieve the value of the PICTURE extension with DBINFO mode 109.

Data Set

A collection of entities with identical attributes (but not values) is represented in IMAGE by a **data set**. A data set can also represent a connection or relationship between two entities. A data set is a collection of data items that describe the entity or relationship. You can have up to 1023 data items in a data set.

Here are two examples of data sets. The BOOK data set keeps information about an entity, a book. The INVENTORY data set keeps information about the relationship between two entities, book and borrower.

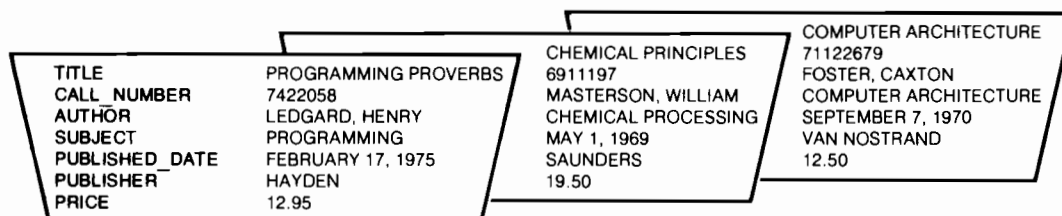


Data Entry

An instance of an entity is represented in a data base by a data entry. A data entry, or record, is a collection of values for the items in a data set; these values describe a particular entity or relationship. A data entry can be up to 2 147 483 643 bytes in length, but is limited in practice to the available space on the mass storage media.

A data set is composed of a number of data entries. When you define a data set, you specify the maximum number of data entries that it can have unless extended with DBEXTEND. The greatest possible number of data entries is 2 147 483 647, limited in practice by storage media space.

Here are three sample entries from the BOOK data set:



Key Items

Normally, each data set has one or more data items by which a data entry is identified or retrieved. These items are called key items. Key items are indicated with an asterisk (*) in the diagram of the sample data base. For example, a particular book can be identified by its CALL_NUMBER, so this is a key item. Additionally, since one use of the sample data base is to identify books on a particular subject, SUBJECT is also a key item.

Key items also tie together related data entries that have the same value for the key item. For example, all books that have the same author (AUTHOR is a key item) are automatically linked together.

A third use for key items is to act as a link for information in different data sets. For example, the INVENTORY set contains information about which employee has checked out which books. If you know an employee number and want to find out the titles of the books checked out to that person, the key item CALL_NUMBER links the BOOK and INVENTORY data sets.

The important points about key items are:

- Key items are stored only in master data sets.
- Each key item in a master data set has a unique value.
- Key items cannot be array items or strings longer than 256 bytes.
- Key items are located by the calculated access method.

Search Items

“Key item” is so named because its entry in a master data set provides the key to related entries in linked detail data sets. The keyed access method is used to find the first entry in a detail data set which has the same value as the key item.

Within a detail data set you can locate additional entries with the same key item value by searching a chain of linked entries. Therefore, within a detail data set key items are referred to as search items, not key items. For example, EMPLOYEE_NO is a key item in the BORROWER data set and is a search item in the INVENTORY data set.

The important points about search items are:

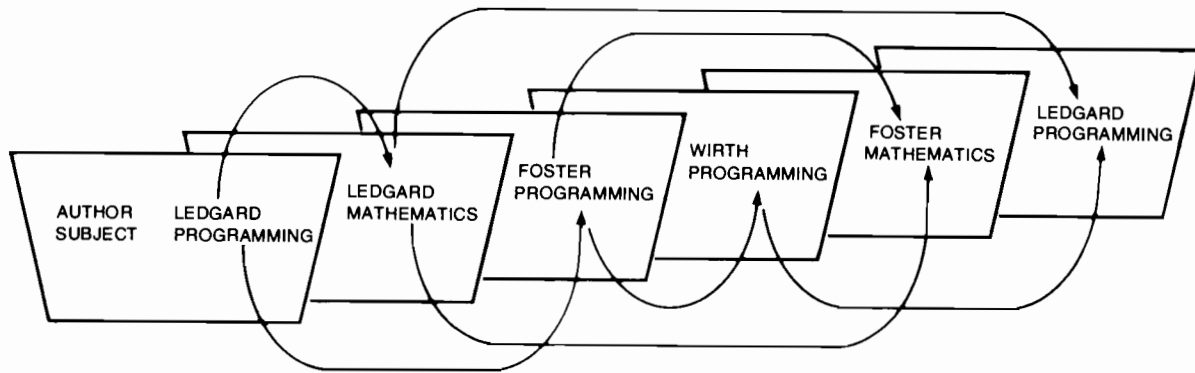
- Search items are located only in detail data sets.
- The same search item value can occur more than once in the data set.
- Identical search item values are linked by a chain. The chain head is the key item in a master set having the same value.
- Search items cannot be array items or strings longer than 256 bytes.

One unique key item value in a master data set is linked to zero or more identical search item values in zero or more detail data sets. The items may not have the same name in each set, but they always have the same structure (type and size).

Data Chain

A data chain is used to link together data entries having search item values which are identical to the value of the linked key item. This makes it easier to retrieve entries that are alike in some way. The first entry in a chain is the chain head. IMAGE automatically keeps track of the pointers necessary to link the entries in a chain.

For example, a data chain links together all the books having the same subject. Another chain links books with the same author.



The chain of related data entries is a doubly linked list. Each entry has a forward and a backward pointer denoting the next and previous entry on the chain. When you add a new entry to a chain, it is linked to the end of the chain, that is, the chain is sorted in chronological order. The "Sort Items" section discusses how to sort a chain according to item values.

Other Items

In both manual master and detail data sets you can store items which are neither key items nor search items. These items frequently constitute the bulk of your stored data. They are the only item values you can change without deleting the entry. To locate a particular value of a non-key, non-search, non-sort item, you generally need to perform a serial read.

SORT Items

For any path, it is possible to designate some data item other than the search item as a sort item. If a sort item is specified, each of the chains of the path are maintained in ascending sorted order, based on the value of the sort item. Different paths can have different sort items, and one path's sort item can be another path's search item. Any scalar data item type can be designated as a sort item. Array items cannot be sort items nor strings longer than 256 bytes. CODE items are sorted on the code number, not on the code value (identifier); therefore, CODE item values should be specified in the SCHEMA in the desired sort order.

For example, chains in the BOOK data set composed of entries with the same AUTHOR are maintained in sorted order by TITLE. When a list of all the books by a single author is required, the information is automatically retrieved sorted by TITLE. As with any sorting process, for the list to be meaningful, titles beginning with articles, such as 'The' and 'A', should be altered; otherwise 'The Art of...' is retrieved after 'Data Base...'.

If the chain has a sort item defined, the new entries are either added to the end of, or inserted in, the chain according to the value of the sort item. For string (X) items, the sort is performed using the LEXICAL ORDER defined for the data base, not the current LEXICAL ORDER in the program's environment. In HP-UX defined data bases, the LEXICAL ORDER is always ASCII. Data bases created in BASIC with other than ASCII ordering may cause unexpected results in HP-UX Systems.

If the sort item matches the sort item values already in the chain, the entry is added to the chain after the last matching entry.

Note

The use of sort items should be limited to short chains or data sets which are infrequently updated. The process of inserting an entry in a chain usually requires a serial read of (perhaps all) existing entries in the chain. Although this process is transparent to the program, it can be extremely time consuming and I/O-intensive for a long chain.

Storing a Data Set

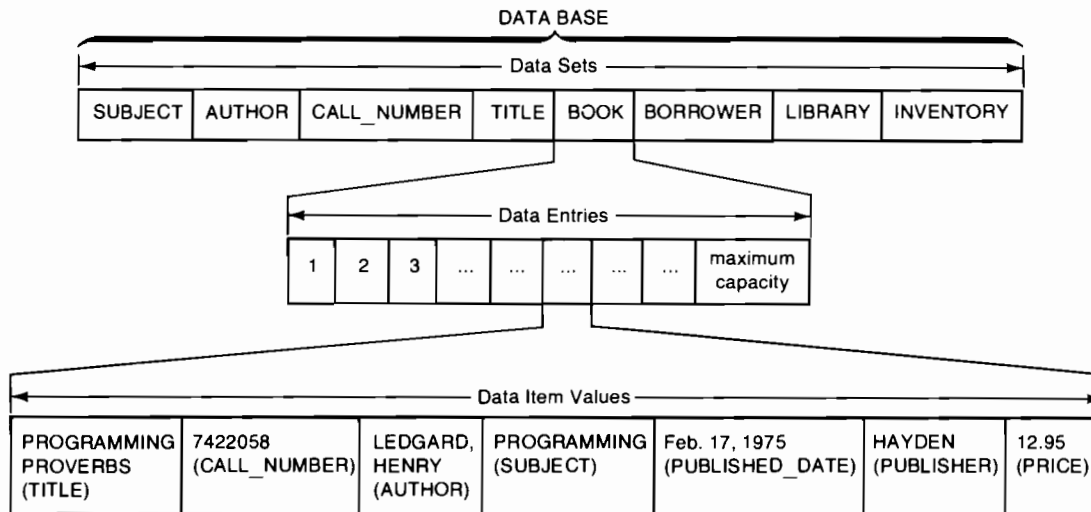
All the entries in a data set are stored together in a separate **data set file** on the mass storage media. Each entry is stored in the file in a unique record known as a logical record or **media record**. A media record contains both the data entry data and linkage information. Its record number is based on its position in the file. The records are numbered from one to the actual capacity of the set.

The portion of the media records that contains the linkage information is called the **header**. Media records always contain the header information but may not contain a data entry. Records with no data entry are called **empty records**.

BASIC distinguishes file types. Data set files are shown as type DSET in a directory listing.

Data Base

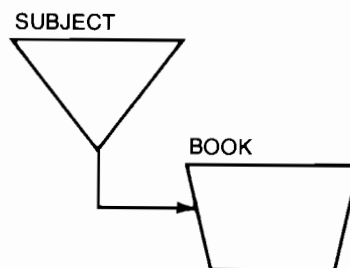
An IMAGE data base consists of one or more data sets, or files, which have some logical relationship with one another. The following diagram demonstrates the structural organization of an IMAGE data base.



Types of Data Sets

There are two types of data sets defined by IMAGE: master data sets and detail data sets. Detail data sets often contain the bulk of the information. They generally contain data items that describe an entity or relationship between entities. A master data set is a collection of key item values and is generally used as an index for fast access to the information in one or more detail sets. Each master entry contains information, transparent to your program, about related detail chains, including the beginning, end and number of entries in the chain.

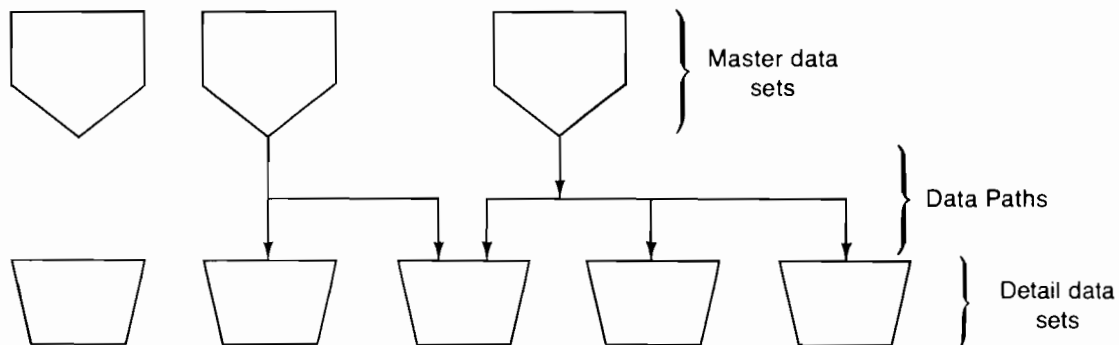
For example, the **BOOK** detail data set contains information about a book, such as title and subject. A related master set, **SUBJECT**, is used to find all the books on a particular subject.



Data Path

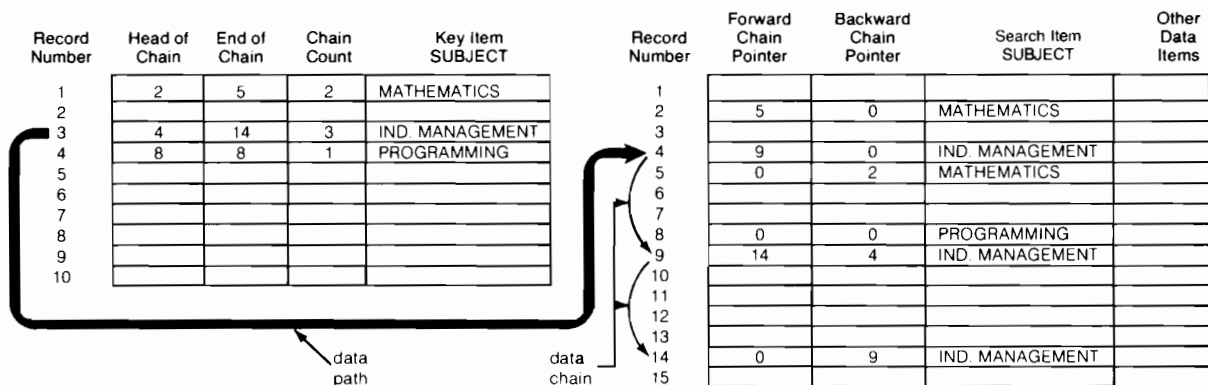
The connection between a master data set and a detail data set via a key item is known as a **data path**. A detail data set can contain up to 31 search items; each search item defines a path to a matching key item in the master data set. A master data set can have paths to 31 detail data sets.

The diagram below illustrates how master and detail data sets can be linked together by a variety of paths.



A data path is the **collection** of links between a key item in a master and a search item in related detail set. There is one link for each value of the key item. A data chain is the link for a particular value of the key item.

The following example illustrates the difference between paths and chains. The path between SUBJECT and BOOK facilitates retrieval of the first entry in a chain of entries having the same subject. The chain links those entries together.



Types of Master Sets

There are two types of master sets: automatic and manual. An automatic master data set contains one item, the key item. A data entry in an automatic master set is made automatically by IMAGE whenever you add an entry with a new search item value to a related detail data set. Similarly, when you delete the last detail entry containing a particular search item value, IMAGE automatically deletes the entry in the automatic master containing that key item.

You cannot explicitly add or delete entries in an automatic master. You can only indirectly add and delete entries by operations in related detail sets. Thus, an automatic master must have at least one path relating it to a detail set.

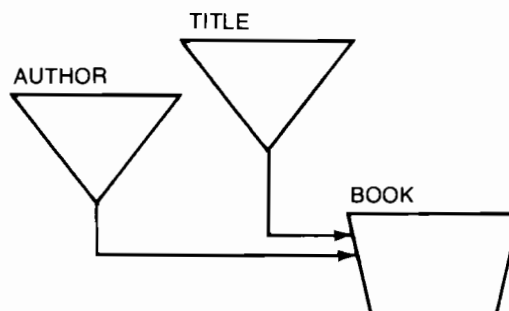
Automatic masters reduce programming effort when updating related detail sets and provide an easy way to establish multiple keyed access to detail data. Automatic masters provide no way to ensure that search items added to detail sets are valid. If you misspell an author's name, IMAGE cannot detect the error and creates a new master entry for the misspelled name.

A manual master data set contains one key item and can contain zero or more non-key items. If a manual master is linked to a detail data set, you cannot add an entry to the detail set if the search item value for that path does not already match a key item value in the master. You must first "manually" add the new key item value to the master set. For example, to add a book with a subject not already in the SUBJECT master, you must first add a new entry to SUBJECT. IMAGE does not automatically add the master entry for you. Similarly, deleting the last book on a given subject from detail set BOOK does not delete that subject's entry in SUBJECT.

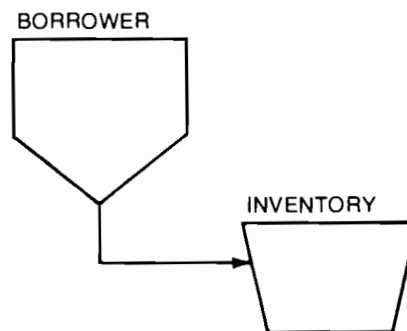
A manual master can contain non-key data items. This is useful in reducing duplication of data. The BORROWER master set contains all of the information about employees and only the EMPLOYEE_NO is duplicated in the INVENTORY detail set. INVENTORY is not littered with redundant BORROWER_NAME, LOCATION and EMPLOYEE_PHONE data. A manual master data set can also stand alone; it need not be linked to any detail data set. This is useful for simple keyed-access file applications.

Master Set Examples

Books in the BOOK detail set are often accessed by title and by AUTHOR. Since it would be time consuming to add, the TITLE and AUTHOR are automatic master data sets.



The BORROWER data set is a manual master to allow information about a borrower to be kept with the EMPLOYEE_NO, rather than being repeated in INVENTORY entries. Because it is a manual master, it also prevents the loaning of books to non-existent borrowers.



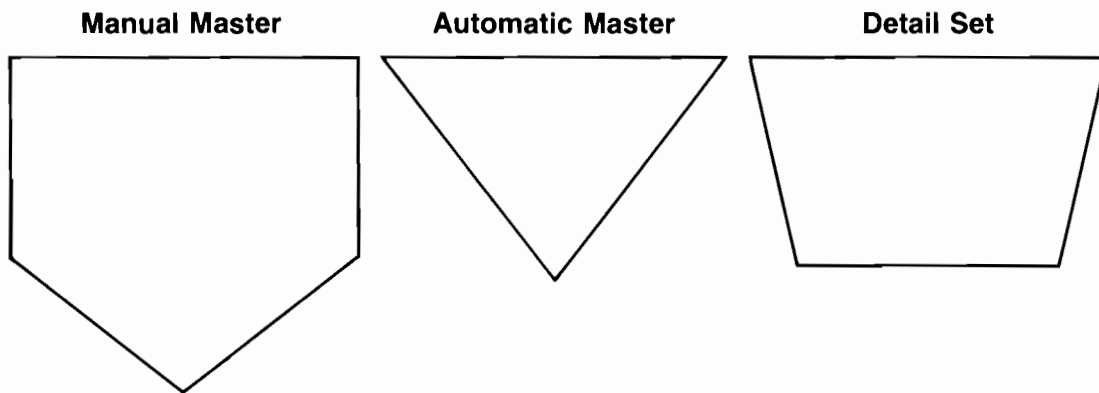
Master Data Set Summary

The following table summarizes the characteristics of manual and automatic master data sets.

Automatic	Manual
Contains one item, the key item	Contains the key item and may contain non-key items
Updating related detail set updates master automatically	Must be updated manually before detail data sets
Contains current key values	Contains all valid key values
Use when manual update is unreasonable and data value checking is unnecessary	Use to check for valid key item and to reduce data repetition by storing details with the key
Must be linked to at least one detail data set .	May stand alone

Diagramming a Data Base

To show the relationship between the various sets in a data base, a data base block diagram is used. Each type of data set is represented by a different shape. Each data path is represented by a line. Here are the three data set shapes:



Storing Data Entries

A data entry is stored in both automatic and manual master data sets in a record number derived by the hashing process described earlier in this chapter. The value of the key item is transformed by the hashing algorithm to generate the record number. This allows entries to be located quickly. An entry is stored in a detail data set in a serial fashion. The next available free record is used. Once occupied by the entry, the record is linked to each search item chain.

If there are no sort items defined for the path, the new entry is linked to the end of the chain. If there is a sort item defined for any search item, the entry is linked into that chain according to the value of the sort item.

Accessing Data Entries

Your environment's file system intrinsics support two traditional methods for retrieving data: serial and random (or direct) access. IMAGE supports four methods: serial, directed, calculated and chained. Each method has different purposes and advantages. The four methods are discussed next.

Serial Access

Serial mode is used with both master and detail data sets. It is the most basic method and is useful when all entries in a set are to be accessed. When accessing in serial mode, IMAGE starts at the record adjacent to the current record and sequentially examines each subsequent (SERIAL FORWARD) or previous (SERIAL BACKWARD) record until the next occupied record is found. This record is made the new the new current record. Serial access may require the reading of numerous empty records. They are skipped automatically.

Serial Access Example

The following example shows entries in the LIBRARY master data set. You can use serial access to find all of the librarians. The arrows to the left show the order in which entries are retrieved in SERIAL FORWARD mode.

Record Number	Key Item PLANT_NAME	LIBRARIAN	Other Data
1			
2	BOISE	SANDY BARLOW	
3			
4	CORVALLIS	BECKY ASHBY	
5			
6			
7	DCD	ANITA NELSON	
8	GSD	STACY LARSEN	
9			
10			
11	SAN DIEGO	JOHN BABCOCK	

You cannot serially read backward beyond the first occupied entry nor forward beyond the last. When the set is first opened for access the current record is zero. From a current record of zero you can serially read forward to the first occupied record or backward to the last. If there is a non-zero current record, you can "rewind" the data set to record zero with a directed read or by closing the data set with DBCLOSE mode 3.

Directed Access

Another method of selecting the master or detail entry is to specify its record number. This is known as a directed read and is similar to random access. If the specified record is empty, an IMAGE status error occurs. Directed access is useful when your program has determined, in some manner, which record to access.

Directed Access Example

The following example shows entries in the BOOK detail data set. The record number of the desired data entry is five.

Record Number	TITLE	AUTHOR	Other Data
1			
2	PROGRAMMING PROVERBS	LEDGARD, HENRY	
3			
4			
→ 5	CHEMICAL PRINCIPLES	MASTERTON, WILLIAM	
6			
7			
8	COMPUTER ARCHITECTURE	FOSTER, CAXTON	
9			

Rewinding a Data Set

A directed read to record 0 has the effect of “rewinding” the data set, allowing a subsequent serial read to access the first or last occupied entry.

Chained Access

Chained access is used primarily to retrieve the entries in a detail data set which have the same value for a given search item and are therefore on the same chain. The related entry in the master set contains pointers to (the record numbers of) the beginning (head) and end of the chain, as well as the number of entries in the chain. Chained access can find either the next or previous entry on the chain.

Chained access can also retrieve master data set entries which are synonyms. This is useful if you have a large number of synonyms and wish to determine which data values cause them.

Chained Access Example

The following example shows how you can use chained access in the INVENTORY detail data set to find all of the books the borrower with EMPLOYEE_NO 9122 has checked out:

Record Number	Forward Chain Pointer	Backward Chain Pointer	Search Item EMPLOYEE_NO	Other Items
1	4	0	9122	
2	6	0	2359	
3	0	0	4105	
4	8	1	9122	
5	10	0	2311	
6	0	2	2359	
7				
8	0	4	9122	
9				
10	0	5	2311	

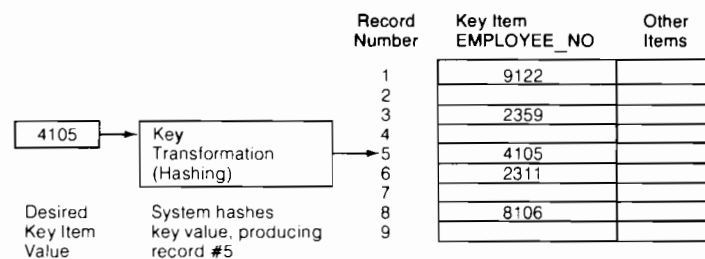
Head of Chain	End of Chain	Chain Count	Key Item EMPLOYEE_NO
1	8	3	9122

Calculated Access

Calculated access is used to access a master data set entry by specifying the key item value you want to access. This method utilizes the same hashing process used when the entry was stored. Calculated access is an efficient method when you know the key item value.

Calculated Access Example

The following example shows retrieval of an entry using calculated access. Calculated access is used to retrieve information from the BORROWER master data set about a borrower with EMPLOYEE_NO = 4105.



Data Base Files

Data Base files can only reside in the HP Structured Directory Format (SDF) file system. On operating systems such as BASIC, which support more than one type of file directory structure, the media on which the data base is created must have been initialized with an SDF directory.

There are three types of files associated with IMAGE/9000 data bases.

Data Base Directory

When the data base is defined, a sub-directory file of type DIR is created for it. The sub-directory name is the data base name specified in DEFINE or SCHEMA.

When the root file is purged with DBPURGE, the directory is also purged if it contains no other files. You may create other types of files in the data base's directory. If these files are related to the data base this may be a reasonable thing to do. In this case DBPURGE is not able to purge the directory after purging the root file.

Data Base Root File

When the data base is defined, a file of type ROOT is created for the data base root file. The file name is "ROOT". The root file is always created in the same sub-directory created for the data base, and cannot be relocated to another directory.

The root file contains all the structural information about the data base.

Because the root file name is "ROOT", and not the data base name, only one data base root file can exist in any data base directory. You can have multiple data bases, each in its own directory.

Data Set Files

When a data set is created, a file of type DSET in BASIC and BDAT in HP-UX is created for it. The file name is the data set name specified in DEFINE or SCHEMA.

Unless otherwise specified in DEFINE or SCHEMA, a data set file is created in the same sub-directory as the root file. A data set file can be created in a different directory, or in a different volume. See “Multiple Volume Data Bases” in the “Advanced Data Base Concepts” chapter.

Note

Although data set files need to be on-line only during access to those sets, the data base root file must not be moved or removed while the data base is open.

Passwords and File Names (BASIC Only)

All three types of data base files are created with the password “IMAGE”. Do not confuse this file password with the set (DBOPEN) passwords specified in DEFINE or SCHEMA.

You do not need to specify this password in data base operations. You may add other passwords to these files, but you must never change or delete the password “IMAGE”, otherwise IMAGE intrinsics and utility programs cannot access the data base.

The names of the data base files must also not be changed, or IMAGE errors result. To assure data base integrity, IMAGE:

- checks the data base directory name against the data base name stored in the root file.
- expects the root file to have the name “ROOT” and be of type ROOT.
- checks each data set name against the set name stored in the root file. Each data set must be of type DSET.

Permission Bits (HP-UX Only)

All three types of data base files have permission bits set to 770. They are owned by the superuser. Do not confuse the file’s permission bits with the set (*dbopen*) passwords specified in the schema. it is recommended that you access and update data base files only through IMAGE-UX library calls.

Shared Data Base Access

If you are using IMAGE in a multi-programming operating system, it is possible for the data base to be open to more than one program at the same time.

If the data base is open in shared read/write mode, you are required to lock the data set or the entire data base during DBUPDATE, DBPUT and DBDELETE. This does not automatically insure that data integrity is maintained, or that unexpected IMAGE errors cannot occur. Consider these two examples:

- Another program is just about to update item Y in the data entry you have just retrieved. You update item X in this entry after the other program's update. When you write the entry back into the data base, you restore item Y to its original value. The data is now invalid. This can happen because you are not required to lock the data set during shared DBGET.
- You store the record number of an entry for rapid retrieval later in your program. While your program is busy doing something else, a second program deletes that entry, and a third adds it back in. You perform a directed read to the entry using the saved record number. You get an "empty entry" status error. If your program even anticipated the possibility of an empty record, it may still erroneously conclude that the record does not exist. In fact the entry does exist, but it is now located in a different record number.

Chapter 3 discusses shared data base access, and how to use DBLOCK and DBUNLOCK to preserve data base integrity.

Data Security

IMAGE can prevent unauthorized access to your data via passwords. You can specify one or more passwords during DEFINE/SCHEMA. For each data set you should specify which passwords have write/read, read-only or no access in that set. A program must supply a valid password to open the data base. For example, in a payroll data base, a mailing list program would typically have read-only access to the employee name and address data set, and no-access to the salary data set.

You can also restrict access to a data base by storing different data sets on separate media (volumes). Access to data sets on mounted volumes does not require that non-accessed sets be mounted.

Corrupt Data Base

A data base is said to be corrupt when data or structural information is lost. If this happens, it may not be possible to access the remaining data. A data base is most commonly corrupted by failing to close it properly. The following actions can corrupt an open data base:

- turning off power to your computer;
- removing the disc medium from the drive;
- purging a ROOT or DSET file by means other than DBPURGE.

Your environment may perform some “clean up” actions for you on program termination. For example, the BASIC Language System closes any data bases open to a program that executes a STOP statement or is stopped by an operator command. You should not rely on this support because the system may not be able to perform this service in all cases. Your data base programs should always perform a DBCLOSE.

Data Base Backup

As with any file management application, making frequent copies, or back-ups, is prudent. Your environment provides at least one method of copying ROOT and all DSET files to other disc files or to magnetic tape. If QUERY is available on your system, it provides an additional method of backing up your data base.

Data bases infrequently changed need to be backed up only as often as they change. Data bases updated regularly need to be backed up more often. Even with a backup, should your data base become corrupt, you still need to re-enter all the updates since the last backup. So the general rule is to decide how much data re-entry is tolerable and then backup as often as necessary to keep the re-entry below that level.

BASIC and HP-UX Systems

If the data base is used in both the BASIC Language System and the HP-UX System, you should create the data base in the BASIC system.

A multi-volume data base used in both systems must be created in BASIC. A multi-volume (multi-directory) data base created in HP-UX cannot be used in BASIC.

Lexical orders other than ASCII can only be defined in BASIC. If non-ASCII lexical is specified, changes to the data base in HP-UX could affect sorted chains. Extension characters are NOT supported in HP-UX. Extension characters in the data base may not be read correctly by HP-UX. Lexical orders other than ASCII are not part of the Hewlett-Packard corporate standard for IMAGE on HP-UX systems.

CODES, PSEUDONYMS and other item part extensions can be defined in BASIC but are not recommended. HP-UX supports these extensions, but they are not part of the Hewlett-Packard corporate standard the IMAGE on HP-UX systems.

A data base created in BASIC has permission bits set to 001. You should change these bits to 770 for use in HP-UX. You should change the owner to root and the group to other. To use the data base in BASIC, the permission bits should be changed to 777.

When storing the time and date in the data base, you should not use the time and data functions available in BASIC and HP-UX because they have different formats. Instead, use an ASCII string expression. For example, define DATE to be type X20 and store Nov. 1, 1983.

Chapter 2

Creating the Schema and the Data Base

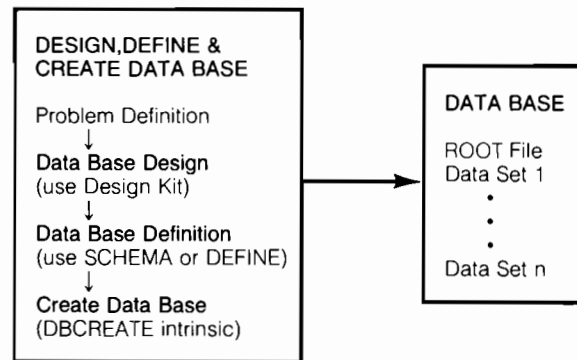


Introduction

After understanding IMAGE concepts and terms presented in Chapter 1, you can design and create your data base.

The design process is covered in the Data Base Design Kit. The creation process is covered in this chapter.

There are three steps in the data base creation process:



1. Translate the data base design into a formal description of the data base structure.
2. Use the SCHEMA program or the DEFINE module of the QUERY¹ program to process the description and create the ROOT file.
3. Create the data sets with QUERY/DEFINE¹ or DBCREATE.

The Schema

The formal description of a data base structure is called a **schema**. The schema is a block of text which defines all of the external aspects of the data base, such as data base, set and item names, passwords, relationships and capacities.

The schema is prepared as a file of text. The SCHEMA utility program reads this file as data. SCHEMA can optionally generate a listing of the schema and information about the data base. If any errors were made, they appear in the listing. Through the use of options you can have SCHEMA simply error-check your schema, or actually create the ROOT file.

If QUERY is available in your environment, and has a DEFINE module, you can also use it to create a data base. DEFINE requires the same data base structure information as SCHEMA, but does not process the information as a data file. DEFINE displays interactive screens at your workstation. If you have prepared your data base design as a formal schema, the information can be more easily entered using these screens. DEFINE is described in the QUERY User's Guide.

¹ QUERY is only available in the BASIC system

Creating a Schema

You can create the schema text file with the standard text or program editing tools available on your computer. You can also generate the file programmatically.

The file can be any type of data file supported in your environment. It can contain any number of lines (records). Each line can have any length, but only the first 160 characters are processed by SCHEMA. Each line can contain a schema statement, schema instruction, comment, or can be entirely blank, for example:

```
schema statement; <<schema comment>>
```

In the BASIC operating system, schemas are normally entered as a “program” of BASIC comments. That is, each line must begin with a line number and the exclamation character. The text should be stored in the schema file with the SAVE or SAVE ASCII statements, and not the STORE or COMPILE statements. In BASIC the example above is entered as:

```
110 !      schema statement; <<schema comment>>
```

Schema Structure

The overall schema and its individual parts are described in the sections following the “Schema Language Conventions” section. Each schema part is defined by one or more syntax drawings of the part and statements, a table of terms and an example. A listing of the complete schema for the sample data base appears at the end of this chapter.

Schema Language Conventions

The parts and elements of a schema are generally free-form. You can place parts, statements, elements and terms on the same line or separate lines, although a single term cannot span a line boundary. The SCHEMA processor interprets the schema file based on the position of reserved words and punctuation.

Case

Reserved words are always uppercase (in both BASIC and HP-UX). Variable terms, such as item names, can contain lowercase letters. Schema does not case-fold lower to uppercase. Item name “BOOK” is not the same as item name “Book”.

Comments

Text within pairs of less-than/greater-than symbols represents a schema comment. When you specify a comment, you must include the << and >> symbols. You can place a comment before, after or within another schema statement, but not within a term of a statement nor within another comment. Comments may take more than one line. The << and matching >> symbols must be within columns 1 through 160.

Readability

You can use leading, embedded and trailing blanks to indent schema statements. The blanks are ignored. You can use entirely blank or zero-length lines to separate schema parts. SCHEMA ignores blank and empty lines. Blanks cannot appear within terms.

Syntax Drawings

Schema part and statement syntax is represented pictorially. All characters enclosed by an oval depict a reserved word and must be entered exactly as shown. Phrases and words enclosed by a box represent elements or terms of a schema statement. A description of each element or term is given in a subsequent drawing, the table or in the Glossary.

Lines between elements can be followed in only one direction, as indicated by the arrows at the end of the line. Any legal path through the diagram represents a syntactically correct schema, part or statement. An element is optional if there is a legal path around it. Optional items usually have a default value when you omit the element. Consult the table following the syntax diagram.

Tables

The table of syntax elements define each element in terms of its syntax. The semantic description of each element (what it means or does) is in the Semantics section following the item tables.

Restrictions

The schema text file can contain a maximum of 32 767 lines. The number of component parts (items, sets, or passwords) is limited. In general you are limited by the available storage on your disc media. The following sections on schema syntax note some of the more restrictive limitations. For a complete summary of data base bounds, see Appendix I, "IMAGE Data Base Compatibility".

Reserved Words

You cannot use the following reserved words as a data base, data set or data item name or pseudonym.

BASE, BEGIN, CAPACITY, CODE, DATA, DIMENSION, DIRECTORIES, END, ENTRY, ITEMS, LEXICAL, NAME, NULL, ORDER, PASSWORDS, PICTURE, PSEUDONYM, RANGE, SETS

You cannot use the following reserved words as data set names.

FORMS, ROOT, SEARCH, SORT, THREAD

Schema Syntax

A schema has five main parts: The **data base part**, **directories part**, **password part**, **item part**, and **set part**. Each part consists of one or more statements in **schema language**. Additional statements are used to express SCHEMA instructions and to designate the end of the schema.

The overall schema structure is:



Example

A complete listing of the sample "LIBRARY" data base schema appears at the end of this chapter.

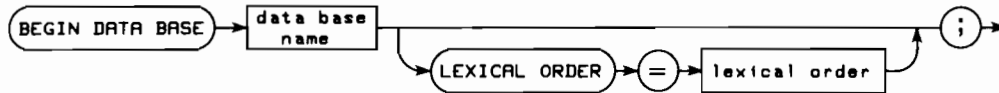
Semantics

The diagram shows only those parts of the schema that are required. There is also a class of schema instructions which control the actions of the SCHEMA program. These instructions can appear on a separate line anywhere before the END. statement except within a statement or comment. They are discussed in this chapter after "Set Part".

Any text or data in the schema file after the END. statement is ignored by the SCHEMA processor.

Data Base Part

The data base part defines the name of the data base and optionally the lexical order.



Element	Description/Default	Range Restrictions
data base name	string constant	IMAGE name
lexical order	string constant; default = ASCII	ASCII, FRENCH, GERMAN, SPANISH, SWEDISH In HP-UX: ASCII only

Example

```

BEGIN DATA BASE LIBRARY                <<Library data base>>
LEXICAL ORDER ASCII;                    <<Explicit default>>

```

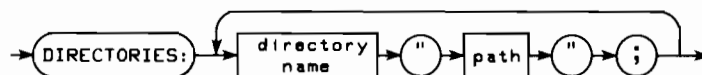
Semantics

To allow access to this data base in both the HP 9000 BASIC Language System and HP-UX operating system environments, restrict the **data base name** to an IMAGE name of 14 characters.

The **lexical order** specifies how a DBPUT operation determines where to insert a sorted data entry in a chain when the sort item is a string item. The DBPUT intrinsic always uses the lexical order defined during data base creation and ignores the current lexical order in your environment.

Directory Part

The directory part is optional. It is only required if one or more data sets of the data base are not located on the same volume (physical media) as the root file.



Element	Description/Default	Range Restrictions
directory name	string constant	BASIC volume label
path	string constant	null string ¹ or path specifier (see path in the Glossary); In HP-UX, less than 160 characters

Example

```

DIRECTORIES:  LIBRARY_1 "First/Second/Third";    <<HP-UX or BASIC>>
               DIR_NAME "/Log_records/Data_sets";  <<HP-UX or BASIC>>
               MY_VOL " "                          <<No Path BASIC>>
  
```

Semantics

This part specifies I/O paths for data sets which are not in the same medium as the ROOT file. Data sets must reside either in the same directory as the ROOT file (no directory name name required) or on a separate medium (both Directory Part and Set Part directory name required).

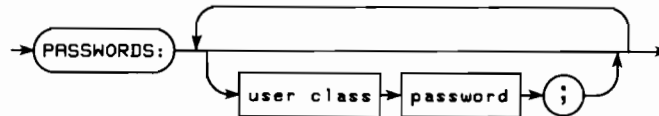
In the BASIC Language System, the directory part identifies volume labels and subdirectory paths. IMAGE locates the physical media by comparing the specified **directory name** to the actual mounted volume labels of all volumes identified by the most recent VOLUME DEVICES ARE statement. The **path** identifies a subdirectory, if any, within that directory in which one or more data sets may reside.

In the HP-UX operating system, the **directory name** identifies a particular path. The **path** identifies either the file path from the HP-UX root directory (has a leading "/"). The BASIC SCHEMA processor ignores a leading "/".

The directory names specified in this part are associated with particular data sets in the Set Part. The directory part does not identify the location of the data base ROOT file. You specify the location of the ROOT file when you run the SCHEMA processor and subsequently when you open the data base with DBOPEN or a utility intrinsic.

Password Part

This part defines levels of read and write access. Each password is assigned a user class. When you define each data set, you specify which user classes have read/write, read-only or no-access capability within the set. If you define no passwords, any password opens the data base and has read/write access in all sets.



Element	Description/Default	Range Restrictions
user class	integer constant	1 thru 13
password	string constant	IMAGE name

Example

```

PASSWORDS:  1      Global_xyzzy;          <<R/W in all sets>>
            2      DBA_Fred;    <<Data base administrator>>
            3      THX1138;    <<Clerk: RALPH T.>>
            7      QUERY;      <<The users>>
  
```

Semantics

If you define any passwords, you must define one with user class 1. You must give this password read/write access to all data sets or SCHEMA generates an error. This password and class are called the “global” password and class. If you define no passwords, you must still specify the “PASSWORD:” term.

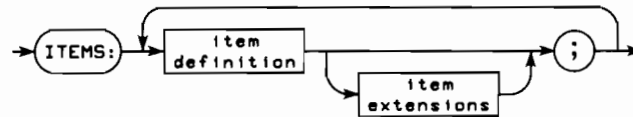
You can specify a maximum of 13 **passwords**.

The capability of the **user class** number is determined during the set part. You specify which user class numbers have what capability. The capability of the user class has nothing to do with the value of the number, except for the global class. Class two is not necessarily more or less capable than class five. It’s up to you to define the capabilities.

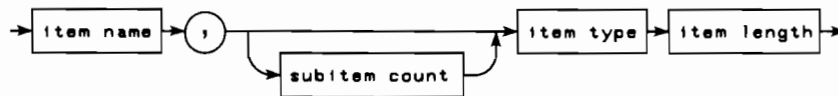
2-8 Creating the Schema and the Data Base

Item Part

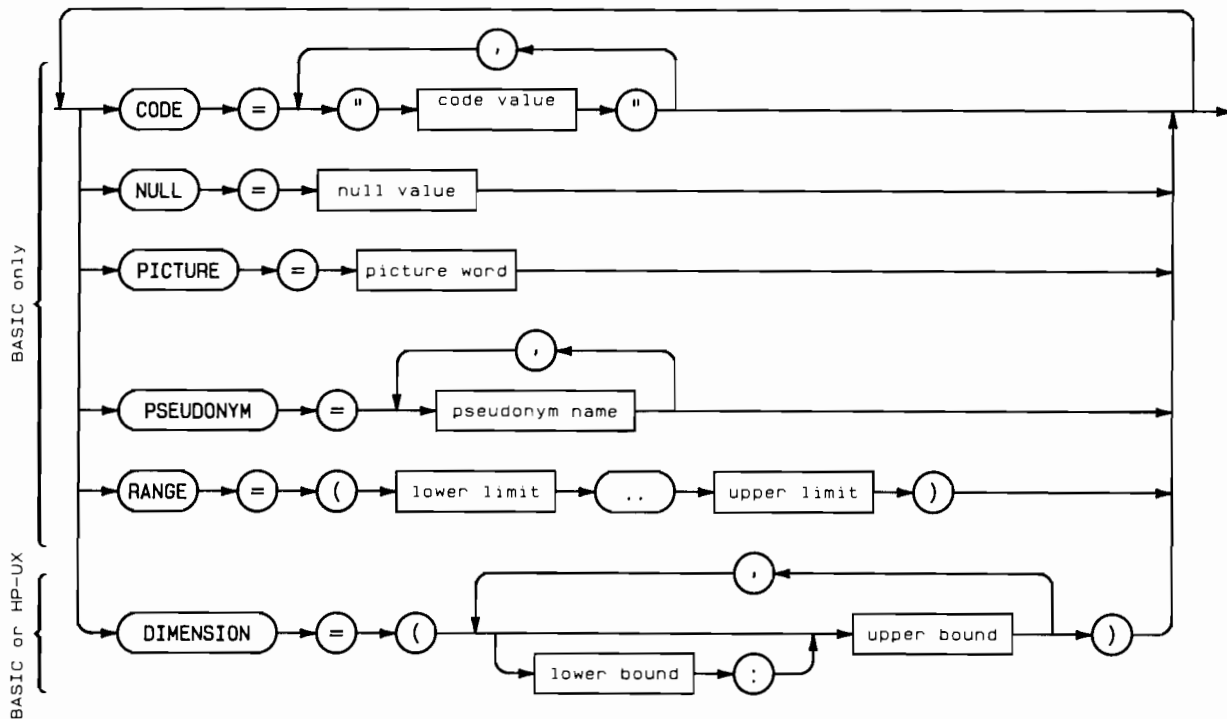
This part defines the name, type, size and any extensions of the data base items. It does not define in which data sets the items reside. That is done in the set part.



item definition:



item extensions:



Element	Description/Default	Range Restrictions										
item definition	(see diagram)	—										
item extensions	(see diagram)	—										
item name	string constant	IMAGE name										
sub-item count	integer constant	range of double integer > 0										
item type	string constant	C ¹ , I, R or X										
item length	integer constant (bytes); default (X only) = 1	<table><tr><th>item type</th><th>length restrictions</th></tr><tr><td>C</td><td>2</td></tr><tr><td>I</td><td>2 or 4</td></tr><tr><td>R</td><td>4 or 8</td></tr><tr><td>X</td><td>range of DOUBLE>0; must be even in HP-UX</td></tr></table>	item type	length restrictions	C	2	I	2 or 4	R	4 or 8	X	range of DOUBLE>0; must be even in HP-UX
item type	length restrictions											
C	2											
I	2 or 4											
R	4 or 8											
X	range of DOUBLE>0; must be even in HP-UX											
code value ¹	string constant, 1 thru 16 characters long	maximum of 32 CODE values per CODE item										
null value ¹	integer constant; default = 0	range of item type										
picture word ¹	integer constant	range of double integer										
pseudonym name ¹	string constant	IMAGE name										
lower limit ¹	numeric constant	range of item type										
upper limit ¹	numeric constant	range of item type ≥ lower limit										
lower bound	integer constant; default = 1	range of double integer										
upper bound	integer constant;	range of double integer ≥ lower bound										

BASIC Example

```

ITEMS:  EMPLOYEE_NO,      I2          <<Employee number>>
        RANGE=(0,,9999);           <<Positive non-zero values only>>
        PLANT_ADDRESS,    X30        <<Facility address>>
        DIMENSION=(1:3)           <<Three lines, name, st, city>>
        PSEUDONYM=ADDRESS,         <<Alias>>
                                PLANT_SITE;
COLORS,      C2                   <<Eight principal colors>>
        CODE="WHITE","RED","YELLOW",
        "GREEN","CYAN","BLUE",
        "MAGENTA","BLACK";
READINGS,    10000I2              <<A data array>>
        DIMENSION=(10,10,10,10)   <<Four dimensional>>
        NULL=-32768;              <<Out-of-range value>>
TIME_STAMP,   R8                   <<Another data array>>
        DIMENSION=(10,10,10,10)   <<Map onto READINGS>>
        PICTURE=-536870912;       <<QUERY "DATE" format>>

```

¹ BASIC only

2-10 Creating the Schema and the Data Base

HP-UX Example

```
ITEMS:  EMPLOYEE_NO,      I2;      <<Employee number>>
        PLANT_ADDRESS,   X30      <<Facility address>>
        DIMENSION=(1:3);  <<Three lines, name, st, city>>
        READINGS,       100I2     <<A data array>>
        DIMENSION=(10,10,10,10); <<DIMENSION overrides the subitem count>>
        TIME_STAMP,      R8;      <<Another data array>>
```

Semantics

You can specify up to 1023 items. Item names cannot contain a comma or semicolon.

The sub-item count specifies that the item is an array. The count specifies the number of elements. An array item cannot also be a key, search or sort item. If you do not also specify a DIMENSION extension, the array is assumed to be one-dimensional with a lower bound of one. For an array, you can specify either a sub-item count, or a DIMENSION extension or both.

You can use the DIMENSION extension to specify an array item of one to six dimensions. For each dimension you can specify both lower and upper bounds. If you also specify a sub-item count, the total number of elements in the DIMENSION expression must agree with the sub-item count in BASIC. In HP-UX the DIMENSION expression is used if you also specify a sub-item count.

You can specify as a code value¹ any 16-character string composed of HP 9000 characters. To specify a quote character in a code value, specify two quotes for each quote in the code value. Code values having less than 16 characters are treated as having trailing blanks. You can specify a CODE extension only for CODE items (C2).

You can specify a null value¹ only for the numeric data types I2, I4, R4 and R8. The null value is used during DBPUT operations in which you omit a value for a data item. If you do not specify a null value in the SCHEMA, IMAGE uses a default value of zero. If you do not have any particular null value already chosen and you are specifying a range, choose a value outside the range.

The default picture word¹ of zero is interpreted by QUERY as an unspecified user-defined PICTURE. See Appendix Q for a description of QUERY data types and their associated picture words.

You can specify one to four pseudonym names¹ for each item name. Leading blanks are ignored. Names less than 16 characters long are treated as having trailing blanks. Pseudonym names cannot contain a comma or semicolon.

You can specify a range¹ only for the numeric types I2, I4, R4 and R8. If you do not specify a range, the SCHEMA processor stores the minimum and maximum possible values for the data type in the root file. For CODE items, SCHEMA always stores a range of 1..maxcode, where maxcode is the number of CODE values you specified.

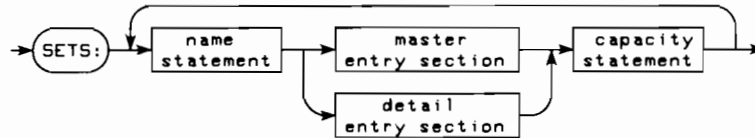
The SCHEMA processor assigns an item number in the range one through 1023 to each item. The numbers are assigned in the order in which you specify the items in the item part.

String items longer than 256 bytes and array items cannot be key, search or sort items.

¹ BASIC only

Set Part

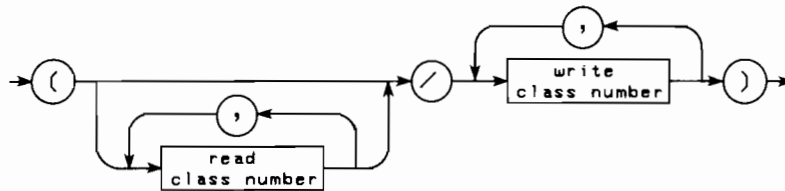
This part defines the name, type, security, data entry structure and capacity of each data set. The data entry structure consists of the item names, paths and sort items, if any.



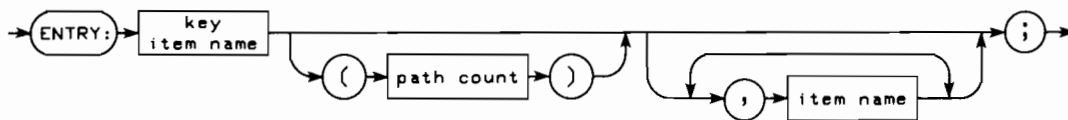
name statement:



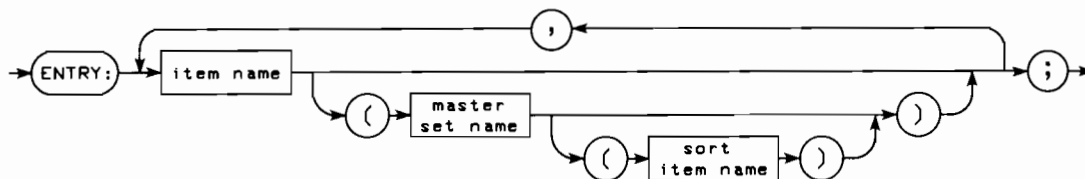
access list:



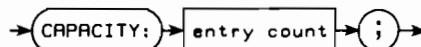
master entry section:



detail entry section:



capacity statement:



2-12 Creating the Schema and the Data Base

Element	Description/Default	Range restrictions
name statement	(see diagram)	—
master entry section	(see diagram)	—
detail entry section	(see diagram)	—
capacity statement	(see diagram)	—
data set name	string constant	IMAGE name
data set type	string constant	A, AUTOMATIC, M, MANUAL, D, DETAIL
access list	(see diagram)	—
volume label	string constant; default = root file's volume	(see DIRECTORY part)
read class number	integer constant	0 thru 13
write class number	integer constant	0 thru 13
item name	string constant	IMAGE name
path count	integer constant; default = 0	1 thru 31
master set	string constant	IMAGE name name
sort item name	string constant	IMAGE name
entry count	integer constant	range of double integer > 0

Examples

SETS:

```

NAME:      Set_name, AUTOMATIC (1/1);    <<set names can be l.c.>>
ENTRY:     Key_item(1);                  <<as can item names>>
CAPACITY:  89;                          <<a Prime number>>

NAME:      LIBRARY, M(2/1);              <<from the sample data base>>
ENTRY:     PLANT_NAME(1),                <<Key item, 1 Path>>
           PLANT_ADDRESS,                <<detail item (array)>>
           LIBRARIAN,                    <<detail item>>
           PHONE_NUMBER;                 <<detail item>>
CAPACITY:  13;                          <<a Prime number>>

NAME:      TYPICAL_DETAIL, D(5,3/1,4,7); <<has 2-search, 3 detail items>>
ENTRY:     Key_item(Set_name(Sorter)),   <<search item, is sorted>>
           Another_Key(Master_3),        <<non-sorted search item>>
           Detail_item,                  <<detail items>>
           Sorter,
           MISC;
CAPACITY:  1000000;                      <<need not be prime>>

NAME:      REMOVED, D(7/1), MY_VOL;      <<a removable data set>>
ENTRY:     Secrets(Xref);
CAPACITY:  32800;

```

Semantics

You can define one through 63 data sets. Each data set can contain any of the maximum of 1023 data items. When the SCHEMA program processes the schema, it assigns a set number in the range one thru 63 to each set. The first set defined is set one.

You must define master sets first, and then detail sets. If you have a schema diagram, you should define each group in the left-to-right order in which they are diagrammed.

If the set is an AUTOMATIC master, use the **master entry** syntax. The ENTRY section must contain only one item, the key item. This item must have a path count of at least one.

If the set is a MANUAL master, use the **master entry** syntax. The ENTRY section must contain one key item and zero or more non-key items. If the set is a “stand alone” master omit the path count for the key item. If the set is linked to any detail sets, the path count must be non-zero.

The **path count** specifies the number of detail data sets in which the key item is a search item.

If the set is a DETAIL data set, use the **detail entry** syntax. The ENTRY section can contain both search and detail items. Each search item must specify the name of the master data set in which an item of the same type is the key item. The first search item specified defines the **primary** (default) **path**. Do not specify a master set name for a non-search item.

You can enable sorting of a search item’s chain by also specifying the name of another item in that entry as the **sort item**. Entries added to that chain are sorted in ascending order according to the value of the sort item. A sort item can be any other item in the entry except a string item over 256 bytes long or an array item. You can specify sorting only for search items, as only search items are chained.

The access list denotes which user classes (zero and those defined in the password part) have read/write, read-only or no-access to the data set. For class numbers to the right of the “/”, the associated passwords have both read and write access to the data set. For class numbers to the left of the “/”, the associated passwords have only read access to the set. If you omit a class number, the associated password has no access to the data set. If you omit the access list or specify zero, any IMAGE name has read/write access to the set.

If you specify an access list, you must specify user class one in the write access list.

If the set is not on the same medium as the ROOT file, you must specify the **directory name** of a path identified in the DIRECTORY part. If the data set resides in the same directory as the ROOT file, you can omit the directory name.

In the ENTRY section, standard practice is to define first the item representing the desired primary path. Then specify the search items in the same order as their master sets, followed by the detail items. The order in which you specify the items in the ENTRY section defines the order of the item fields and offsets in the entry.

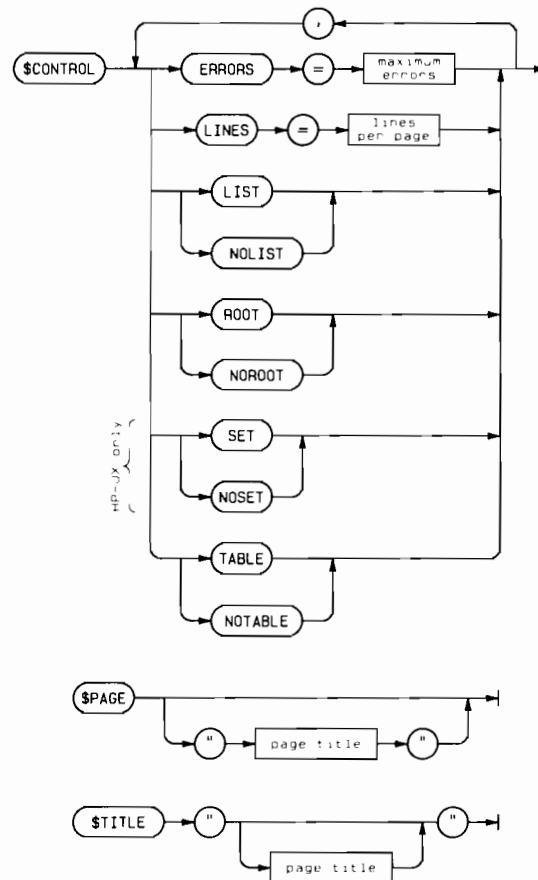
2-14 Creating the Schema and the Data Base

For a master data set, the entry count specified in the CAPACITY statement should be a prime number. If you do not chose a prime number, do not use an even number. If you do not intend to extend the capacity of the master set with DBEXTEND, the entry count should be at least 25% greater than the expected number of master entries. If the set might be extended, the entry count can merely equal or exceed the expected number of entries. There is a table of prime numbers and a listing of a prime-finder program in Appendix T.

For a detail data set, the entry count should equal or exceed the expected number of detail entries. If the number of entries is expected to grow slowly over a long period, you can conserve disc space by specifying a low initial capacity and increasing the size as required with DBEXTEND. See the "Data Base Maintenance" chapter for more information.

Schema Instructions

The three Schema Instructions, \$CONTROL, \$PAGE and \$TITLE specify processing and page formatting options during processing of the schema. You can enter them during creation of the schema. A schema instruction must appear on a separate line and cannot contain a schema comment. The instructions must be separated from their terms by at least one space.



Element	Description/Default	Range restrictions
maximum errors	integer constant; default is ERRORS = 100	0 thru 999
lines per page	integer constant; default is LINES = 62 in BASIC; 53 in HP-UX	20 thru 999
page title	string constant, 0 thru 60 characters long in BASIC; 0 thru 80 characters in HP-UX	—

Examples

```

$CONTROL LIST,ROOT,TABLE
$CONTROL ERRORS=10, LINES=63, TABLE, NOROOT
$PAGE "NOP Company Library Data Base"
$TITLE "Schema for "Data Log" DB"
  
```

2-16 Creating the Schema and the Data Base

Semantics

The **\$CONTROL** instruction affects whether or not your data base ROOT file is actually created and how information from the SCHEMA processor is presented. Some of the terms in the **\$CONTROL** instruction are mutually exclusive, for example, if you specify both **ROOT** and **NOROOT**, the last specified term prevails.

The **defaults** for the **\$CONTROL** terms are:

```
$CONTROL ERRORS=100, LINES=62, LIST, ROOT, TABLE <<BASIC>>
$CONTROL ERRORS=100, LINES=53, LIST, ROOT, TABLE, SET <<HPUX>>
```

The **ERRORS** expression specifies whether or not SCHEMA is to continue processing after one or more errors. If you do not specify an **ERRORS** expression, the default is **ERRORS=100**. That is, SCHEMA ceases processing on detection of the 101th error.

For a very long schema you may want to use a smaller number to conserve paper. If you are listing the schema output to a CRT display, you may want to specify **ERRORS=0** in order to have the process stop with the defective line still visible on the display.

When an error occurs, SCHEMA outputs the defective line and a line identifying the error. If the maximum number of errors have occurred, SCHEMA outputs the message:

```
*** TOO MANY ERRORS. PROCESSOR TERMINATED.
```

The **LINES** expression specifies the number of lines on the page of output. At the end of each page, SCHEMA outputs a form-feed (ASCII FF, code 12) character. If your printer does not support FF, you should specify the actual page length.

When you first run SCHEMA, you enter a destination for its printed output. If you also specify **LIST**, SCHEMA prints each line of the schema file and any error messages on this device. If you specify **NOLIST**, SCHEMA prints only defective lines and error messages.

SCHEMA can process a schema file and check it for errors with or without actually creating the data base root file. If you specify **ROOT**, and there are no errors, SCHEMA creates the ROOT file and prints the following message:

```
ROOT FILE GENERATED.
```

If there are errors, SCHEMA does not create the ROOT file. If you specify **NOROOT**, SCHEMA does not create the ROOT file. **NOROOT** or an error also causes an implied **NOTABLE**.

If you specify **SET**, SCHEMA creates the set files after the ROOT file is created.

SCHEMA can also print useful information about the data sets in the form of a **TABLE** or **ROOT FILE SUMMARY**. You enable the output of this information with the **TABLE** term and disable it with the **NOTABLE** term. This consists of a block of information about each set in the form:

```
- SET NAME: AUTHOR          SET TYPE      : A
FLD CNT : 1                ENTRY LENGTH  : 37  bytes
PATH CNT: 1                MED REC LGTH   : 49  bytes
CAPACITY: 89               PHYS. RECORD(S): 18
LOCATION: Dir/Path: LABEL Vol_name
```

FLD CNT is the number of items (entry fields). For a master set, PATH CNT is the number paths to linked detail sets. For a detail set, PATH CNT is the number of paths from linked masters. LOCATION is the path specifier to the set within the volume denoted by LABEL. The SET TYPE is abbreviated A, M or D. The ENTRY LENGTH is the sum of the lengths of the data items. The MED REC LGTH (media record length) is the sum of the entry length and all the path and chain information.

The final information output by TABLE is the message:

```
DATA ITEM COUNT:  17      DATA SET COUNT:   8
NUMBER OF ERRORS(S):  0
```

You can specify a heading for SCHEMA to output at the top of each page with either the **\$PAGE** or **\$TITLE** instructions. The difference between them is that **\$TITLE** merely defines the heading whereas **\$PAGE** both defines the heading and outputs a form-feed immediately.

You can change the heading to a blank line by specifying a blank line or null string in **\$PAGE** or **\$TITLE**. You can cancel the heading altogether by omitting the page title term in these instructions. In BASIC page titles longer than 60 characters are truncated to 60. In HP-UX page titles longer than 80 characters cause an error.

The BASIC SCHEMA Process

When you have entered your schema and saved it in a text file, you are ready to have the SCHEMA program process it and create the ROOT file. This involves three steps.

1. Load the SCHEMA program.
2. Run the SCHEMA program.
3. Supply the SCHEMA program with information about the schema file, the root file and the printer.

For the BASIC Language System, the SCHEMA program is supplied in source form in a file named "SCHEMA". SCHEMA requires that the following BIN (binary program) option files be loaded in your system:

"IMAGE_DBM" Data Base Management.
"MS" Mass Storage.

If these options are not loaded, you must load them with the BASIC LOADBIN command before performing the remaining steps.

Step 1

The SCHEMA program is supplied in a file of type PROG. After assuring that no valuable program is loaded in your computer, or after creating a partition for SCHEMA, load the SCHEMA program with the BASIC LOAD statement. As an example, if you have stored all of the IMAGE software in a directory (DIR) named "QUERY" on your internal winchester disc, you would use the following command:

```
LOAD "QUERY/SCHEMA:CS80,7" EXECUTE
```

Step 2

Once SCHEMA is loaded, simply press **RUN**. Or, if you prefer, you can substitute the following command for the previous LOAD command:

```
LOAD "QUERY/SCHEMA:CS80,7",1 EXECUTE
```

This command loads the program and runs it from line 1.

Step 3

Once running, SCHEMA initially interacts with you via the CRT display. It requests that you enter four responses.

1. Enter the file specifier of the schema text file. If this file is not in the current MASS STORAGE IS directory, enter the path and media specifier information as part of the file specifier.
2. Enter the media specifier of the ROOT file. Do not include the ROOT file name or any path information.
3. Enter the path specifier of the ROOT file. Do not include the ROOT file name or media specifier.
4. Enter the (numeric) device selector of the printer to which SCHEMA is to send its output. Do not use any device selector functions such as PRT or CRT. SCHEMA's prompt advises you of their numeric equivalents in any case.

SCHEMA uses the EDIT...\$ statement and displays default responses. Because SCHEMA is supplied in source form you may want to change the default responses to streamline multiple executions of SCHEMA.

Creating the ROOT File

If you did not specify NOROOT and no errors occur, SCHEMA creates the ROOT file directory and ROOT file after processing the schema text. Note that errors in schema instructions do not prevent creation of the ROOT file. Once the ROOT file is created, you can create the data sets with the DBCREATE statement.

If another file of the same name, such as an older version of the data base directory, already exists in the same directory needed for the data base, SCHEMA outputs an error message and terminates without creating the ROOT file directory. You must rename such a file or purge the old data base before running SCHEMA.

Programming Hint

For best results specify \$CONTROL...NOROOT until the schema has been processed at least once without error.

SCHEMA Errors

See Appendix E, "Errors", for a summary of SCHEMA processor errors.

BASIC Example

Here is the output from the BASIC SCHEMA processor using the schema of the sample data base:

```

Date: 29 Sep 1982
Schema text file:      DATABASE/LIBRARY_SCHEMA
Location of root file: DATABASE:CS80,7
-----
1      $CONTROL LIST,ROOT,TABLE
2
3      $TITLE  "NOP Company Library Data Base"
4
5      BEGIN DATA BASE      LIBRARY; <<sample data base>>
6
7      PASSWORDS:
8          1      LIBRMGR;
9          2      ENGINEER;
10
11     ITEMS:
12         AUTHOR,
13             X25
14             PICTURE=-536870912 <<Name format>>
15             PSEUDONYM=NOVELIST,WRITER,POET,AUTHOR_NAME;
16         BORROW_DATE,
17             R8
18             PICTURE=-536870912 <<Date format>>
19             RANGE=(208657814400,,211813488000)
20             <<1 JAN 1900 .. 1 JAN 2000>>
21             NULL=148699584000 <<1 JAN 0>>
22             PSEUDONYM=DATE_BORROWED;
23         BORROWER_NAME,
24             X25
25             PICTURE=-536870912 <<Name format>>
26             PSEUDONYM=BORROWER;
27         CALL_NUMBER,
28             I4
29             PICTURE=-276824064 <<M9D format>>
30             RANGE=(0,,999999999);
31         COPY_NUMBER,
32             I2
33             PICTURE=-269484032 <<M2D format>>

```

2-20 Creating the Schema and the Data Base

```
34      RANGE=(0.,20)
35      PSEUDONYM=COPY,COPY_NO,NUMBER_OF_COPY;
36      EMPLOYEE_NO,
37      I4
38      PICTURE=-276824064 <<M9D format>>
39      RANGE=(0.,999999999)
40      PSEUDONYM=EMPLOYEE;
41      EMPLOYEE_PHONE,
42      X14
43      PSEUDONYM=PHONE_NO;
44      LIBRARIAN,
45      X25
46      PICTURE=-536870912 <<Name format>>
47      PSEUDONYM=HEAD_LIBRARIAN;
48      LOCATION,
49      C2
50      CODE="MANAGEMENT", "MARKETING", "PRODUCTION",
51      "R&D", "FINANCE", "PERSONNEL", "QA"
52      PSEUDONYM=AREA_LOCATED, DEPARTMENT;
53      PHONE_NUMBER,
54      X14
55      PSEUDONYM=PHONE, TELEPHONE, TELEPHONE_NO,
56      LIBRARY_PHONE;
```

PAGE 1

NOP Company Library Data Base

```
57      PLANT_ADDRESS,
58      X30
59      DIMENSION=(1:3)
60      PSEUDONYM=PLANT_LOCATION, ADDRESS;
61      PLANT_NAME,
62      X25
63      PSEUDONYM=LIBRARY_BRANCH, PLANT_LIBRARY;
64      PRICE,
65      R4
66      PICTURE=-270663680 <<M3D,2D format>>
67      RANGE=(0.,500)
68      NULL=-1
69      PSEUDONYM=COST, BOOK_PRICE, PRICE_OF_BOOK,
70      COST_OF_BOOK;
71      PUBLISHED_DATE,
72      R8
73      PICTURE=-536870912 <<Date format>>
74      RANGE=(208657814400..211813488000)
75      <<1 JAN 1900 .. 1 JAN 2000>>
76      NULL=148699584000 <<1 JAN 0>>
77      PSEUDONYM=PUBLISH_DATE, DATE_PUBLISHED;
78      PUBLISHER,
79      X30
80      PSEUDONYM=PUBLISHER_NAME;
81      SUBJECT,
82      X30
83      PSEUDONYM=TOPIC;
84      TITLE,
85      X50
86      PSEUDONYM=HEADING, NAME_OF_BOOK;
87
88      SETS:
89      NAME:          AUTHOR,A(/1);
90      ENTRY:         AUTHOR(1);
91      CAPACITY:      89;
92
93      NAME:          CALL_NUMBER,A(/1);
94      ENTRY:         CALL_NUMBER(2);
95      CAPACITY:      89;
96
97      NAME:          SUBJECT,M(/1,2);
98      ENTRY:         SUBJECT(1);
99      CAPACITY:      53;
```

```

100
101     NAME:      TITLE,A(1);
102     ENTRY:     TITLE(1);
103     CAPACITY:  89;
104
105     NAME:      LIBRARY,M(2/1);
106     ENTRY:     PLANT_NAME(1),
107                PLANT_ADDRESS,
108                LIBRARIAN,
109                PHONE_NUMBER;
110     CAPACITY:  13;
111
112     NAME:      BORROWER,M(2/1);
113     ENTRY:     EMPLOYEE_NO(1),

```

PAGE 2

NOP Company Library Data Base

```

114                BORROWER_NAME,
115                LOCATION,
116                EMPLOYEE_PHONE;
117     CAPACITY:  79;
118
119     NAME:      BOOK,D(1,2);
120     ENTRY:     TITLE(TITLE),
121                CALL_NUMBER(CALL_NUMBER),
122                AUTHOR(AUTHOR(TITLE)),
123                SUBJECT(SUBJECT),
124                PUBLISHED_DATE,
125                PUBLISHER,
126                PRICE;
127     CAPACITY:  89;
128
129     NAME:      INVENTORY,D(2/1);
130     ENTRY:     CALL_NUMBER(CALL_NUMBER),
131                COPY_NUMBER,
132                PLANT_NAME(LIBRARY),
133                EMPLOYEE_NO(BORROWER),
134                BORROW_DATE;
135     CAPACITY:  193;
136
137     END.

```

ROOT FILE HAS BEEN GENERATED.

ROOT FILE SUMMARY :

```

=====
- SET NAME: AUTHOR          SET TYPE      : A
  FLD CNT : 1              ENTRY LENGTH  : 37  bytes
  PATH CNT: 1              MED REC LGTH  : 49  bytes
  CAPACITY: 89             PHYS. RECORD(S): 18
  LOCATION: DATABASE:LABEL HP97093A

- SET NAME: CALL_NUMBER     SET TYPE      : A
  FLD CNT : 1              ENTRY LENGTH  : 28  bytes
  PATH CNT: 2              MED REC LGTH  : 40  bytes
  CAPACITY: 89             PHYS. RECORD(S): 14
  LOCATION: DATABASE:LABEL HP97093A

- SET NAME: SUBJECT         SET TYPE      : M
  FLD CNT : 1              ENTRY LENGTH  : 42  bytes
  PATH CNT: 1              MED REC LGTH  : 54  bytes
  CAPACITY: 53             PHYS. RECORD(S): 12
  LOCATION: DATABASE:LABEL HP97093A

```

2-22 Creating the Schema and the Data Base

```
- SET NAME: TITLE          SET TYPE      : A
  FLD CNT : 1              ENTRY LENGTH  : 62  bytes
  PATH CNT: 1              MED REC LGTH   : 74  bytes
  CAPACITY: 89             PHYS. RECORD(S): 26
  LOCATION: DATABASE:LABEL HP97093A

- SET NAME: LIBRARY        SET TYPE      : M
  FLD CNT : 4              ENTRY LENGTH  : 166  bytes
  PATH CNT: 1              MED REC LGTH   : 178  bytes
  CAPACITY: 13             PHYS. RECORD(S): 10
  LOCATION: DATABASE:LABEL HP97093A

- SET NAME: BORROWER       SET TYPE      : M
  FLD CNT : 4              ENTRY LENGTH  : 57  bytes
  PATH CNT: 1              MED REC LGTH   : 69  bytes
  CAPACITY: 79             PHYS. RECORD(S): 22
  LOCATION: DATABASE:LABEL HP97093A

- SET NAME: BOOK           SET TYPE      : D
  FLD CNT : 7              ENTRY LENGTH  : 183  bytes
  PATH CNT: 4              MED REC LGTH   : 183  bytes
  CAPACITY: 89             PHYS. RECORD(S): 64
  LOCATION: DATABASE:LABEL HP97093A

- SET NAME: INVENTORY      SET TYPE      : D
  FLD CNT : 5              ENTRY LENGTH  : 67  bytes
  PATH CNT: 3              MED REC LGTH   : 67  bytes
  CAPACITY: 193            PHYS. RECORD(S): 51
  LOCATION: DATABASE:LABEL HP97093A
```

```
DATA ITEM COUNT: 17      DATA SET COUNT: 8
NUMBER OF ERROR(S): 0
```

The HP-UX Schema Process

For the HP-UX system, the schema program is supplied in the `/usr/bin` directory of IMAGE-UX.

When you have entered your schema and saved it in a file, you are ready to have the schema program process it and create the ROOT file. To do this, you execute the *schema* command and optionally specify the schema text file, the path on which the ROOT file is created and an output file.

For example,

```
schema -f skema -P /user/mine/database >mylisting
```

Schema uses stdin file if you do not specify a schema file. The ROOT file is created in the current directory unless you specify a path (`-p`). If you asked for a listing of the schema, *schema* uses the stdout file unless you redirect the output.

Creating the ROOT file

Schema creates the ROOT file directory and the ROOT file after processing the schema file if you did not specify **NOROOT** and no errors occurred.

If another file of the same name, such as an older version of the data base directory, already exists in the same directory needed for the data base, *schema* outputs an error message and terminates without creating the ROOT file directory. You must rename such a file or purge the old data base before running *schema*.

Programming Hint

For best results specify `$CONTROL...NOROOT` until the schema has been processed without error.

Creating the Data Sets

Schema creates the data sets after creating the ROOT file if you did not specify **NOSET**.

Example

Here is the output from the HP-UX schema processor using the schema of the sample data base:

```
BEGIN DATA BASE LIBRARY; << single volume data base >>
PASSWORDS:
  1  LIBRMGR;
  2  ENGINEER;
ITEMS:
  Author,           X26;<< item length has to be even >>
  Borrow_date,      X10;<<No PICTURE extension;so date is a string>>
  Borrower_name,    X26;
  Call_number,      I4;
  Copy_number,      I2;
  Employee_no,      I4;
  Employee_phone,    X14;
  Librarian,        X26;
  Location,         X26;
  Phone_number,     X14;
  Plant_address,    X30
  DIMENSION =      (1:3);
  Plant_name,       X26;
```

2-24 Creating the Schema and the Data Base

```
Price,           R4;
Published_date,  R8;
Publisher,       X30;
Subject,         X30;
Title,          X50;

SETS:

  NAME:          AUTHOR, A(/1);
  ENTRY:         Author(1);
  CAPACITY:      89;

  NAME:          CALL_NUMBER, A(/1);
  ENTRY:         Call_number(2);
  CAPACITY:      89;

  NAME:          SUBJECT, A(/1,2);
  ENTRY:         Subject(1);
  CAPACITY:      53;

  NAME:          TITLE, A(/1);
  ENTRY:         Title(1);
  CAPACITY:      89;

  NAME:          LIBRARY, M(2/1);
  ENTRY:         Plant_name(1),
  Plant_address,
  Librarian,
  Phone_number;
  CAPACITY:      13;

  NAME:          BORROWER, M(2/1);
  ENTRY:         Employee_no(1),
  Borrower_name,
  Location,
  Employee_phone;
  CAPACITY:      79;

  NAME:          BOOK, D(/1,2);
  ENTRY:         Title(TITLE),
  Call_number(CALL_NUMBER),
  Author(AUTHOR (Title)),
  Subject(SUBJECT),
  Published_date,
  Publisher,
  Price;
  CAPACITY:      89;

  NAME:          INVENTORY, D(2/1);
  ENTRY:         Call_number(CALL_NUMBER),
  Copy_number,
  Plant_name(LIBRARY),
  Employee_no,
  Borrow_date;
  CAPACITY:      193;

END,

*** ROOT FILE GENERATED
*** DATABASE CREATED
*** ROOT FILE SUMMARY

- SET NAME: AUTHOR          SETTYPE      : A
  FLD CNT : 1              ENTRY LEN   : 38 bytes
  PATH CNT: 1              MED REC LEN  : 50 bytes
  CAPACITY: 89             PHYS. RECORDS : 1
  LOCATION: LIBRARY

- SET NAME: CALL_NUMBER     SETTYPE      : A
  FLD CNT : 1              ENTRY LEN   : 28 bytes
  PATH CNT: 2              MED REC LEN  : 40 bytes
  CAPACITY: 89             PHYS. RECORDS : 1
  LOCATION: LIBRARY

- SET NAME: SUBJECT         SETTYPE      : A
  FLD CNT : 1              ENTRY LEN   : 42 bytes
  PATH CNT: 1              MED REC LEN  : 54 bytes
```

CAPACITY: 53	PHYS. RECORDS : 1
LOCATION: LIBRARY	
- SET NAME: TITLE	SETTYPE : A
FLD CNT : 1	ENTRY LEN : 62 bytes
PATH CNT: 1	MED REC LEN : 74 bytes
CAPACITY: 89	PHYS. RECORDS : 1
LOCATION: LIBRARY	
- SET NAME: LIBRARY	SETTYPE : M
FLD CNT : 4	ENTRY LEN : 138 bytes
PATH CNT: 1	MED REC LEN : 150 bytes
CAPACITY: 13	PHYS. RECORDS : 1
LOCATION: LIBRARY	
- SET NAME: BORROWER	SETTYPE : M
FLD CNT : 4	ENTRY LEN : 82 bytes
PATH CNT: 0	MED REC LEN : 94 bytes
CAPACITY: 79	PHYS. RECORDS : 1
LOCATION: LIBRARY	
- SET NAME: BOOK	SETTYPE : D
FLD CNT : 7	ENTRY LEN : 184 bytes
PATH CNT: 4	MED REC LEN : 184 bytes
CAPACITY: 89	PHYS. RECORDS : 1
LOCATION: LIBRARY	
- SET NAME: INVENTORY	SETTYPE : D
FLD CNT : 5	ENTRY LEN : 62 bytes
PATH CNT: 2	MED REC LEN : 62 bytes
CAPACITY: 193	PHYS. RECORDS : 1
LOCATION: LIBRARY	

Creating the Data Set Files

After creating the ROOT file with SCHEMA, you must create the data set files before you can access the data base. You create data set files with the DBCREATE intrinsic. This intrinsic can create a single set, a combination of sets or all the sets.

DBCREATE creates the sets in the same directory as the ROOT file unless you specified a volume label for that set in the schema SET part. These directories are located according to the path specifiers in the DIRECTORY part.

Refer to the BASIC Language or HP-UX Reference manual for the syntax details of the DBCREATE intrinsic.

2-26 Creating the Schema and the Data Base

(This page intentionally blank)

Chapter 3

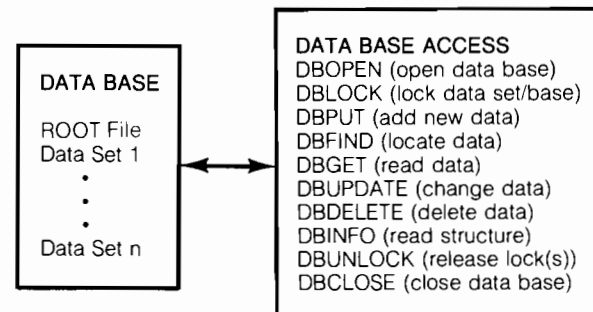
Data Base Access



Introduction

After you have designed and created your data base, you are ready to begin accessing it. There are ten statements which provide different ways to access the data in a data base.

DBOPEN and **DBCLOSE** are necessary for all data base operations. **DBOPEN** opens the data base according to the read/write security level associated with the password you supply. You must use **DBCLOSE** to terminate data base access at the end of your program. You can also use partial **DBCLOSE** during your program to ensure that all updates are written (posted) to the data base media, close individual data sets or release unneeded resources.



DBPUT adds an entry to a manual master or detail data set.

DBFIND locates the head of the detail data set chain linked to the specified key item value. You can use **DBFIND** to setup subsequent chained **DBGET**'s in the detail set.

DBGET retrieves an entry from the specified master or detail set.

DBUPDATE changes the value of items (non-key, non-search, non-sort) in a manual master or detail data set.

DBDELETE deletes a data entry from a manual master or detail data set.

DBLOCK and **DBUNLOCK** provide your program with exclusive access to part or all of a data base which may be open to other programs or users. You use **DBLOCK/DBUNLOCK** to ensure that no other program changes the data base between your operations.

DBINFO provides information about the data base. Your program can use **DBINFO** to obtain much of the information you specified in the schema.

Except for **DBBUFFER**¹ and **DBCONTROL**, these are the data base **management** intrinsics. There is also a group of data base **utility** intrinsics. **DBBUFFER**, **DBCONTROL** and the utility intrinsics are discussed in later chapters.

¹ BASIC only

Host Language Considerations

This section discusses considerations common to all of the programming languages which you can use to access a data base.

For considerations unique to each language, refer to the appropriate appendix:

Appendix B: BASIC Language Considerations

Appendix C: C Language Considerations

Appendix F: Fortran Language Considerations

Appendix P: Pascal Language Considerations

Because of the differences between languages, this and subsequent chapters do not include many examples of actual program code. Each topic's examples appear instead in the language's appendix.

This chapter also does not define precise syntax for each intrinsic. Syntax is defined in the Reference manual for your environment. You should have it available as you read this chapter.

Parameter Data Types

The terms and expressions specified in data base intrinsics generally must be, or evaluate to, either numeric or character (string) data types. In the BASIC Language System, the syntaxing performed during EDIT prevents most errors and the run-time compiler can perform automatic numeric type conversions for you. In HP-UX the editor performs no syntax checks and a batch compiler like FORTRAN cannot determine if the form of your CALL correctly matches the requirements of the intrinsic. You must, therefore, be more rigorous about specifying precisely the correct parameters and data types.

The Data Base Variable

When you first open the data base with DBOPEN you specify the location of the data base, and in BASIC any file passwords necessary to access the directory containing the data base directory. You must specify this information in the form of a string variable and not a string expression. Also, the variable must have exactly two leading blank (ASCII code 32) characters.

You should use this same variable in subsequent data base operations in your program. When the data base is opened, the variable is changed. If your program is highly structured, segmented or overlayed, you need to consider how to pass this variable to all the parts of your program that invoke data base intrinsics. For example, you can pass it as a subprogram parameter or place it in a COMMON area.

Refer to the section on DBOPEN for further discussion of the data base variable.

The Status Array

Unlike most I/O or file system operations in your programming language, very few errors that occur during data base intrinsic operations result in a system error "trap". Your BASIC program does not pause and display an error message. Your FORTRAN program does not abort. Instead, whether or not the operation fails, IMAGE stores information about the operation in an array and your program continues running.

The array is the **status array**. After each intrinsic completes, it contains a pass/fail word (the **condition word**) and useful information about the current entry. In the case of failure, it contains further information about the cause of the error. Not all status errors indicate a "problem" with the data base. For example, a simple way to find the end of a chain is to perform a chained read until an "end of chain" status error occurs.

In BASIC the status array is a DOUBLE array of at least ten elements. Unlike many other parameters, it must be of type DOUBLE. If it is not, or has fewer than ten elements, your program pauses and displays an error.

In the HP-UX operating system, IMAGE assumes that the status array represents 20 bytes of contiguous storage. If it does not, other program variables may be altered when IMAGE updates this block of memory.

You should examine the status array after every data base operation and check the condition word and correct the problem if a status error is detected.

Opening the Data Base - DBOPEN

To access a data base with the data base **management** intrinsics you must first open the data base with DBOPEN. The **utility** intrinsics perform their own DBOPEN. There is an example program segment demonstrating DBOPEN in each language Appendix.

Data Base Number

Each time you successfully open a data base, IMAGE assigns an ASCII **data base number** to that particular invocation of DBOPEN. This data base number is a pair of ASCII characters in the range "00" through "99". A new number is allocated even if you already have the same data base open. Each time you successfully close a data base, the number is released. The maximum number of DBOPENs that can be in effect at one time is 100.

This is the only arbitrary limit on data base access. Within the constraint of 100 DBOPENs, the following are valid combinations of active DBOPENs.

	A single program or user can...	Multiple programs or users can...
...open the same data base once.	YES	YES
...open the same data base more than once.	YES	YES
...open different data bases once.	YES	YES
...open different data bases more than once.	YES	YES

DBOPEN Combinations

DBOPEN stores the data base number in the **data base variable**. All the other management intrinsics use this data base number to identify the data base they are accessing. It is for this reason that you should use the same variable in all intrinsic invocations. If you do not, or if you modify the data base variable, the intrinsics may be unable to perform the requested operation.

In the case where you re-open a data base already open to you, IMAGE still assigns a new available data base number. You must use different data base variables to accomplish this, even though their contents are identical (before DBOPEN). Your accesses using the resulting (different) data base variables are treated just as if you are accessing different data bases. For this reason, the data base is referred to as being open "to the data base variable" rather than "open to you".

Specifying the Data Base

In addition to having the correct form, the data base variable must at least identify the **name** of the data base root file directory.

Path Specifiers

If the data base root file directory is not located in the current MSI (MASS STORAGE IS) directory (BASIC) or in the current working directory (HP-UX), then you must also include a path specifier to the data base root file directory.

Directories

If the data base is on a separate volume not identified by MSI (BASIC), you must either change the MSI or include the volume LABEL in the data base variable. The volume LABEL must be one identified by a VDA (VOLUME DEVICES ARE) statement. In HP-UX, you must mount the volume to a directory in an accessible path. You must also mount the volumes containing any data sets.

If the volume requires a volume password for read access (BASIC only), you must specify this password in the VDA statement or in the media specifier of the data base variable. Volume passwords are not recommended for data bases to be accessed in HP-UX. There is no method for specifying this password in HP-UX.

Access Capability

You must have at least **read** access capability in every directory in the path above the root file directory. In BASIC, you must specify directory passwords. In HP-UX you must have the appropriate permissions for all directories down to the data base directory. IMAGE executes with super user permission and has access to the data base directory. You do not need to specify a password for the root file directory itself or the root file. The IMAGE software provides the correct password ("IMAGE" in BASIC) when it opens these files. These passwords have nothing to do with the DBOPEN password parameter.

DBOPEN Password

The DBOPEN password parameter determines your access capability within each data set. **Read** capability in a data set permits you to perform DBFIND, DBGET and DBINFO operations specifying or involving that set. **Write** capability permits you to perform read operations and DBPUT, DBUPDATE and DBDELETE operations within the set.

You must specify a data base **password** in DBOPEN even if there are no passwords defined for the data base. If there are no passwords defined, any password provides read/write access to all data sets. If there are passwords, the password you specify must match one defined during SCHEMA or DEFINE. If you specify an invalid password, DBOPEN does not open the data base.

Your (valid) password is matched to a user **class number**. This class number determines which data sets you may access. You can read and write in data sets in which it is a write class number. You can only read data in sets in which it is only a read class number. Data sets in which the password denotes neither a read nor write class number are effectively invisible to you. You cannot read, write or even obtain information about such data sets.

Passwords and Automatic Masters

If you are performing write (DBPUT, DBUPDATE, DBDELETE) operations in a detail set and that set is linked to any automatic master data sets, your password need not have write access to those masters.

DBOPEN Mode

The open mode specifies the type of data base access you intend to perform. This mode is independent of whether your password permits you to actually perform the access. There are four open modes, summarized in the following table.

Open Mode	Read/Write access	Exclusive or shared access	DBLOCK/DBUNLOCK requirements	DBCONTROL modes 1 and 2 default
1	Read/Write	shared with other mode 1 users	read: optional write: required	mode 2, immediate posting
3	Read/Write	exclusive	ignored	mode 2
8	Read Only	shared with other mode 8 users	read: optional write: n.a.	not applicable
11 ¹	Read/Write	exclusive	ignored	mode 1, deferred posting

DBOPEN Open Modes

Read vs Read/Write

The first question to consider when selecting a mode is whether or not you need modify the data base or merely read data from it. If you do not need to write into, modify or delete data entries, consider using DBOPEN mode 8.

Sharing of a Data Base

Will any other user or program attempt to access the same data base while you have it open? This question includes whether you might open the same data base more than once in your own program.

For a DBOPEN specifying an open data base to succeed, the first DBOPEN must have specified mode 1 or 8 and the second and subsequent DBOPENs must specify the same mode. Both modes permit reading the data base. Mode 1 permits both reading and writing.

If you do not want another user, including your own program, to open the data base while you have it open, you must open it exclusively. You can use DBOPEN modes 3 or 11 to do this.

If you merely want to prevent other users from modifying the data base, you can prevent this by the use of the DBLOCK intrinsic. In fact, when the data base is open in shared mode, you are required to have the affected data set(s) locked with DBLOCK prior to performing any write operations.

Note

You are not required to lock a shared data base during read operations. If you are modifying the data base by performing a read (DBFIND, DBGET, DBINFO) operation followed by a write operation, another user could modify the data base between your read and write. To avoid this, you should lock the data set(s) prior to the read operations and not just before the write operation.

¹ BASIC only

Buffering and Posting (BASIC Only)

If your data base program has to meet demanding performance criteria, you need to understand and take advantage of disc buffering, or deferred posting.

All of your data, both read and written, is held temporarily in a block of system memory called a buffer. How much is held there affects performance. How long it is held there has implications for data integrity. If you open your data base with DBOPEN mode 3, write operations always result in immediate disc access. If you open the data base in mode 11, your data is held in memory for as long as possible before writing (posting) it to disc. It may be held until you close the data base.

In general, you should use DBOPEN mode 3 instead of mode 11 unless your program's performance is unsatisfactory. Buffering and posting are discussed further in the "Advanced Data Base Operations" chapter.

Closing the Data Base - DBCLOSE

When you have concluded all data base operations you must close the data base. If you fail to close it you may lose data or corrupt your data base.

The principal purpose for the DBCLOSE intrinsic is to close the entire data base. It also has three other modes which perform partial closes, "rewind" and posting operations. The modes are summarized in the following table.

	Close mode 1	Close mode 2	Close mode 3	Close mode 4
portion of data base closed	root and sets if not shared	set, if not shared	none	none
locks on data base or set(s)	released	retained	retained	retained
I/O buffers	all are posted if not shared	set's are posted if not shared	no action	all are posted
Current Record Number	n.a.	retained	set to 0	retained
Current List Specifier	n.a.	retained	lost	retained
Current Chain	n.a.	retained	lost	retained

DBCLOSE Modes

Ordinary Close - Mode 1

This mode closes the entire data base. Further access to the data base using the same data base variable requires that you re-open the data base with DBOPEN. The data base number in the data base variable is changed back to two leading blanks.

DBCLOSE mode 1 does not close the data base to other users. If you opened the data base to more than one data base variable, you can still access the data base via that variable. To close the data base to that variable you must perform a DBCLOSE mode 1 specifying that variable.

If the data base is not open to any other user or variable, DBCLOSE mode 1 causes all outstanding modifications to be written from the system buffers to the data base (posting). It also closes the data set and data base root files. If no other files are open in the root directory, DBCLOSE closes that directory as well. If you have the data base or any data sets locked, DBCLOSE mode 1 performs an implied DBUNLOCK.

There is an example of DBCLOSE mode 1 in each language Appendix.

You should check the condition word after a DBCLOSE to ensure that the statement operated correctly. If a status error occurs during DBCLOSE, the data base is not closed. Possible status errors during DBCLOSE are listed in the reference manual entry covering DBCLOSE. One status error is -1, which means that a system error occurred. You should check the last Status Array element to obtain the system error number.

The reason DBCLOSE leaves the data base open if an error occurs is to give you an opportunity to correct the problem. For example, if a data set volume is off-line, you can display a message requesting that the operator insert the disc in the drive. If the problem is one that you cannot correct you need to perform a forced close.

Forced Close

Your environment can automatically perform a special type of DBCLOSE when your program terminates execution in BASIC. This close is not performed if your program merely executes a BASIC PAUSE statement.

This forced DBCLOSE posts as many system buffers as possible, and closes all data base files. Inaccessible sets may be marked as CORRUPT in the root file if their buffers contain updates which could not be posted.

In the BASIC Language System forced close is performed for all open data bases in the affected partition when you execute any of the following statements: DEL PARTITION, END, GET, LOAD, **RUN**, STOP and **STOP**. A DBCLOSE mode 1 is automatically performed for all open data bases in the computer if you execute any of the following statements and commands: LOADBIN, SCRATCH A, **SHIFT** **RUN** (RUN ALL) and **SHIFT** **STOP** (STOP ALL).

In the HP-UX system, forced close occurs on program termination. If you logout while your data base process is running, IMAGE-UX closes the data base.

The purpose of this automatic DBCLOSE is to try to prevent loss of data due to inadvertent statements and commands. This automatic close simplifies program debugging, but you should not rely on it for normal operations. However, if you execute a DBCLOSE which fails, and you cannot correct the problem, you can close the data base with any of the forced close operations.

Partial Close - Mode 2

For a data base with numerous data sets, a significant amount of system memory can become unavailable for other uses. This is because a system buffer is allocated for I/O each time a data set is accessed for the first time. This buffer remains in existence until you close the entire data base or until you close one or more data sets. The purpose of DBCLOSE mode 2 is to close one or more data sets without closing the entire data base.

If the data base is open only to your data base variable, DBCLOSE mode 2 posts the buffers to the specified sets and closes the set files. If the data base is open to another user or variable, DBCLOSE mode 2 has no effect. If the sets are closed, they are automatically reopened whenever you next access them. You do not need to perform a DBOPEN as long as you have the data base open.

If have limited memory, you should use DBCLOSE mode 2 whenever you are finished using a data set. You can also use mode 2 to cause posting of buffers to their data sets, although mode 4 is a better choice for this operation.

Rewind a Data Set - Mode 3

To perform a SERIAL FORWARD read to the first entry in a data set or a SERIAL BACKWARD read to the last entry, the current record must be zero. The current record is zero until you perform a DBPUT, DBGET or DBDELETE operation in the set, at which time the current record becomes some non-zero value.

If the current record is non-zero, you can reset it to zero in several ways. The two most efficient ways are DBCLOSE mode 3 and DBGET mode 4 DIRECTED read specifying record zero.

DBCLOSE mode 3 does not actually close any portion of the data base, nor does it post any buffers. It resets the current record to zero in the specified set. Unlike DBGET mode 4, DBCLOSE mode 3 has two side effects which may be undesirable:

- In a detail data set, DBCLOSE mode 3 resets the current path to the primary path. This is only significant if you were performing chained DBGETs prior to the DBCLOSE mode 3. To perform a CHAINED DBGET after a DBCLOSE mode 3, you must first perform a DBFIND to re-establish the path.
- When you are performing DBPUT, DBGET, DBUPDATE and DBDELETE operations you have a choice of methods of specifying the list of data items read or written. Once specified, you can specify that you want to use the "current list" mode in subsequent statements. DBCLOSE mode 3 causes IMAGE to clear the current list mode for the re-wound data set. After a DBCLOSE mode 3, you must re-establish the list mode in the next DBPUT, DBGET, DBUPDATE or DBDELETE.

If you need to rewind a data set and want to avoid these side effects, use DBGET mode 4.

Force Buffer Posting - DBCLOSE mode 4

If you are using deferred posting mode (see DBOPEN, DBCONTROL and DBBUFFER), it is wise to occasionally post the buffers to their data sets.

DBCLOSE modes 1 and 2 cause posting of data set buffer data to the data sets if the data base is open only to your data base variable. DBCLOSE mode 3 never posts buffers.

If you want to force posting, regardless of how many users have the data base open, use DBCLOSE mode 4. Mode 4 causes all buffers containing unposted data base modifications to be written to their data sets. It does not close any data sets, nor does it have any side effects. The current record and current list for each data set are retained.

Reading and Writing Data

Once you have opened your data base, you can add data to data entries with the DBPUT intrinsic, read data from them with DBGET, modify data with DBUPDATE and return entries to the empty state with DBDELETE. DBPUT, DBGET and DBUPDATE permit you specify the data items of interest in two forms and specify the program variables in two forms.

The way you specify which data items to read or write is via a parameter called the list specifier. Depending on your environment, the program variables can be either a list of variables or a single variable called a packing variable. The next sections discuss how to choose and use the list specifier and variable list.

Identifying the Data Items - the List Specifier

When you read data from an entry with DBGET, you are not required to retrieve the value of every data item (field) in the entry. You can request that IMAGE return values for one, several or all of the items. Similarly, when you create a new data entry and write items values to it with DBPUT, you can omit item values for which you want IMAGE to supply a null value. Finally, when you change a data entry with DBUPDATE, you can cause items to be left unchanged by omitting them from the DBUPDATE intrinsic. The way in which you identify which items to read or write is denoted by the list specifier.

The Form of the List Specifier

The precise syntax of the list specifier is shown in your reference manual for the DBGET, DBPUT and DBUPDATE intrinsics. In general it is either a string variable, a literal or a string expression. It must evaluate to one of three forms: a @ (commercial at) character, a list of data base item names (or an array of numbers in HP-UX), or an * (asterisk) character.

The parameter passing conventions of your host language may require that you terminate the character or name list with a semicolon. Consult your reference manual or the Language Consideration Appendices for information concerning string parameters. In BASIC, for example, the semicolon is optional. IMAGE ignores any characters in the string beyond the semicolon.

Full Record List Mode (@)

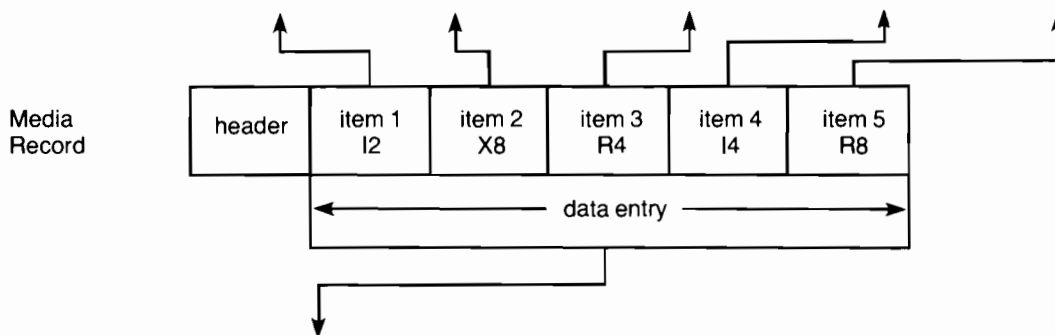
If you want to access every item in the data entry, you can use full-record mode. This mode offers simplicity in program coding and a slight performance increase over item list mode.

Full record mode implies that you are supplying a value for every item in DBPUT and DBUPDATE or that you expect to retrieve a value for every item in DBGET. You can use either a variable¹ or a packing variable with full record mode subject to these constraints:

- The packing variable must have a length in bytes equal to or greater than the length of the data entry (not the media record).
- The list of variables¹ must contain one variable for each data base item in the data entry. Each variable must be the same type as the item. The variables must be in the same order as the items in the entry. You can determine the order from the Set Part of your schema, or via DBINFO requests. This syntax is available only in the BASIC Language System.

The following illustration shows how a DBGET transfers data values from a data entry to a variable list and a packing variable. The syntax examples are in BASIC.

```
DBGET(...,"@",Intvar,Charvar$(1,8),Shortvar,Doublevar,Longvar,...)
```



```
DBGET(...,"@",Packingvar$(1,26),...)
```

Full Record List Mode**Item List Mode**

If you want to read less than all of the items from an entry you could use full-record mode with a packing variable and ignore the undesired items in the packing variable. You can also use item list mode. If you want to write to less than all the items you must use item list mode.

Item list mode implies that you are specifying a list of the names or numbers of each item to be read from or written to. Names are the same names specified in the schema or DEFINE. The names are separated by a comma. They do not need to be in the same order as the items in the entry. The name list must be specified as a single expression or variable in the parameter list. The item numbers are the numbers assigned by SCHEMA or DEFINE when you created the root file. Items numbers starting with one are assigned in the same order that you specified the items in the Item Part. The suggested method of determining item numbers is via DBINFO mode 101 within your program.

¹ BASIC only

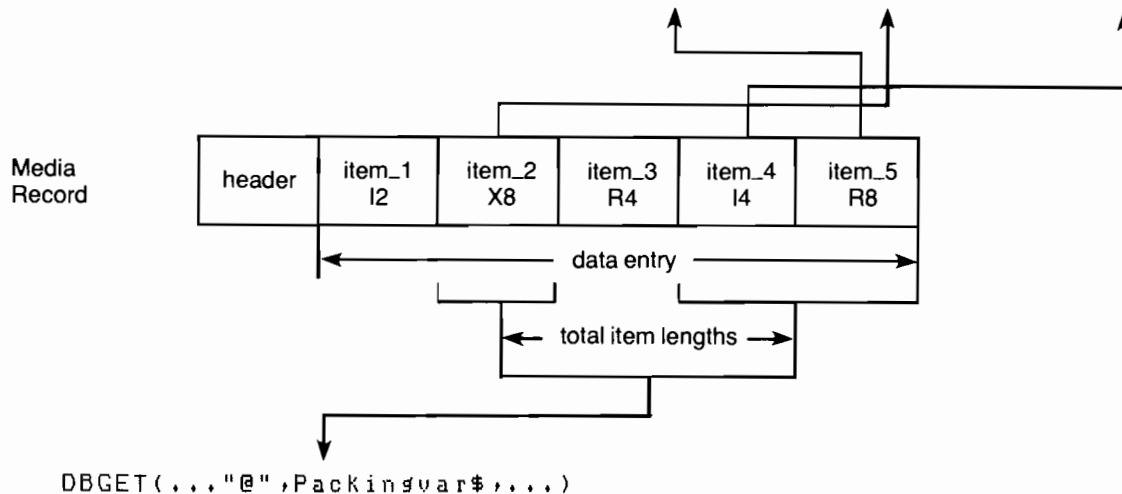
An item list consisting of item names is a string expression. The names are separated by a comma. The list is terminated with a semicolon. An item list of item numbers is an integer array. The first element specifies the number of item numbers. Each succeeding element is an item number. Item number list mode is not available in BASIC.

With an item list you can use either a variable list or a packing variable for the data within the following constraints:

- The packing variable must have a length in bytes equal to or greater than the sum of the lengths of the named or numbered data items.
- The list of variables must contain one variable for each data item name or number in the item list. Each variable must be the same type as the item. The variables must be in the same order as the items in the item list, not the entry. This method is available only in BASIC.

The following illustration shows how a DBGET transfers data values from a data entry to a variable list and a packing variable using item list (name list) mode. The syntax examples are in BASIC.

```
DBGET(...,"Item_5,Item_2,Item_4",Longvar,Charvar$(1,8),Shortvar,...)
```



Item List Mode

If you find that you need to specify all of the item names or numbers, consider using full-record mode.

You cannot use pseudonym names in the name list. To access a data item by its pseudonym name you must first request the item number with DBINFO mode 101, and then request that item number's name with DBINFO mode 102.

Current List Mode

The most recently executed DBPUT, DBGET or DBUPDATE which specifies either full record or item list mode establishes the current list mode. In subsequent statements you can simplify program coding and improve performance by specifying that you want to use the current list mode.

The list specifier for current list mode is a string expression which evaluates to an * (asterisk) character followed by a semicolon.

Program Variables for Item values

There are two ways to specify the data written to the data base with DBPUT and DBUPDATE and read it with DBGET. In the BASIC Language system you can specify either a single packing variable or a list of program variables. In HP-UX you can specify only the packing variable form. In either case you must keep the following in mind.

Note

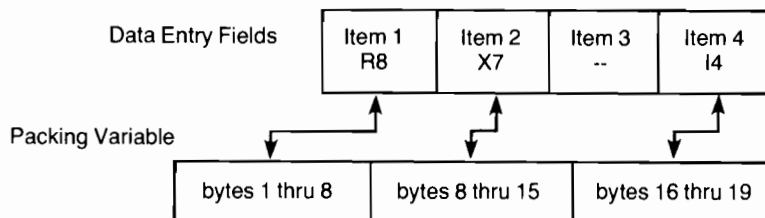
The values stored in a data entry for data items are not typed. The data entry fields contain only the item values. There is no information in the data entry about the data type (integer, shortreal, string) or size of any item. You must specify the correct type and size of program variables when reading from and writing to a data entry.

Item Data in a Packing Variable

The first method, and the only method available in HP-UX or for DBINFO in BASIC, is to read the data values from the data base into, or write them to the data base from, a single variable. In BASIC, it must be a string variable and is called a "packing string".

In this mode data values are transferred directly to and from the data entry. For example, during DBGET, each item's value is simply read from the entry field and "packed" into the packing variable. To retrieve this data from the packing variable, you must know the type and size of each item. If you obtained the entry in full-record mode, you must further know how the items are organized in the entry.

The List Specifier section illustrated examples of the use of the packing variable. The following illustration shows the relationship between the item values in the entry fields and the item values in the packing variable.



Packing Variable Data Transfer

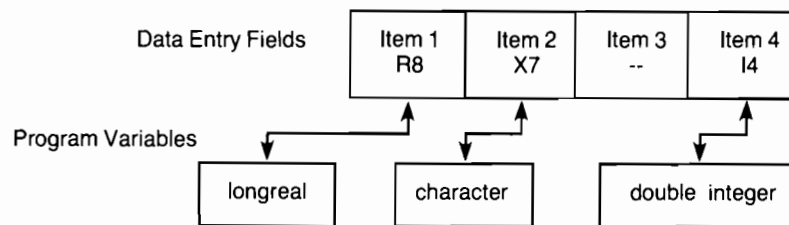
Methods of building or reading from a packing variable vary considerably between languages. For example, in FORTRAN you can simply EQUIVALENCE your program variables to locations within the packing variable. In BASIC you must write to and read from the variable with the PACK USING and UNPACK USING statements provided specifically for that purpose. Accessing the packing variable is a major topic of the language Appendices.

Item Data in a Variable List

The second method, available only in BASIC, is to specify a list of expressions or variables, each of which represent an individual data base item in the current entry. Each must be an appropriate data type for the item. For example, you cannot read shortreal data into a string variable or double integer or vice-versa.

This mode is simple to use because you need only account for the appropriate number and type of items. You can determine this from the schema, or your program can make inquiries with DBINFO.

The preceding section (List Specifier) showed illustrations of the use of the variable list. Here is another showing the relationship between item values in entry fields and program variables.



Variable List Data Transfer

Adding Data to the Data Base - DBPUT

Once you have successfully opened your data base, you can add data to it using the DBPUT intrinsic. Keep in mind that you can add an entry using DBPUT only in a manual master data or detail data set. Entries in automatic master data sets are added automatically by IMAGE.

Performing a DBPUT operation is relatively simple. You do not need to specify the access method. IMAGE uses the method appropriate for the data set type. However, there are some considerations you should keep in mind, so DBPUT operations in each data set type are described separately. There are examples of DBPUT in the language Appendices.

For either type data set, you must have the data base open in one of the read/write DBOPEN modes (1, 3 or 11¹). You cannot perform a DBPUT in read-only DBOPEN mode 8.

DBPUT in a Manual Master

When you add a detail data entry which contains a new value for a search item, IMAGE adds the key item value automatically to the automatic master data set in which it is defined. IMAGE does not add it automatically to a manual master data set. You must perform a DBPUT operation in the manual master.

IMAGE uses the calculated (hashed) access method to select the (primary) record. If the record is occupied, IMAGE uses the next record on the free record chain and links this record to the primary record's synonym chain.

When you perform the DBPUT, keep the following in mind:

- If you are using item-list mode, you should specify the key item name or number. If you do not specify the item, IMAGE uses its null value. You can only put the null value of the key item in the data set once.
- Whether or not you specify the key item and its value, the value to be stored in the data set for the key item must be unique. If the key item value already exists in the set, a status error occurs.
- The data set must not be full. There must be at least one empty record in the data set. If the data set is full you can extend it with DBEXTEND. You should use this as a temporary measure until you can re-create the set with a larger defined capacity.
- If you opened the data base in shared DBOPEN mode 1, you must lock the data set or the entire data base with DBLOCK before invoking DBPUT.

¹ BASIC only

DBPUT in a Detail Data Set

When you add a data entry to a detail data set, IMAGE automatically places the new entry on the correct chain(s) in the data base. You do not have to perform a DBFIND to establish a chain.

IMAGE uses the first available empty record from the detail set's free record chain. IMAGE stores your data in the entry and links it to each search item chain in the data set. If the value of a search item does not exist in the related master, then the chain does not yet exist. If it is an automatic master, IMAGE creates the entry in the master, thus creating the new chain. If it is a manual master, a status error occurs. You must first perform a DBPUT in that manual master to establish the master entry and the chain.

When you perform the DBPUT, keep the following in mind:

- All related master data sets must be on-line. This is not a concern if the entire data base resides on a single medium.
- Each related manual master must have an entry containing the value of the search item for that chain. If not, perform a DBPUT in the master before the DBPUT in the detail.
- Each related automatic master must have either an entry for the search item value or an empty entry which IMAGE can use to create the entry.
- The detail data set must not be full. There must be at least one empty record in the data set. If the data set is full you can extend it with DBEXTEND.
- If you opened the data base in shared DBOPEN mode 1, you must lock the data set and its related masters or the entire data base with DBLOCK before invoking DBPUT.

If you use item-list mode and omit any items, IMAGE uses their null values.

IMAGE links new entries onto the end of unsorted chains. Subsequent FORWARD CHAINED reads along unsorted chains encounter these entries in chronological order. IMAGE inserts new entries into sorted chains. IMAGE uses a BACKWARD CHAINED read to establish the location along the sorted chain. Subsequent FORWARD CHAINED reads along sorted chains encounter different search item values in sorted order, and identical search item values in chronological order.

Locating a Chain - DBFIND

In every DBGET mode except CHAINED you can read data from both master and detail data sets without using DBFIND. The use of CHAINED DBGET in detail data sets is sufficiently common that it is appropriate to introduce DBFIND first.

A detail data set can have paths to several master sets. Each path represents a collection of chains. Each chain represents a unique value of the search item on that chain. To perform a CHAINED read, you must establish the desired path and chain. The normal way to do this is with DBFIND.

You do not use DBFIND to prepare for CHAINED reads in a master data set (along a synonym chain). Use a CALCULATED, DIRECTED or SERIAL DBGET to locate a master entry. This automatically establishes that record's synonym chain as the current chain in the master set.

Normal DBFINDs

In the DBFIND intrinsic you specify the detail data set in which you intend to perform chained reads. You also identify the path and chain along which you intend to read by specifying the search item and its value respectively. DBFIND uses the search item identity to select the appropriate master data set (path), and uses the search item value to perform a calculated access to the master set entry containing that value. The entry's media record contains the pointers to the desired chain. IMAGE stores these pointers in the buffer area associated with the detail data set.

After a successful DBFIND, you can perform a CHAINED FORWARD read to the first detail data set entry on the chain or a CHAINED BACKWARD read to the last entry on the chain.

Once established, the "current chain" definition is used for all subsequent chained reads in that detail data set. Each data set has its own current chain information.

There are examples of a DBFIND followed by a DBGET in the language Appendices.

If you have the data base open in DBOPEN mode 1 (shared read/write), you should lock the data base or relevant master and detail sets with DBLOCK prior to the DBFIND/DBGET operation. Even if you do not intend to modify the data, another user could delete a chain entry between your DBGET operations. This can cause a broken chain status error in your program.

Note

DBFIND does not retrieve any data from the data base. It merely locates the chain head/end for the specified detail data set and search item value.

Shortreal and Longreal Search Items

If the search item is a real item (R4 or R8), you need to be cautious about the method of generating the search value. Assume that PRICE is a search item in our sample data base (it is not). If you attempt to generate a histogram of book prices in the library with the following (BASIC) loop, you are apt to find no books for any price but the first.

```

1910 FOR Price=0.01 TO 100.00 STEP 0.01
1920   DBFIND(Base$,"BOOK",1,Status(*),"PRICE",Price)
      :
1990 NEXT Price

```

The reason for this is that binary floating point formats, like the IEEE format used by IMAGE, cannot precisely represent all rational numbers. A small amount of round-off error accumulates during the loop calculation. The calculated access performed by DBFIND requires that the longreal or shortreal value have exactly the same bit pattern as the key value in the master data set.

The way to ensure that the value in the data base and the value specified in DBFIND match, is to always round or truncate the value to the number of significant digits, or power of ten required. The previous example works properly if you add the following line.

```

1912   RPrice=PROUND(Price,-2)    !Round to .01 significance
1920   DBFIND(Base$,"BOOK",1,Status(*),"PRICE",RPrice)

```

Changing the Current Path and Chain

To change the current chain in a detail data set, you can execute another DBFIND which specifies the new information. This is the only way to change both the path and chain. There are other ways to change just the chain.

As long as you perform only CHAINED DBGETs in the detail set, the current chain does not change. If you perform any other access, such as a SERIAL or DIRECTED DBGET, the current chain may change. If the SERIAL or DIRECTED DBGET selects a record that has a different value for the search item, the current chain changes. Subsequent CHAINED DBGETs read along the chain defined by the new search item value. Note that DBGET uses the same search item identity, that is, the path does not change.

The Primary Path

There is a special case in which you can execute a CHAINED DBGET in a detail set without first executing a DBFIND. This case uses the data set's "primary path" and current chain. The primary path is the first path defined for the data set in SCHEMA or DEFINE. The current chain is the chain determined by the combination of the primary path and the current record in the data set.

The current record implies that you have performed at least one DIRECTED or SERIAL DBGET within the data set. If you execute a CHAINED DBGET specifying a data set that has not been accessed since DBOPEN, you get a status error.

IMAGE provides the primary path access technique primarily for use by Hewlett-Packard data base utility programs. It is not recommended for use in normal programming.

Retrieving Data - DBGET

You use the DBGET intrinsic to select a data entry and read the values of data items it contains. The selection process is one of the seven access modes described in Chapter 1. All except the CALCULATED access methods (DBGET modes) are available in both master and detail data sets.

This section is organized by operations, and discusses the modes in the order in which you are most likely to require their use. The following tables summarize the modes.

If you are performing a DBGET to read a data entry which you intend to modify with DBUPDATE or DBDELETE/DBPUT, you should lock the data base or data set with DBLOCK prior to performing the DBGET.

Access Mode	Access Mode Name	Entry Record Obtained	Access Mode Argument
1	RE-READ	Current	ignored
2	SERIAL FORWARD	Next non-empty	ignored
3	SERIAL BACKWARD	Previous non-empty	ignored
4	DIRECTED	Specified	Record Number
5	CHAINED FORWARD	Next on synonym chain	ignored
6	CHAINED BACKWARD	Previous on synonym chain	ignored
7	CALCULATED (hashed)	Function of key item value	Key Item value

Master Data Set DBGET Access Modes

Access Mode	Access Mode Name	Entry Record Obtained	Access Mode Argument	DBFIND required
1	RE-READ	Current	ignored	No
2	SERIAL FORWARD	Next non-empty	ignored	No
3	SERIAL BACKWARD	Previous non-empty	ignored	No
4	DIRECTED	Specified	Record Number	No
5	CHAINED FORWARD	Next with same search item value	ignored	Yes if there is no current chain
6	CHAINED BACKWARD	Previous with same search item value	ignored	Yes if there is no current chain
7	CALCULATED	n.a.	n.a.	n.a.

Detail Data Set DBGET Access Modes

CHAINED Reads - DBGET Modes 5 and 6

DBGET modes 5 and 6 locate the next or previous data entry along the current chain and return the entry's item data values. The normal use for modes 5 and 6 is to read or search the list of entries along a chain in a detail set. For example, in our sample data base, if you want to obtain a list of all the books by a single author, you need to perform a chained read of the detail data set BOOK. The language Appendices have an example of this form of DBGET.

Chained Reads in Detail Data Sets

DBGET mode 5 and 6 use the current chain in the specified detail data set. Current chain information is maintained by IMAGE for each open data set. The normal method of performing chained DBGETs is summarized in the following steps.

1. Perform a DBFIND specifying the desired detail data set and search item value. This establishes the path and chain.
2. Perform a DBGET mode 5 (FORWARD) or mode 6 (BACKWARD) to locate and read the first or last entry on the chain.
3. Continue reading FORWARD to the end of chain (status error 15) or BACKWARD to the beginning of chain (status error 14).

Once a chain is established in a detail set, it remains in effect even while you are performing operations in other data sets. However, if you perform any SERIAL or DIRECTED reads in the detail set, you may change the current chain (see the DBFIND section). The best policy is to always perform a DBFIND before beginning a series of chained reads.

Chained Reads in Master Data Sets

You can also use DBGET mode 5 and 6 in a master data set to determine which key item values are hashing synonyms. This type of DBGET is typically used only as a data base performance tool and is not used in applications. Data base performance is discussed in the "Advanced Data Base Operations" chapter.

Before you perform a DBGET mode 5 or 6 in a master data set, you must establish a current chain in that data set. Any DBGET which successfully accesses an entry (occupied record) establishes a current chain, but does not place you at any particular position on the chain. You must perform a chained read both forward and backward to locate the chain ends.

You can also establish a current chain in a master set with a successful DBPUT. DBPUT always adds a new record to the end of the synonym chain.

In either case, keep in mind that a master set having an appropriate capacity has few, if any, synonyms.

Reading a Key Item's Entry - DBGET Mode 7

DBGET mode 7 (CALCULATED read) locates the master data set entry which contains the specified key item value. The normal use for mode 7 is to read the values of items associated with the key item in a manual master. For example in our sample data base, if a book is overdue in the INVENTORY data set, you need to use the EMPLOYEE_NO from the INVENTORY set as the key item value in a DBGET mode 7 in the BORROWER data set so that you can get the employee's phone number.

As with DBFIND, you should round real key item values to ensure proper calculation of the record number.

You cannot use DBGET mode 7 to establish a detail chain, nor can you use DBGET mode 7 in a detail set. Use DBFIND to establish the chain.

You can use DBGET mode 7 in an automatic master, but its use is typically limited to determining that the specified key item value actually exists.

Locating a Synonym Chain

Each primary entry in a master data set is the head of a synonym chain. The synonym chain links together all of the secondary entries which “hashed” to the same record numbers. If you have a large number of different key item values which “collide” at the same record number when hashed, you may have numerous or long synonym chains, or both.

Each CALCULATED, DIRECTED or SERIAL DBGET in a master data set establishes a synonym chain for CHAINED DBGET reads in that set. See the “Advanced Data Base Operations” chapter for more information on evaluating synonym chains and data base performance.

Reading Every Entry - DBGET modes 2 and 3

The SERIAL read DBGET modes 2 and 3 locate and read the next or previous entry in the specified data set. The normal use for modes 2 and 3 is to either read all the data in a data set or to search the data set for a non-search, non-key item value. For example, suppose you want to generate a report of books 30 days or more overdue from our sample data base. You must perform a SERIAL read of the INVENTORY data set and examine each BORROW_DATE item value.

Performing a SERIAL read is not the same as performing a DIRECTED read for every record in the data set. SERIAL read always locates the next non-empty record, if there is one. DIRECTED read can attempt to read an empty record.

You can use SERIAL read in both master and detail data sets. The normal method of performing a SERIAL read is as follows.

1. Perform a DBGET mode 4 read specifying record zero. This “rewinds” the data set to record “0”.
2. Perform either a DBGET mode 2 FORWARD or mode 3 BACKWARD SERIAL read to locate and read the first or last entry in the data set.
3. Continue reading FORWARD to the end of file (status error 11) or BACKWARD to the beginning of file (status error 10).

Keep in mind that if you execute SERIAL reads after a CHAINED read, you can alter the current chain if the last entry obtained by a SERIAL read contained a different search item value than the original chain.

Rewind or Select - DBGET mode 4

DBGET mode 4, DIRECTED read, specifies the data set record desired. There are two principal uses for mode 4: “rewinding” the data set to record “0”, and reading an entry whose record number your program has somehow determined.

Rewinding a Data Set

The actual (media) records in a data set are numbered from record 1. There is no record 0. Record 0 is merely a convention IMAGE uses to denote “no current record”. Record 0 has the property that from it you can perform forward or backward reads to the first or last entries in the set or entries on a chain.

If you are going to perform a SERIAL read, you can use DBGET mode 4 specifying record 0, to ensure that you are in a position to read all the entries in the set or chain.

You can also rewind a data set with DBCLOSE mode 3, however, DBCLOSE mode 3 causes IMAGE to clear the current chain and the current list specifier. This means that you must execute a new DBFIND before performing any subsequent CHAINED DBGETs and the next DBPUT, DBGET or DBUPDATE must re-establish the list specifier. DBGET mode 4 retains the current chain and list specifier and is therefore the preferred method of rewinding a data set.

Selecting an Entry

For record number values other than zero, DBGET mode 4 attempts to read the media record having that number. It is only an attempt because no read occurs if the record is empty (has no entry).

In general, DIRECTED reads to other than record 0 are not good programming practice. If a data base is open in shared read/write mode, another user could alter or delete an entry after you have determined its number but before you can access it.

It may happen that you have located an entry by a time-consuming SERIAL search and need to perform other accesses in the same data set and then return to the original entry. If you chose to do this with a DIRECTED read, be certain to lock the data set for the duration of all accesses to it.

It is also possible for a power or hardware failure to corrupt a data base. If the data set has broken chains as result of such a failure, and you want to recover whatever data you can, you can use DIRECTED reads to examine each record. The preferred practice is to back up your data base frequently and to have your data entry/update programs log their data base updates in a separate (serial) data file. This makes it relatively simple to recover all but the last transaction.

Re-read Current Record - DBGET mode 1

DBGET mode 1 re-reads the current record of the selected data set. Mode 1 performs no entry location operations. There must be a current record established for the data set. You can establish a current record with any of the other DBGET modes and with DBPUT. DBFIND does not establish a current record, instead, it rewinds the detail set to record 0.

There are two circumstances in which you might find it useful to perform a RE-READ DBGET:

- If the data entry is very large (long), you can read individual items from the entry with successive individual re-reads. You need to employ the item-list form of the list specifier to select each item.
- If you are using DBOPEN mode 11 or have executed a DBCONTROL mode 1, you can use RE-READ to verify that the data has been properly posted to the disc after each DBPUT or DBUPDATE.

Note

The re-read performed by DBGET mode 1 can return a status error if the current record was established by a DIRECTED read to an empty record. It can also return a status error if another (shared) user has deleted the entry.

Deleting a Data Entry - DBDELETE

In the specified data set, DBDELETE deletes the entry in the current record. It returns the current record to the empty state, and alters the pointers in the header to link the record to the end of the free record chain. DBDELETE has two purposes:

- You can use DBDELETE to delete an entry which contains incorrect or obsolete data;
- You must use DBDELETE, followed by a DBPUT, to modify the key, search or sort items of an entry. You cannot use DBUPDATE for this purpose.

There are different considerations for DBDELETEs in either master or detail data set. They are summarized in the following paragraphs. One consideration common to both is that if you need to delete all of the entries in an entire data set, you should use DBERASE and not DBDELETE.

Deleting in a Master Data Set

Keep the following in mind when deleting entries in a master data set:

- The set must be a manual master data set. You must lock the set with DBLOCK if you opened the data base in shared read/write DBOPEN mode 1.
- You cannot delete entries in an automatic master, IMAGE performs the delete for you.
- Any chains from the entry to detail data sets must be empty. If there are any related detail entries, you must first delete them before deleting their master entry.
- After the DBDELETE, the current record number is unchanged. The record is, of course, empty, but you are still on the same synonym chain and can perform CHAINED DBGETs forward and backward. If the deleted entry had no synonyms, the synonym chain no longer exists.
- If the deleted entry was a primary entry, and had synonyms, it is now empty but is still on the synonym chain, as well as being on the free record chain. IMAGE/9000 does not "migrate" the first secondary into the deleted entry in order to make it a primary entry. The record can be used by subsequent DBPUT operations.

Deleting in a Detail Data Set

Keep the following considerations in mind when deleting an entry in a detail data set:

- If you opened the data base in shared read/write DBOPEN mode 1, you must lock the detail set and all related masters with DBLOCK.
- The current record number and current chain are unchanged after a DBDELETE. The record is empty, but you can still perform CHAINED DBGETs forward to the next entry and backward to the previous entry. If you go in one direction and then the other, you do not return to the deleted record.
- If the deleted entry was the last entry linked to an entry in an automatic master, IMAGE automatically deletes the master entry.

Modifying Data in the Data Base - DBUPDATE

There are two ways to modify data in a data entry:

- Change one or more items in the entry with DBUPDATE.
- Delete the entry with DBDELETE and replace it with DBPUT.

DBUPDATE modifies the current entry in the specified data set. DBUPDATE cannot locate an entry (establish a current record). Normally you have executed a DBGET to locate the entry you wish to update. A prior DBGET is not required. You can also establish the current entry with a DBPUT. DBUPDATE specifies new data values for items in that entry. The items cannot be key, search or sort items. You cannot use DBUPDATE in an automatic master.

DBUPDATE is like DBPUT in that you specify a list mode and the data value(s) to write to the entry. DBUPDATE is unlike DBPUT in that IMAGE does not use null (default) values for items omitted from the list. If you do not supply a value for an item, it remains unchanged.

Subject to the limitations of your password, you can always perform a DBDELETE/DBPUT. You cannot always perform a DBUPDATE. Nonetheless, DBUPDATE is useful because it is simpler and quicker than DBDELETE/DBPUT. Like DBPUT, you must have the data base open in one of the read/write DBOPEN modes (1, 3 or 11). You cannot perform a DBUPDATE in read-only DBOPEN mode 8.

If you opened the data base in shared read/write DBOPEN mode 1, then like DBPUT, you must lock the set with DBLOCK. Unlike DBPUT, in a detail set, you do not need to lock related masters.

Note

When you modify an entry with DBUPDATE, none of the chain pointers change. The entry is still in the same position on each chain. However, it is no longer in its chronological location on those chains. If having entries in proper chronological order is important, do not use DBUPDATE. Use a DBDELETE followed by a DBPUT instead.

The reason that you cannot always perform a DBUPDATE is that DBUPDATE cannot relocate the record or change chain pointers. If you could change a key item value, IMAGE would have to move the record so that the new hash value pointed to it. If you could change a search item, IMAGE would have to link it to a new chain. If you could change a sort item, IMAGE would have to consult the root file to determine that it is a sort item, and then change the chain pointers for the related search item(s). All of these activities would take as much time as DBDELETE/DBPUT. To provide good performance you can only alter non-key, non-search, non-sort items. You can specify key, search and sort items, but their values must match those already in the data entry.

Coordinating Access - DBLOCK/DBUNLOCK

The DBLOCK intrinsic gives your program temporary exclusive write access to one or more data sets, or the entire data base. It does this by preventing other users from executing DBLOCK intrinsics specifying any of the same portions of the data base.

If you have a data base open in shared write DBOPEN mode 1, IMAGE requires that you lock the affected data sets, or the entire data base, during DBPUT, DBUPDATE and DBDELETE operations. Locking is not required for DBINFO, DBFIND and DBGET operations, nor are these operations denied access to data sets by any locks in effect.

When you perform a DBLOCK, the lock is placed on the data base number. The data set or data base is said to be locked to that data base number. If you have the data base open to more than one data base number, your own program is denied write access to the affected data sets via the unlocked data base numbers.

How Not to Use Locking

Because locking is not required for DBFIND and DBGET operations in DBOPEN mode 1, it is not generally sufficient to lock the data set/base just for the write operation. Consider the following case in which two users want to revise the price of a book in the BOOK data set. The example starts just after their DBFINDs.

User One	User Two
↓	↓
DBGET(B\$, "BOOK", 5, S(*), "PRICE", Price, X)	
↓	↓
↓	DBGET(B\$, "BOOK", 5, S(*), "PRICE", Price, X)
↓	↓
Price=Price+2.00	↓
DBLOCK(B\$, "BOOK", 3, S(*))	↓
DBUPDATE(B\$, "BOOK", (*), 1, S(*), "PRICE", Price)	↓
DBUNLOCK(B\$, "BOOK", 2, S(*))	↓
↓	Price=Price+3.00
↓	DBLOCK(B\$, "BOOK", 3, S(*))
↓	DBUPDATE(B\$, "BOOK", 1, S(*), "PRICE", Price)
↓	DBUNLOCK(B\$, "BOOK", 2, S(*))
↓	↓

Improper Use of Locking

Both of these requests execute without error, but the data in the data base is now incorrect. The final book price is not the old price plus 5.00, but the old price plus 3.00. Both users updated the original price.

Other errors are possible if you perform locking only "around" the DBPUT, DBUPDATE or DBDELETE intrinsic. For example, if user one had performed a DBDELETE instead of the DBUPDATE, user two's DBUPDATE would fail, returning a status error.

The Correct Way to Use Locking

To avoid invalid data and unexpected status errors, you should perform locking around your entire "transaction". If you are performing a DBFIND, followed by a one or more DBGETs, and finally a DBPUT, DBUPDATE or DBDELETE, you should perform the DBLOCK prior to the DBFIND.

Here is the previous example with correct locking. User two's entire update is deferred until user one's update is complete. When both users are finished, the book price is the old price plus 5.00.

User One

```

↓
DBLOCK(B$, "BOOK", 3, S(*))
DBFIND(...
DBGET(B$, "BOOK", 5, S(*), "PRICE", Price, X) Price=Price+2.00
DBUPDATE(B$, "BOOK", (*), 1, S(*), "PRICE", Price)
DBUNLOCK(B$, "BOOK", 2, S(*))
↓
↓
↓
↓
↓
↓
↓
↓

```

User Two

```

↓
↓
↓
↓
↓
DBLOCK(B$, "BOOK", 3, S(*))
DBFIND(...
DBGET(B$, "BOOK", 5, S(*), "PRICE", Price, X)
Price=Price+3.00
DBUPDATE(B$, "BOOK", 1, S(*), "PRICE", Price)
DBUNLOCK(B$, "BOOK", 2, S(*))
↓

```

Proper use of Locking

Locking With and Without Wait

Whenever you execute a DBLOCK, you must consider the possibility that one or more of the data sets you are trying to lock may already be locked to another data base number. In the coding of the DBLOCK, you have two choices:

- The execution of your program can be suspended until all of the sets are available. This is known as locking "with wait". Your program waits until the lock is granted.
- Your program can continue executing whether or not the lock is granted. In this case, known as locking "without wait", you must examine Status array element 3 after the DBLOCK to determine if the lock was granted.

The following table summarizes the DBLOCK modes. The paragraphs following discuss the locking of the different portions of the data base.

Lock Mode	Locked Portion of Data Base	Wait?	Data Set List Specifies
1	Entire data base	With wait	<ignored>
2	Entire data base	Without wait	<ignored>
3	Data set	With wait	A single set
4	Data set	Without wait	A single set
7	One or more sets	With wait	A list of sets
8	One or more sets	Without wait	A list of sets

DBLOCK Modes

Locked Portion of Data Base

In the DBLOCK intrinsic, you can lock a single set, multiple sets or the entire data base. This section discusses how to decide what to lock, and how IMAGE allocates the lock(s).

What Should You Lock?

You are only required to lock those data sets in which you are actually adding or changing data or pointers. For example, when you add an entry in a detail set, it causes changes to pointers in related master entries. It can also cause the addition of an entry to an automatic master.

You have seen that you should generally execute DBLOCK earlier than it is actually required. Likewise, in many cases you should lock more than just the required sets. For example, if you are performing a DBUPDATE in a detail set, you may want to lock related manual masters so that another user cannot change non-key information related to the chain you are using.

Intrinsic & Access	You Must Lock...	You Should Lock....
DBPUT, Master	Master	Master
DBPUT, Detail	Master and Detail	Master and Detail
DBUPDATE, Master	Master	Master
DBUPDATE, Detail	Detail	Master and Detail
DBDELETE, Master	Master	Master
DBDELETE, Detail	Master and Detail	Master and Detail
DBFIND	None	Master and Detail
DBGET, Master	None	Master
DBGET, Detail	None	Master and Detail
DBINFO mode 504*	None	Entire Data Base
DBINFO, other modes	None	None
DBBUFFER	None	None
DBCONTROL	None	None
DBOPEN	n.a.	n.a.
DBCLOSE	n.a.	n.a.

* Requests modification count.

Suggested Locking

IMAGE Lock Allocation

There is no practical difference between a DBLOCK operation requesting the entire data base (modes 1,2) and a single DBLOCK which requests all of the sets (modes 7,8). In either case, with or without wait, your program obtains all of the sets or none. The same is true of modes 5 and 6. IMAGE grants either your entire request or none of it. Conversely, when you execute a DBUNLOCK, you can release any number of sets, or all of them, regardless of how you obtained the locks.

There is a difference between locking some number of sets in a single statement and in multiple statements. To prevent "deadlock", IMAGE does not allow you to execute a DBLOCK with wait if you already have any locks in effect on any data bases opened in your program.

If you need to "accumulate" locks on sets, it is suggested that you not execute DBLOCKS without wait while you have a lock in effect. To do this you generally need to test Status(3) and re-execute the DBLOCK at a later time. If you do this, you can effectively cause a situation similar to the deadlock IMAGE is trying to prevent. In the following example, the two users are trying to lock sets "INVENTORY" and "BOOK" using separate DBLOCKS.

User One

User Two

<pre> ↓ 100 DBLOCK(Base\$,"INVENTORY",3,Status(*)) ↓ ↓ ↓ 200 DBLOCK(Base\$,"BOOK",4,Status(*)) 210 IF Status(1)=20 THEN GOTO 200 ↓ ↓ ↓ ↓ </pre>	<pre> ↓ 100 DBLOCK(Base\$,"BOOK",3,Status(*)) ↓ ↓ 200 DBLOCK(Base\$,"INVENTORY",4,Status(*)) 210 IF Status(1)=20 THEN GOTO 200 ↓ </pre>
--	--

In this example, both programs successfully obtain their first lock, but neither can obtain their second lock because the other program has the lock. Both programs loop indefinitely attempting to obtain the lock.

If you have only a single data base open, the proper way to increase (or change) the sets on which you have locks is as follows:

- Release all of the old locks you have in effect with DBUNLOCK.
- Request all of the new locks you require with a new DBLOCK.

Reading the ROOT File - DBINFO

DBINFO returns information from the data base ROOT file to a packing variable. It does not provide access to all of the information in the ROOT file; for example, you cannot read passwords.

Why do you need DBINFO ?

The IMAGE intrinsics introduced so far require that you know the following about the data base:

- the file specifier of the data base and a valid data base password;
- name of the data set you want to access;
- the name of any data items you want to access;
- the type and size of the data item.

This provides your data base program with a significant amount of independence from the structure of the data base; however, should you re-create the data base with an item removed or an item type changed, you do have to change your program if you are not using DBINFO.

You can avoid program changes by having your programs request data base information with DBINFO. DBINFO can return all of the information in the previous list with the exception of the data base specifier and password.

Although you can write a program which has variables and constants that describe every aspect of the data base, this is not suggested. In fact, it defeats a major purpose of using a data base: program independence. IMAGE provides the DBINFO intrinsic so that your program can "read" the root file and obtain all the information your program requires to correctly access the data base as it is currently structured.

This section of the manual does not discuss every possible use of DBINFO. It covers the two most important uses: retrieving type extension values and program independence from the data base.

Using CODEs and Type Extensions (BASIC Only)

The data type CODE and the type extensions PICTURE, PSEUDONYM and RANGE are provided primarily for use with the QUERY subsystem available with IMAGE/9000 for the BASIC Language System. If you want to use these features in your own programs, you must perform DBINFO requests, as QUERY does.

Using CODEs - DBINFO mode 108 (BASIC Only)

The CODE (C2) data type is a special case of the integer data type. The value stored in a data entry should always be an integer in the range 0 through 32. What distinguishes CODE from the ordinary I2 integer data type, and gives it its usefulness, are the CODE values stored in the root file. CODE values are strings. For a given CODE item, each CODE value in the root file is matched to a CODE number. The following table shows an example from the sample data base.

Item: LOCATION

CODE value stored in root file by SCHEMA/DEFINE and accessed via DBINFO mode 108	CODE number you store in data entry
"MANAGEMENT"	1
"MARKETING"	2
"PRODUCTION"	3
"R&D"	4
"FINANCE"	5
"PERSONNEL"	6
"QA"	7

Sample Code Values and Numbers

Although you have defined a data item as a CODE type, you cannot store its CODE value in a data entry. You can only store its CODE number. This means that your program must correlate the CODE value and number. You do this with DBINFO mode 108, which retrieves the table of CODE values defined for the item. The packing variable consists of an integer denoting the number of CODE values in the variable, followed by an ordered list of the CODE values. Each value is 16 characters long.

If your program has the CODE value, you must search the table of CODE values returned in the packing variable. Once you know at which starting byte the CODE value is found, you can calculate the associated CODE number with the algorithm:

$$\text{Code_number} = (\text{Start_byte} - 2) / 16$$

If your program has read a CODE number (with DBGET), you can locate the CODE value in the packing string with the following algorithm:

$$\text{Start_byte} = (\text{Code_number} * 16) - 14$$

Keep in mind that if DBGET returns a CODE number of 0, that denotes that the NULL value of the item was stored in the entry. The interpretation of the meaning of the null value is your responsibility because CODE number 0 has no CODE value.

If your program performs frequent CODE correlations using DBINFO mode 108, you can improve performance by loading the table of all CODE values into system memory with DBCONTROL. You can also avoid redundant DBINFO mode 108 operations by reading the CODE values once into global or common program variables and then accessing only the variables thereafter.

Note

IMAGE does not perform any conversions of CODE values to CODE numbers, nor does it check your CODE number to ensure that it is in the range 1 through 32. It is your responsibility to perform these operations. If another user of the data base does not observe this convention, a DBGET can return CODE numbers outside the range 0 through 32.

Using the PICTURE Extension - DBINFO mode 109 (BASIC Only)

Each data item in a data base has an associated PICTURE word stored in the root file. The PICTURE word is a double integer. The PICTURE word has any meaning that you care to give it. QUERY defines many common meanings to data items via PICTURE word values. For example, a PICTURE value of -536870912 (E0000000 hex) means that the associated string item is a name. The associated string should be stored in the form:

“Lastname,Firstname Middle,Title”

IMAGE does not automatically perform any checks on data values which have a non-zero PICTURE word. It is your responsibility to perform a DBINFO mode 109, evaluate the result, and convert your data to the appropriate form before performing a DBPUT. The advantage of the PICTURE word is that other programs (like QUERY) that read your data with DBGET can determine the form of the data.

Appendix Q discusses the QUERY data types. Note that there is a large range of values for the PICTURE word which are undefined. If you want to “invent” any new interpretations for PICTURE word values, you should select a value from the undefined range.

Note

The interpretation of the PICTURE word is a convention. Each program that performs a DBPUT, DBGET or DBUPDATE must subscribe to the convention in order for the PICTURE word to accurately describe the data in the data base.

Using the PSEUDONYM Extension - DBINFO mode 101 (BASIC Only)

One advantage of an IMAGE data base is that your programs can be independent of the structure of a data entry. They only need to supply the name of a data item and not its location in the entry. Normally you choose a simple, meaningful name for data items.

If your program interactively requests item names from a user, that user may not know all of the item names exactly. You may want to offer users of your programs the capability to enter any one of several “alias” or pseudonym names. Pseudonyms you should consider are: abbreviations, synonyms and common misspellings.

IMAGE does not accept PSEUDONYM names in DBPUT, DBGET and DBUPATE. It accepts only the DEFINE/SCHEMA item names. When supplied with a PSEUDONYM, your program must invoke DBINFO mode 101 to determine the item number. You can then use the item number in DBPUT, DBGET and DBUPDATE. You can also use the item number in DBINFO mode 102 if you want to determine the true item name.

If you want the complete list of PSEUDONYMs for any item, use DBINFO mode 107. If your program performs frequent PSEUDONYM correlations using DBINFO mode 107, you can improve performance by loading the table of all PSEUDONYM values into system memory with DBCONTROL. You can also avoid redundant DBINFO mode 107 operations by reading the PSEUDONYM values once into global or common program arrays and then accessing only the arrays thereafter.

IMAGE does not case-fold item names. "AUTHOR" is not the same as "Author" or "author". Since you can define only four PSEUDONYMs per item and a maximum of 255 PSEUDONYMs per data base, it is suggested that you not use PSEUDONYMs to handle upper and lower case variations of item names. Use the string operators and functions of your programming language to convert entered names to a standard form.

Using the RANGE Extension - DBINFO mode 109 (BASIC Only)

Each data item in the data base has an associated pair of RANGE values in the root file. You specify (or default) the RANGE values for each item in the schema or during DEFINE. IMAGE ignores the RANGE values during DBPUT and DBUPDATE. If you want to validate entered data item values against the item's RANGE, you must request the RANGE values with DBINFO mode 109.

Chapter 4

Data Base Maintenance



Introduction

IMAGE is designed to provide a higher level of data integrity than conventional file access methods. There is the possibility, however, of losing data or structural information due to hardware failure, or an operator error. For example, if you turn off the computer's power while a data base is open in read/write mode ¹, data base modifications may not have been written to disc, and are lost. When data or structural information is lost, the data base is said to be corrupt. It is essential that you adopt one or more strategies in anticipation of possible trouble. The first two sections of this chapter deal with data base backup and recovery.

As your data base accumulates data, you may find that your original estimates for data base capacity were inaccurate and you may need to increase the size of one or more sets. Also, unless you are an experienced data base designer, you may decide that your data base would serve you better if it had a slightly different structure. To change it, you need to create a new data base and copy the old data to it. The third section of this chapter deals with restructuring and discusses how to extend, unload and re-load your data base.

The Utility Intrinsic

This chapter introduces many of the uses of the utility intrinsic DBCREATE, DBERASE, DBEXTEND and DBPURGE and DBSTATUS. These intrinsic are unlike the data base management intrinsic in that the data base they specify must be closed. When executed, these intrinsic open the data base (exclusively), perform their operation, and close it.

With the exception of DBSTATUS, the utility intrinsic require a special password to perform their operations. This password is called the maintenance word. The maintenance word is not one of the DBOPEN passwords, although you can define it to be the same as one of them. You establish the maintenance word when you create the data sets with the DBCREATE utility. Once assigned, you can normally only change it by purging and re-creating the sets. If you have QUERY, you can also use it to change the maintenance word.

Data Base Validation

You should inspect your data base periodically to insure that it is not corrupt. It is especially important to do this prior to data base backup. A backup of a corrupt data base is worthless. When you are testing your data base programs, it is also reassuring to know that unexpected results are not due to a corrupt data base.

There are two methods of verifying that your data base is not corrupt. The first provides a quick "snapshot" of the state of the data base and reports any problems that IMAGE has already recorded. The second is exhaustive, and can find problems not yet detected.

¹ BASIC only

The Snapshot - DBSTATUS

This is a statement in BASIC and a command in HP-UX. It generates a report of the state of the data base. The form of the report is similar to a file directory listing. One of the columns of the report, STATUS, reports the state of each data set. IMAGE maintains a record in the root file for each set. This record has one of four states: ERASED, OK, PURGED and CORRUPT.

When a data set has no data entries because it has just been created (DBCREATE) or because it has just been erased (DBERASE), its status is ERASED. If it contains no entries because they have all been deleted, its status is not ERASED, but OK.

When you add the first entry to a created set (DBPUT), its status changes from ERASED to OK. It normally remains OK until erased (DBERASE) or purged (DBPURGE).

If you purge a data set with DBPURGE, IMAGE records its status as PURGED. If you purge a data set with any other purge command, this fact is not recorded in the root file. The set's status remains unchanged until the next access via IMAGE.

If IMAGE detects an error during data base access, such as a broken chain, it sets the STATUS to CORRUPT. If one or more data sets is CORRUPT you can only open the data base in read-only mode. By specifying DBOPEN mode 8, IMAGE returns a status error of 94, but the data base is open. See the "Corrupt Data Base Recovery" section of this chapter for more information about this access mode.

DBSTATUS does not detect any errors not already recorded. It merely provides a look at the record.

Exhaustive Validation

If you want to verify that all data entries are not corrupt, you need to write a simple program to do so. This process is very time consuming. You should only consider it for a small data base. The program must perform the following steps:

1. Open each master data set
2. Perform a SERIAL DBGET to obtain each occupied record
3. Perform a DBFIND for each linked detail set and each key item value found.
4. Execute a complete series of FORWARD CHAINED DBGETs until end of chain and BACKWARD DBGETs until beginning of chain.

This process also does not check free record chains. This is not generally a problem because the data is valid up to the time the broken free chain is detected. At that point you can unload the set, erase it and reload it.

Data Base Backup

Backing up a data base means making a copy of the item data, data set and root files or the volume on which the data base resides. If your data base becomes corrupt, you can re-build the entries, replace the data sets or replace the volume. There are four issues to consider in selecting a backup strategy:

- if you need to use the backup data base, how to bring it “up to date”;
- what technique to use;
- how frequently to back up your data base;
- how long to retain the backup copies.

It is not enough to simply back up your data base. As soon as the next modification occurs, the backup copy is out of date. If you need to use that backup copy, some data has been lost. How do you re-enter the lost data?

The general rule for backup technique is: select a method that is both easy enough to encourage its regular use and fast enough to permit a rapid recovery.

The general rule for frequency of backup is: don’t risk losing any more information than you care to re-enter.

The general rule for backup retention is: forever. Because this may not be practical, the following sections discuss the implications of this and the other issues, and offer some suggestions.

Note

Whenever you back up a data base, the data base should be closed to modification while the backup is in progress. If you back up a data base open in a read/write mode, the backup copy is likely to be corrupt when restored.

Re-entry of Data

Regardless of how frequently you back up your data base, if you have to use the backup copy, you will probably need to re-enter some of the data lost from the original data base. There are two ways to do this: re-enter the data manually, or; re-enter the data programmatically.

Manual Data Re-entry

You may update your data base infrequently and back it up after each “session”. If so, you may find that the most appropriate method for bringing the backup data base up to date is to simply run your application program and enter the data in same way that you entered it originally.

This method requires only that you retain a record of the original data. If you enter the data from documents, retain or duplicate those documents. If the information entered did not originate on a document, you may want to print out a record of each data base update. You can use this printed record to re-enter the data.

4-4 Data Base Maintenance

Programmatic Data Re-entry

If you are updating your data base frequently, it is always inconvenient, and often unacceptable to re-enter the lost data manually. Also, even if you are not updating the data base frequently, when you discover that the data base is corrupt, it may turn out that it has been corrupt for some time, and that one or more backup copies are also corrupt. In this case, your data base could be "down" for a considerable time while you re-enter data for a very old backup.

The best way to avoid these difficulties is to have your data entry programs generate a separate copy of the modifications made to the data base. This technique is called transaction logging or journalling. Transaction logging requires additional programming, but you can create a very simple system with the following technique. Use it for each program which modifies (DBPUT, DBUPDATE, DBDELETE) any data base.

1. When the program starts, create a log file with a unique name. You can construct the file name from the program name, data base name, date, time of day, or some combination of these which prevents duplicate file names, but still identifies the data.
2. If possible, the journal file should be on separate mass storage device from the data base.
3. Before each DBOPEN, DBPUT, DBUPDATE, DBDELETE and DBCLOSE operation the program performs, write a record of that operation to the journal file. Unless performance considerations require binary data, the data you write to the journal file should be character data. This allows you to view and/or edit the file with your environment's text editor. In BASIC, you should write a sequence (line) number before each data value.
4. Write an end of file (EOF) mark immediately upon creation or opening of the journal file. For each transaction, back up one record, write the data for each the intrinsic data followed by another EOF. In the BASIC Language System, this can be done easily with a DATA type file.
5. For each IMAGE intrinsic, write the name of the intrinsic and each parameter except the status array as a separate line or record. Write each data item value in a DBPUT, DBDELETE or DBUPDATE as a separate line. Here is a sample journal of a DBPUT compatible with the BASIC editor.

```
111 ! DBPUT
112 ! 03LIBRARY:CS80,7
113 ! SET,LIBRARY
114 ! MODE,1
115 ! LIST,@
116 ! X25,WAREHOUSE 32
117 ! X30(1),433 WASHINGTON ST.
118 ! X30(2),TIMNATH
119 ! X30(3),CO 80547
120 ! X25,FINSTER,PERCIVAL Q.
121 ! X14,(303) 555-1212
124 ! DBUPDATE
```

When you need to use your backup data base, restore it, and then execute your lost-data recovery program. This program merely opens the journal file and follows the instructions it finds there. If you take the additional measure of prefixing each data item value with its SCHEMA type identifier (I2, R8, etc.) or even its item name, you can write a single data recovery program which can be used with any data base and journal file in your system.

If you retain the journal files, you could actually recover a data base that had no other backup. You would re-create it and process all of the journal files. This is not recommended, however, because any recovery program which executes IMAGE intrinsics takes longer to execute than a utility which merely copies entire data set files. Also, transaction logging relies on the convention that all modifications to the data base are logged. If a user modified the data base with a non-participating program, such as QUERY, the reconstructed data base would have incorrect or incomplete data. You should use a journal file only to bring the most recent backup up to date.

Backup Techniques

When you choose a data base backup method, you need to decide what it is that you want to back up. There are three levels you can select.

- You can back up just the data and not the data base structure. This method normally treats each data entry as a separate record. This is the method used by QUERY/UNLOAD.
- You can back up both the structure and content of entire data sets by treating them as files, the method used by QUERY/BACKUP.
- You can back up an entire data base subdirectory or even an entire volume. This requires a physical copy of the disc media.

Entry Level Backups

The transaction logging discussed in the previous section is an example of an entry-level backup. For each modification of a data entry, a record is written to the journal file. Transaction logging occurs in “real-time” and is not usually considered a backup technique. The more common implementation is, at one time, to read every entry in the data base and write it to a backup file.

To perform an entry backup, you need to open each data set to be backed up and execute a SERIAL read for every entry in the set. The data values in each entry are written to the backup file. Entry backups are the slowest method of performing backup and are therefore not preferred, except when implemented in a transaction logging scheme.

Entry copying is recommended for data base restructuring, and is discussed in that section of this chapter.

File Level Backups

File level backup simply treats the root file and the data sets as ordinary data files. These files are, in fact, just data files. However, they have special file types and cannot be accessed by the file system intrinsics of your environment. This prevents accidental and unauthorized access to their data. Consequently, to perform a file backup, you must use a system intrinsic such as COPY (BASIC).

In the BASIC Language System, you can copy individual ROOT and DSET files with the COPY statement or the DISC_COPY utility program. You can also use the DBBACKUP capability of QUERY.

In HP-UX, use tar or cp. You must have super user access to do this.

Directory and Volume Level Backups

Your environment also supports utility programs that make a sector-by-sector or track-by-track copy of discs. These utilities are usually the fastest. If your data base is in use more than eight hours a day, you may only be able to close it for a short period of time. In this case, making a fast "binary disc image" copy of the data base subdirectory or the entire medium, may be the only choice.

The disadvantage of copies made at this level is that if you need to restore just a single data set, you may have to restore the entire directory or volume.

In the BASIC Language System, you can use the DISC_COPY program to copy an entire subdirectory, and the DISC_BACKUP program to copy an entire volume.

In HP-UX use the normal nightly backup.

Partial Backup

If you have a large data base or one in which most of the updates are restricted to a few sets, you may be tempted to back up just those sets. For example, in the sample LIBRARY data base, the INVENTORY data set changes daily, but the BOOK data set only changes when a new shipment of books arrive, or old books are sold. It is not sufficient to backup the INVENTORY set daily and the whole data base monthly. A change to the INVENTORY set can affect pointers in every related master data set and the root file.

Note

Due to the difficulty of assuring that the root file and related data sets reflect the correct state of a restored data set, you should always back up and restore the entire data base if you are using a file or volume-level technique. Only if you are using an entry-level technique can you be assured that the data base structure is correct after restoring part of the data base.

Frequency of Backup

Once you have selected a method for re-entering lost data, and have considered the backup techniques available, you need to decide how often to perform backup. To decide, you should consider how long it takes to recover from a problem. Here are some issues to consider.

- Is the most recent backup copy of the data base also corrupt? This is quite often the case if you discover that people have been adding systematically incorrect data to the data base. You can minimize the possibility of a corrupt structure by performing validation prior to backup.
- How long does it take to load the backup? Is there a person available to perform the recovery at all times? Is the backup disc or tape on site? Does the type of backup require that the system disc be off-line or unavailable to other users?
- How long does it take to re-enter lost data either manually or from the transaction log files?

Backup After Session

If you update your data base infrequently, an exhaustive analysis may not be necessary. In a casual data base, or one which is primarily read-only, a suitable backup technique is to provide backup capability in each program which can modify the data base. At the end of the session of updates, the program performs a backup.

At DBCLOSE, have the program prompt the user and inquire if a backup is desired. In cases where no physical interaction, such as mounting a tape, is required, the program can simply perform the backup automatically.

Backup on Modification Count

You may prefer to perform backup only after a fixed number of updates. For example, to limit the size of the transaction log file, you may want to backup after every 1000 modifications.

An easy way to accomplish this is to have your program request the data base Modification Count with DBINFO mode 504 just after DBOPEN and just before DBCLOSE. If the count “passed through” a multiple of 1000 while the data base was open, the program can perform or request a backup operation.

Note

HP-supplied data base backup utilities set the modification count to zero on successful completion. A DBERASE or DBPURGE/DBCLEAR of the entire data base also resets the counter to zero.

Scheduled Backups

With heavily used data bases, especially in multi-user environments, it is best not to rely on the data base users to perform backup. Backup is more likely to be reliably performed as part of regularly scheduled system maintenance. If one person is responsible for backup of the system's disc drives, that person should also backup the data bases, either as part of disc drive backup or as a separate operation.

If you have scheduled preventative maintenance on your system, you should always perform a backup of serviced disc drives prior to turning the system over to your HP Customer Engineer (CE). Should a problem be found in a sealed “internal” disc drive, the CE may need to replace the drive assembly. Needless to say, your data is not on the replacement disc platters.

Backup Retention

The first step in developing a procedure for backup retention is to determine whether any policies or regulations apply to your data. For example, many organizations are required by their insurance carriers to store backup copies of magnetic media either off-site or in a fireproof safe. Also, if your data base contains information related to product safety, the length of time you must retain the information may be established by statute or regulation.

Media Selection

You have three choices in backup media: flexible disc, 1/4-inch data cartridge and 1/2-inch magnetic tape. When selecting a medium, you need to consider capacity, cost, time and convenience.

Capacity may rule out the use of flexible discs. These discs are available in capacities of .25 to 1.2 million bytes (Mb). If any data set in your data base exceeds the size of the flexible disc, the backup process becomes significantly more complicated. No HP-supplied utility can copy a single data set file to more than a single flexible disc.

If you perform backups often, cost may also rule out the use of flexible discs. Flexible discs are more expensive than tape media. 1/2-inch 9-track magnetic tape is slightly less expensive than 1/4-inch 88140L/S data cartridges, but the initial cost of a 9-track tape drive is considerably higher than the incremental cost of a 1/4-inch cartridge drive in a CS80 disc.

In terms of time and convenience, the fastest media are the magnetic tapes and the most convenient are the flexible discs. You could actually access a backed-up data set directly on its backup disc. Tapes must be restored to disc before data base access is possible.

Media Labels

Regardless of how often you back up your data base and how careful you are about storing the media, the backup is useless if it is not properly identified. Here are some suggestions for labelling the media:

- Label the medium and not the container. If the tape reel or cartridge has a separate canister or box container, do not label the container unless both it and the medium bear the same serial number.
- Identify the backup format, especially if you are using your own backup program. If you do not identify the backup program name and revision, it may be impossible in later years to determine what the form of the data is.
- Identify the data base and what portion of the data base that the backup represents. This is especially important if you are performing a partial backup.
- Identify the schema. If you ever restructure the data base, the use of prior backups becomes more complicated. A good way to identify the data base structure is to back up a copy of the schema with the data base.

IMAGE/9000 Data Base Backup	
Date performed:	Performed by:
Utility/version:	System/version:
Data Base:	Schema version:
Source disc type:	Destroy date:
Files saved:	

Sample Data Base Backup Label

Media Rotation

You should use new media for backup operations. If this is not practical, you can re-cycle old backups or other “scratch” media provided you do so with caution.

- If you are re-using a medium, do not just purge the “scratch” data or files on it. Completely re-initialize the medium. This is especially important for heavily used flexible discs. The initialization process locates any questionable areas and makes them unavailable for data.
- If you are writing your own backup programs, verify the data. That is, as you write each record, back up, read it and compare it to the original data.
- If you find media errors, investigate. For example, on 9-track magnetic tape, if you see the drive reels reversing direction and re-trying, stop the operation, clean the tape heads and restart. If the problem occurs again, use a different tape.
- If you are re-cycling old backups, do not simply take the oldest backup. Never re-use the only backup. Establish a formula for re-using more and more of the older backups, but never use them all. For example, if you back up weekly, retain all of the current month’s backups, one for each month of the current year, one per quarter for the previous year and one per prior year.

Corrupt Data Base Recovery

The first step in recovering from a data base problem is to identify the extent of the problem. Is the data simply invalid, or is the data base structure corrupt as well? Does the problem affect only data sets, or is the root file corrupt too? You should also try to identify the cause of the problem. In some cases, such as a power failure, the cause is obvious. In others, the problem may be more subtle.

The next step in the recovery process depends on whether or not you have a backup. If you do not, or if you find that your backup is corrupt, special techniques are required. These are discussed at the end of this section.

Recovery From a Backup

Just as with the backup process, there are two levels at which you can perform recovery. You can recover just the data or the entire data base. The type of backup you performed affects your options, but you should start with a damage assessment.

Bad Data Recovery

If you discover that your data base simply contains bad data, the data base itself is probably not corrupt. Bad data usually results from errors in the source data, operator error or programming error.

You should start the recovery process by establishing an “audit trail” of the steps by which the data entered the data base. If you have a transaction log, you should examine it to ensure that correct data was entered to your program.

Once you have identified the point in the process or the point in time at which the problem occurred, you need to restore the latest valid backup, and re-enter the correct data since that time.

Entry-level Recovery

If your backup was performed at the entry or record level, you need to have an empty data set or data base to perform the recovery. If the structure of the data base is intact, you can simply erase the sets with DBERASE and run the recovery program.

If the data set file structure is also damaged, but the root file is undamaged, you can purge the sets with DBPURGE and re-create them with DBCREATE. In this case you do not need to erase them. Note, however, that DBPURGE/DBCREATE re-creates the sets with their defined capacities. If you had extended the original sets with DBEXTEND, you must extend the re-created sets.

If the root file is also damaged (or purged), you need to re-create the entire data base. If you cannot use DBPURGE to destroy the old sets, you may need to use a file system command (PURGE, rm, rmdir) to purge the files. When they are all purged, recreate the data base with SCHEMA or QUERY/DEFINE.

Set-level Recovery

If you saved the data base as a series of files, you must restore all of the sets and the root file. This generally requires that you purge the current (corrupt) data base. You do not re-create the data base with either SCHEMA or DBCREATE. The backup files are restored with their IMAGE structures intact.

Volume-level recovery

In general, volume recovery automatically destroys any old copy of whatever is being restored. It may also destroy other files. Before beginning a volume restore operation, you should make a file copy of every file on the disc which may have changed since the backup, for example, the transaction log.

Recovery Without a Backup

To recover a data base that has no backup requires that you first assess the damage. The recovery process is straightforward if only the data is corrupt. If the IMAGE pointers are corrupt, recovery is more difficult, but still possible. If the data set file structure or the root file is damaged, recovery may be impossible.

Bad Data Only

If the data base structure is intact, but your data is invalid, you need to write a program to either correct the data or delete and re-enter it.

Bad IMAGE Pointers

If a problem such as a disc hardware failure has corrupted your data set's chain pointers, you can still retrieve the data. You should write a program which opens the data base in DBOPEN mode 8. IMAGE returns a status error of 94, but the data base is still open.

Establish a destination data file for each manual master and detail data set. Open each data set and perform a DIRECTED DBGET for every record in the data set. If the record is occupied, write its data values to the destination file.

After saving all of the data, purge the data sets with DBPURGE. If the root file is not corrupt, re-create the data sets with DBCREATE. If the root file is corrupt, purge it as well and re-create the data base with SCHEMA or QUERY/DEFINE.

Re-open the new, empty, data base. For each manual master data set, open the saved data file and use DBPUT to re-enter the data. When all the manual masters are set up, use DBPUT to restore the data to all the detail sets. Save the backup program you have just written for future use.

Corrupt Root File or File System

If the data base root file is damaged, or the file structure of the data sets are damaged, you may not be able to recover any data. Consult your local HP Sales and Support office. Your local HP Systems Engineer (SE) may have software tools which provide access to data sets as ordinary data files or tools which provide access to physical tracks and sectors on the disc.

Purged Data Set Files

If your data base has been corrupted because one or more data set files have been purged by means other than DBPURGE, you must purge them from the root file before you can re-create them. To do this, use the DBPURGE intrinsic and include an "*" (asterisk) in the set list, for example:

```
DBPURGE ("DATA_BASE", SETS "NAME_SET, *;")
```

Data Base Restructure

There are generally two reasons for restructuring a data base. The first is because you have filled one or more data sets and need more capacity. The second is because your data needs have changed.

IMAGE provides a simple tool for increasing data set capacity. IMAGE does not provide tools for making other structural changes to the data base. These changes generally require saving the data from the data base, creating the new data base and restoring the data. The process is no more complicated than writing an entry-level backup and restore program.

Increasing Data Set Capacity - DBEXTEND

The root file of the data base records two capacity values for each data set: the defined capacity and the actual capacity. The defined capacity is the capacity specified during SCHEMA or QUERY/DEFINE. It is the capacity that the data set has after you create it with DBCREATE. The actual capacity of a set equals the defined capacity after a DBCREATE or DBPURGE/DBCREATE operation specifying that set.

You cannot change the defined capacity of a set without re-defining the data base. You can increase the actual capacity at any time with DBEXTEND. In DBEXTEND you specify the number of additional entries you need. DBEXTEND calculates the disc space required and allocates it from the file system. If successful, DBEXTEND adds the specified entry count to the current actual capacity of the data set.

The disc space allocated may not be contiguous to the original data set (or prior extensions). This can adversely affect performance, but is generally not noticable in detail data sets.

In master data sets, whether or not the extension is contiguous to the data set is not a major performance factor. What is significant is how full the original data set is. When you extend a master data set, the hashing algorithm does not change. If the set was full, each new entry hashes to an occupied entry (collision). The new entries are placed in the extension, but in every case they are on a synonym chain. Since each DBPUT to the master now requires a minimum of two accesses to different locations on the disc, performance can suffer dramatically. See the next chapter for more discussion of DBEXTEND and performance.

Changing the Schema

If you have the BASIC Language System, you can use QUERY to make many kinds of changes to the data base. If you do not, or if you need to make a change that QUERY does not support, you must write a program. The complexity of the program is directly related to the complexity of the change. This section divides data base changes into three areas: cosmetic, data structure and set structure.

Cosmetic Changes

Examples of cosmetic changes are changes to: passwords, names, RANGE and NULL values, and changing single-item master data set types. To change these attributes of the data base is a straightforward five step process. If you already have an entry-level backup program, you can probably use it with few changes.

1. Write a program to perform a SERIAL DBGET in each manual master and detail data set. Use full-record reads and write the data to a data file established for each data set. This is called "unloading" the data base.
2. Purge the data base.
3. Edit the old schema to reflect the changes.
4. Create the modified data base.
5. Write a program to read the saved data, and perform full-record DBPUTs to store it in each data set. This is called "loading" the data base. Load the master sets first.

If you change a manual master to an automatic master, you do not need to save the manual master's data. The new entries are stored automatically when you reload the detail sets. Conversely, if you are changing an automatic master to a manual master, you do need to unload the automatic master because you must "manually" load the new manual master.

Data Structure Changes

Examples of data changes are: changing an account number from I4 to R8, increasing the length of a string or adding an element to an array item.

The method for performing this type of restructure is similar to that used for cosmetic changes. The difference is that at some point in the process, you need to perform a type or data conversion. For example, when the load program reads the saved data, it builds a new full-record packing variable by extracting the old data values one by one from the old record. It simply copies those which did not change, and converts or extends those that did.

You can also use this method if you are simply adding one or more new items to the new data set. You can have the load program either acquire the new item values from some other source, or just ignore them, which causes IMAGE to store their NULL value in the entries.

Set Structure Changes

Examples of set structure changes are: moving a data item from one set to another, adding a new manual master and adding a new detail set. The procedure is conceptually the same: you unload the old data base, create the new one and load it with the old data.

In the case of significant structural changes, the suggested method of performing the unload is a little different. You should create a data file for each new data set. When you unload the old data base, access as many sets as necessary to save the data in the data files in the form required for the new structure.

If you are adding new items, have your unload program acquire their values and store them in the data files.

Data Base to Data Base

If you have ample disc space, you can reduce the unload/load task to a single program. Create the new data base without purging the old one, then write the unload/load program to simply copy the data from the old data base to the new one.

4-14 Data Base Maintenance

Chapter 5

Advanced Data Base Operations



Introduction

This chapter is concerned primarily with data base performance. It assumes that you are familiar with your host environment's file system and data base operations. This chapter discusses the choices you can make that affect data base performance, both at time of data base definition and during data base access. It also introduces two IMAGE intrinsics: DBBUFFER and DBCONTROL.

Data Base Definition Considerations

When you define your data base you must choose its defined capacities and its location in your mass storage system. Although both considerations affect performance, the location of the data base can have a far greater influence in a heavily used system.

Locating the Data Base

The objective in selecting a disc location is to minimize competition for the disc. Each access to a disc usually requires a seek operation, which positions the read/write head to the desired track, and a waiting period, known as the latency, during which the disc controller waits for the platter to rotate until the desired sector(s) arrive under the read/write head. These operations are transparent to your program.

You should also choose a high performance disc. Even a busy 7912P Disc Drive is likely to provide better performance than a dedicated 9895A flexible disc.

5-2 Advanced Data Base Operations

Multiple Volume Data Bases

If your data base resides on multiple volumes, you must provide IMAGE with information on how to find them. You do this in two ways:

- In the DIRECTORY part of the SCHEMA, you enter the volume label and path specifier of each volume in the data base.
- Before you access any required volumes in your programs, you must mount them. In the BASIC Language System, you must tell the system where the media is with a VOLUME DEVICES ARE statement. In the HP-UX System, use the mount intrinsic. You must mount all secondary volumes **before** using the data base or unexpected results occur.

You do not need to mount all of the volumes of a multi-volume data base at one time. You must mount only the sets explicitly or implicitly (linked) accessed by the current operations. The ROOT file must be mounted at all times while open.

Best performance is seen with single volume data bases. Multi-volume data bases have slower performance and could affect general HP-UX performance.

Master Data Set Capacities

The hashing algorithm used by IMAGE translates different key item values into a uniform distribution of record numbers. These record numbers are always within the range established by the defined capacity of the specified master data set.

It is possible for two different key item values to “hash” to the same record number. This is called a collision. The first key item to hash to the record number, the primary key, is stored there. The colliding key item values are called synonyms. The second and subsequent key item values, the secondaries, are stored in other records linked to the primary record by a synonym chain.

The performance implications of this are that collisions are time consuming and they have other, less obvious consequences. They are time consuming because they result in at least one extra disc access when they occur, and the same amount of extra access on every CALCULATED DBGET which specifies that secondary key value.

The less obvious consequence is that the secondary entries consume records which are now unavailable for use as primary records, thus increasing the chances that more secondaries will occur. A DBPUT specifying a key value which hashes to that secondary record number cannot use that record, yet the record is not even occupied by a synonym of the key item. This is a kind of “secondary collision”. It results from the fact that free records are drawn from the same area of the data set to which future key values may hash.

There are two ways to minimize the performance effects. You can reduce the chance of synonyms (primary collisions) and you can eliminate most secondary collisions altogether. You can eliminate the secondary collisions by moving the beginning of the free record chain out of the range of the hashing algorithm.

Minimizing Synonyms

The hashing algorithm IMAGE uses was chosen to provide a uniform distribution of record numbers from typical data values. It does this in a manner similar to the transformation of a “seed” value by a random number generator. It is not random, of course, because the same key item value always produces the same result. The algorithm operates in two steps. The first transforms the key value into a large double integer value, and the second evaluates it modulo C , where C is the data set capacity.

To obtain an even distribution of remainders (record numbers) after the modulo operation, C should be a prime number. There is a table of prime numbers, and a prime-finder program listing in Appendix T.

Any key item value has about an equal probability of hashing to any record in the set, so the chances of hashing to an occupied record are a function of the percentage of occupied records. In addition to choosing a prime number, you should choose the largest data set capacity you can afford.

Minimizing Secondary Collisions

In a newly created master data set, all of the records are on the free record chain. These records are also within the range of the hashing algorithm. You can move the initial free record chain out of the range of the hashing algorithm by extending the data set with DBEXTEND.

You can extend the master set at any time, but to have the desired effect, you must do so immediately after DBCREATE. When you extend a created and empty master set with DBEXTEND, the records in the extension are added to the head of the free record list. The hashing algorithm is not changed. It cannot generate the number of a record in the extension.

Collisions can still occur, of course, but in an extended set, the secondaries are placed in the extension. No records in the range of the hashing algorithm are used for secondary entries until the extension is filled or until the first DBDELETE. Primary collisions can still occur, but secondary collisions do not. DBDELETE places the deleted record at the head of the free record chain.

To use this technique, you should still set each master data set's defined capacity to be at least 133% of the anticipated number of entries. Create the data base and sets. Extend each master data set with DBEXTEND by about 20% of the estimated entry count. This minimizes both primary secondary collisions.

High Performance Access Methods

Regardless of where you locate your data base and how you establish the data set capacities, there are other measures you can take to improve the performance of your programs. These measures fall into two categories. The first is to minimize the amount of processing that the IMAGE intrinsic must do and the second is to minimize the amount of disc I/O.

Minimizing Processing

When you read or write a data entry, even though you only specify an item, IMAGE must always read or write the entire media record. IMAGE must set up a buffer for it, and set up tables for retrieving or storing your data value at the appropriate offset (field) in the record.

There are two steps you can take to minimize this processing in DBPUT, DBGET and DBUPDATE:

- Use full-record mode. This eliminates the extra code required to isolate the field for a specific item.
- Regardless of whether you use full-record or item-list mode, only specify the mode once. Thereafter, use current-list mode. This eliminates the overhead required to change the internal tables.

Minimizing ROOT File I/O

If your program makes frequent use of DBINFO modes 101, 107 and 108, you can reduce the amount of disc access to the ROOT file by keeping the sought information in memory. There are two ways to do this. The first and most obvious is to perform your DBINFO operations once at the start of your program, load all of the information into variables and arrays, and thereafter use the variables and arrays. For example, if your program needs to correlate item names and numbers, you can set up a string array of item names.

DBCONTROL Modes 3 and 4 (BASIC Only)

You can also have IMAGE load information into a system buffer in two cases. For both the PSEUDONYM and CODE tables, you can use DBCONTROL to load the tables in memory. DBCONTROL mode 3 allocates memory for the CODE table and DBCONTROL mode 5 allocates memory for the PSEUDONYM table. DBCONTROL modes 4 and 5 release those tables.

Once loaded, you do not access these tables as variables or arrays. You simply continue to use DBINFO modes 101, 107 and 108 as before. However, performance is improved because no disc access is required.

Minimizing Data Set I/O (BASIC Only)

To discuss how to reduce data set I/O, it is first necessary to explain how that I/O is performed. This discussion uses a DBGET as its example. When you access a data set for the first time, the host file system opens the set file and allocates one or more blocks of memory for I/O. These blocks are typically 512 bytes long. The system allocates two for a master data set and one for a detail data set.

Buffer Use in Reads (BASIC Only)

In the DBGET, you either specify a record number or imply one. The host file system uses the record number to compute the physical location on disc where that record is found. The file system transfers the disc data (sectors) at that location to the buffer, and then transfers the record data to IMAGE.

If the records in the set are small, the buffer may contain more than one record. If you are performing a SERIAL DBGET, the next DBGET may not require a disc access. The data sector which contains the record may already be in the buffer. If it is not, the file system performs another access to update the buffer.

As you might imagine, you can significantly reduce the disc accesses by making the buffer larger, at least for serial reads. For directed reads to arbitrary records, chained reads or calculated reads, the chances are reduced that the desired record is in the buffer. In fact, for non-serial reads, performance can actually be reduced because each disc access takes longer due to the larger buffer size. For random access reads, you should increase the number of buffers. For a sparsely occupied data set, you can set the buffer size to the media record size and the buffer count to the number of entries (occupied records).

If you make the buffer size equal to the data set size, the record must be in the buffer. For small data sets, this is a practical way to dramatically improve performance. The method for doing this is discussed in the section on "Controlling Buffer Size".

Buffer Use in Writes

When you write data to a data entry with DBPUT, the data is first written to the buffer and the buffer is then copied to the disc. If the original copy of the record is not in the buffer, the file system must first read the record from the disc to make the buffer current.

Note that the normal case is that the buffer is copied to the disc on each write operation. This is called immediate posting. When immediate posting is in effect, having a larger buffer actually degrades performance because more time is required to post the buffer to disc.

To improve I/O performance during write operations, you need to inform IMAGE that you only want the buffer posted when the desired record is not in the buffer. This is called deferred posting. When deferred posting is in effect, the buffer is only posted to disc when: the desired record is not in it, you request posting with DBCLOSE modes 2 and 4 or you close the data base entirely with DBCLOSE mode 1.

As with read operations, if you also set the buffer size to the size of the data set, the data may not be posted until you close the data set.

Controlling Buffer Posting

You can specify whether you want immediate or deferred posting during DBOPEN and you can change the posting mode with DBCONTROL. DBOPEN modes 1 and 3 open the data base with immediate posting in effect. DBOPEN mode 11¹ opens the data base with deferred posting in effect. Mode 11 is the same as mode 3 except for posting. There is no deferred version of mode 1. DBOPEN mode 8 opens the data base for read-only access, so there is no posting in that mode.

¹ BASIC only

5-6 Advanced Data Base Operations

After you open the data base, you can change the posting mode with `DBCNTROL`. `DBCNTROL` mode 1 enables deferred posting and `DBCNTROL` mode 2 disables deferred posting. `DBCNTROL` mode 2 causes any buffers containing modifications to be copied to the disc immediately.

Note

You must be cautious when using deferred posting. The performance improvements are obtained at the expense of placing your data at risk. If power fails or the computer is turned off while there are unposted buffers outstanding, the modifications do not get copied to disc and the data base may be corrupted.

To ensure that your data base is current, you should periodically force posting of the data set buffers with a `DBCLOSE` mode 2 or 4.

Controlling Buffer Size (BASIC Only)

In the BASIC Language System, you can directly control the size and number of I/O buffers with the `DBBUFFER` statement. This intrinsic is not available in the HP-UX operating system. Before you consider changing the buffer size, you should consult the `DBBUFFER` section of the BASIC Language Reference manual to determine if the default buffer sizes are adequate.

Since the `DBBUFFER` statement specifies a single data set, you can allocate different buffer sizes for each data set. There are some general rules you can use in determining optimum buffer sizes. Regardless of the size you choose, you should always add 255 bytes to the size. Records do not always align with sector boundaries on the disc. Adding an extra 255 bytes insures that the buffer is always long enough to accommodate the record(s).

A data set's buffer should be at least as large as the data set's media record length. You can compute the media record length from the information in Appendix T, obtain it with `DBINFO` mode 202 or obtain it from a CAT listing of the data base directory. If the buffer is less than the media record length plus 255 bytes, each record access may require two (or more) disc accesses.

If you are performing `SERIAL` reads primarily, increasing the buffer size beyond one record improves performance. If you are using other access methods, a multi-record buffer may not improve performance unless it contains a significant percentage of the records in the set. Use multiple buffers instead.

If you have available memory, the best performance is obtained by specifying a buffer size equal to the set size. You can compute the set size from the media record length and the capacity or from a CAT listing. To insure that the entire set is in the buffer, perform a `DBGET` mode 1 directed read to record one as the first operation.

Appendix B

BASIC Programming Language Considerations and Examples

Getting Started

If you have not installed your IMAGE software on your computer, refer to the Software Manual Catalog.

In the BASIC Language System the IMAGE data base intrinsics are BASIC language statements. To use these statements you must have loaded in your system one or more optional BIN files. The file "IMAGE_DBM" contains the system routines necessary to compile the IMAGE statements. If the data base does not reside on an internal disc, you must also load the BIN file containing the driver for the medium and interface. You may require one or more of the following BINs:

BIN File	Contains driver for:
HP9885	:HP9885
HPIB_FLEX	:HP82901, :HP82902, :HP8290X, :HP9895
MEMORY_VOL	:MEMORY
HP 27110	HP-IB interface driver



Data Base Item Types

The following table summarizes the correspondence between the IMAGE type designators, the terminology used in this manual, and BASIC variable types.

Designator	Description	BASIC Variable Type	Size in Bytes
C2	CODE number	INTEGER	2
I2	integer	INTEGER	2
I4	double integer	DOUBLE	4
R4	shortreal	SHORT	4
R8	longreal	REAL	8
Xn	string	String	n

Invoking the Statements

In general the BASIC run-time compiler checks your data base statements for correct syntax as you type in each line or load each line with a GET statement. The BASIC run-time support code also performs automatic numeric type conversions. For example, you can specify the DBOPEN mode as 1, 1.0, +1E+0, 1/1, or as a variable of any numeric data type.

There are several cases in which the BASIC system cannot discover a syntax error which can result in a run-time IMAGE status error. For example, the Status Array must be an array of type DOUBLE. Each IMAGE statement can determine whether it is or not, but only when the statement is executed. If the array is not type DOUBLE or has less than 10 elements, a BASIC status array error 210 results.

Parameters

In general, parameters which are string expressions can be simple string variables, substrings, literals or expressions. In BASIC, you do not need to terminate a string with a semicolon as long as the string is otherwise terminated by end of string or a blank.

List Specifier

The list specifier can denote full record, item list or current list modes. The item list can only be a list of names in BASIC. It cannot be an integer array of item numbers.

Software Revision Codes and Dates

The revision code of the IMAGE product described here is 01.00. It does not appear in the names of the other files; however, all of the files on the flexibles discs supplied with the product should have MODIFIED DATEs which match those in these tables. The revision "...R0100" appears in the volume labels.

The revision 01.00 IMAGE product should only be used with revision 01.00 of the BASIC Language System. If you "mix" revisions of IMAGE and BASIC software, BASIC programs may not execute correctly, if at all.

Software Installation

Flexible discs wear out after approximately 250 hours of disc rotation. The flexible discs supplied with your IMAGE product the distribution media, should be used only as the backup or archival copies of your software. You should make an on-line copy of your IMAGE software before using it. There are two ways to accomplish this:

- If you have an internal “sealed” or an external disc, you can copy the distribution media to it. You can copy the QUERY files and the example LIBRARY data base with the BUILD_QUERY program. BUILD_QUERY requires that the IMAGE option BIN file IMAGE_DBM and the BASIC option BIN file MS be loaded in your system. You can copy the remaining files with the BASIC COPY statement. Instructions for the BUILD_QUERY program are in one of the following sections.
- If the internal flexible disc is your only mass storage media, you can use the “DISC_BACKUP” utility supplied with your BASIC Language System software to copy each IMAGE disc to another flexible disc.

Note

You should copy the IMAGE software files only to a disc having the Structured Directory Format (SDF). Although you can copy QUERY and the utility programs to any format disc, you cannot copy the LIBRARY data base to a LIF or 9845 format disc.

Preparing for Daily use

Once you have an on-line copy of your IMAGE software, you need to consider how often you will be running data base programs, and how to most conveniently load the IMAGE_DBM file into the BASIC Language System.

To execute the QUERY, SCHEMA, DBXPRT, DBIMPORT, BUILD_QUERY programs and your own data base programs, IMAGE_DBM must be loaded in your system. There are four ways to accomplish this:

- If you run data base programs infrequently, you can load file IMAGE_DBM each time you require it. You use the BASIC LOADBIN command, for example:

```
LOADBIN "IMAGE_DBM:CS80,7"
```

- If your data base programs require other option BIN files, you can load the required files once with individual LOADBIN commands and then store the entire collection in a single file with a STOREBIN command, for example:

```
STOREBIN "MY_NAME<Password>/MY_BINS:CS80,7"
```

When you run your applications, you need LOADBIN only the composite file.

- If you run data base programs frequently, and have an adequate quantity of memory in your computer, consider storing the frequently required BINs in the system's “AUTOBN” file. That way, the BASIC system automatically loads the required BINs at power-up.
- As an alternative to using the “AUTOBN” file, you can also use the “BUILD_BOOT” utility program supplied with your BASIC system to configure the required BIN files in the “BOOT” area of the system disc. The system then powers-up with the required BIN options already present.

B-4 BASIC Programming Language Considerations and Examples

Installing QUERY and the LIBRARY Data Base

Once you have the IMAGE_DBM and MS BINs loaded, refer to the QUERY/9000 User's Guide for instructions for installing QUERY.

Accessing the Packing String

Variables in the BASIC language are stored in memory as values plus additional type and size information. Data in a data entry is stored only as data values. If you are using a packing string in a data base statement, it is also in simple value-only form.

To illustrate this, suppose you perform a DBGET using a packing string to obtain the values of the following data base items from a data entry.

Data Item and Type	Model (X6)	Quantity (I4)	Cost (R4)
-----------------------	------------	---------------	-----------

Data Entry

Bytes	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Data Values	0	2	0	0	1	C	0003481697				- 1.859467 + E00			

Packing Buffer Contents

Because it is already a string, you can use ordinary substring assignment to store the value of the Model item in a string variable, for example:

```
Model$=Packing_buf$[1,6]
```

The Quantity and Cost items are in two's-complement binary and floating point binary representation. You cannot use ordinary BASIC variable assignment to assign their values to a numeric variable because you cannot assign a substring to a numeric variable. Instead, BASIC has three statements specifically for this purpose: PACK USING, UNPACK USING and PACKFMT.

The PACKFMT Statement

The PACKFMT statement is referenced by both the PACK USING and UNPACK USING statements. It serves two purposes.

- PACKFMT specifies the program variables to which or from which packing string data is transferred.
- PACKFMT specifies which blocks of bytes within the packing string correspond to which program variables.

The exact syntax of the PACKFMT statement is described in the BASIC Language Reference manual. Essentially you specify a list of items of two kinds: program variables and skip factors. Each variable is matched to the next block of bytes in the packing string. The size of the block is determined by the variable type and is given in the table at the beginning of Appendix B. The skip factors specify the number of bytes to ignore in the packing string before matching the next variable.

Note

The variables specified in the PACKFMT statement must be the correct (same) type as the data in the packing string. The PACK USING and UNPACK USING statements perform no type conversion.

For the example packing string, an example PACKFMT statement is:

```
140  PACKFMT Model$,Quantity,Cost
```

The variables Quantity and Cost must have been previously dimensioned as DOUBLE and SHORT respectively.

If you obtain the value of a Model\$ using substring assignment, you can use a skip factor to specify just numeric data, for example:

```
140  PACKFMT 6X,Quantity,Cost      !SKIP Model$
```

Note

You must take care that the variables and skip factors specified in the PACKFMT statement match those variables on the correct byte block alignments. Failure to do so can have various symptoms. These effects are summarized in the BASIC Language Reference manual in the semantics of the PACKFMT statement.

The PACK USING and UNPACK USING Statements

The PACKFMT statement specifies the format of the packing string and the program variables to which it is matched. It does not specify the actual packing string involved or the direction of the data flow.

The PACK USING and UNPACK USING statements specify the packing string involved. PACK USING transfers data values from program variables to the packing string. UNPACK USING transfers values from the packing string to program variables. PACK USING and UNPACK USING must reference a PACKFMT statement.

For our example packing string, you can transfer the data values to program variables with the following statements:

```
130 UNPACK USING Fmt;Packing_string$           !Get data using
140 Fmt:PACKFMT Model$,Quantity,Cost           !this "map"
```

The PACK USING statement is the complement of the UNPACK USING statement. If you modify one or more of the variable values and want to write them back to the data base through the packing string, you can use the following statement:

```
210 PACK USING Fmt;Packing_string$           !Use same PACKFMT
```

Notice that both PACK USING and UNPACK USING can use the same PACKFMT statement.

It is not necessary to have a PACKFMT statement for every possible combination of packing string formats. You can transfer data for a single variable with one of four PACKFMT statements and the use of substring positioning. In the example, you can read either numeric value as follows:

```
350 PACKFMT Quantity                           !DOUBLE Quantity
360 PACKFMT Cost                               !SHORT Cost
370 UNPACK USING 360;Packing_string$[11,14] !Get SHORT value
380 UNPACK USING 350;Packing_string$[7,10]  !Get DOUBLE value
```

The IMAGERY and MOD_COUNT sample program segments following show further examples of packing string access.

BASIC Language Examples

The following pages show an example program using each of the IMAGE statements. Each statement is in a separate subprogram. These examples all report errors using an error reporting subprogram which is listed at the end of this Appendix. This subprogram uses the OUTPUT statement, and requires the option BIN file "IO" to be loaded in your system.

BASIC Main Program

```

10 !IMAGERY: Example BASIC MAIN Program for IMAGE/9000
20 !
30 ! Define global variables -----!
40 COM /Base/ Base$[196]           !For Data Base DIR specifier
50 COM /Log/ Log                   !Error list device specifier
60 COM /Prog/ Prog$[16]           !Name of this Program
70 COM /Password/ Password$[16]   !Data base Password
80 COM /Status/ DOUBLE Status(1:10) !IMAGE Status array
90 !
100 ! Define local/passed variables -----!
110 DIM Author$[25],Borrower_name$[25],Employee_Phone$[14]
120 DIM Librarian$[25],Phone_number$[14],Plant_address$(1:3)[30]
130 DIM Plant_name$[25],Pub$[30],Subject$[30],Title$[50]
140 DIM Record$[256]
150 INTEGER Copy_number,Location
160 SHORT Price
170 DOUBLE Call_no,Employee_no,Count
180 REAL Borrow_date,Pub_date
190 !
200 ! Initialize global constants -----!
210 PRINTER IS CRT
220 Prog$="IMAGERY"
230 Log=CRT
240 OUTPUT Log;CHR$(27)&"H"&CHR$(27)&"J" !Clear screen
250 Base$=" DBM/LIBRARY:CSB0,504"       !Change for your system
260 Password$="LIBRMGR"                 !Read/write in all sets
270 !
280 ! Demonstrate DBOPEN by opening the LIBRARY data base !
290 OUTPUT Log;Prog$&" : Opening data base"
300 CALL Open_lib                      !Open the data base
310 IF Status(1)=0 THEN                !If opened OK, then...
320 !
330 ! Demonstrate master DBPUT by adding a new subject -----!
340 Subject$="MAINTENANCE"
350 OUTPUT Log;Prog$&" : Adding SUBJECT"
360 CALL Add_subject(Subject$) !Add a new subject
370 !
380 ! Demonstrate detail DBPUT by adding a new book -----!
390 Title$="EVALUATING A SERVICE CONTRACT"
400 Call_no=1038955
410 Author$="SOAPSTONE,HEATHER"
420 Subject$="MAINTENANCE"
430 Pub_date=DATE("04 NOV 1982")
440 Pub$="BYTEWISE"
450 Price=13.95
460 Record;PACKFMT Title$,Call_no,Author$,Subject$,Pub_date,Pub$,Price
470 PACK USING Record;Record$
480 OUTPUT Log;Prog$&" : Adding BOOK"
490 CALL Add_book(Record$)
500 !
510 ! Demonstrate DBFIND and DBGET by finding same book -----!

```

B-8 BASIC Programming Language Considerations and Examples

```
520   OUTPUT Log;Prog$&": Finding CALL_NUMBER ";Call_no
530   CALL Find_book(Call_no,Record$)
540   IF Status(1)=0 THEN                !If book found....
550       UNPACK USING Title;Record$      !Extract title
560   Title:PACKFMT Title$
570       OUTPUT Log;Prog$&": ";Title$
580   ELSE
590       OUTPUT Log;Prog$&": Call number ";Call_no;" not found"
600   END IF
610 !
620 ! Demonstrate DBUPDATE by changing book's PRICE      !
-----!
630   OUTPUT Log;Prog$&": Changing the price of the book"
640   CALL New_price(Call_no,Price)
650 !
660 ! Demonstrate DBDELETE by deleting the new BOOK entry !
-----!
670   OUTPUT Log;Prog$&": Deleting the entry"
680   CALL Del_book(Call_no)
690 !
700 ! Demonstrate DBINFO by reading the modification count !
-----!
710   OUTPUT Log;Prog$&": Making a DBINFO request"
720   CALL Mod_count(Count)
730   OUTPUT Log;Prog$&": The modification count is ";Count
740 !
750 END IF
760 !
770 ! Demonstrate DBCLOSE by closing the LIBRARY data base !
-----!
780 Close:OFF ERROR
790   OUTPUT Log;Prog$&": Closing the data base"
800   CALL Close_lib
810 !
820 ! End of main program. Stop execution and disp. message !
-----!
830   PRINTER IS CRT
840   PRINT Prog$&": END"
850   END
```

DBOPEN Example

```

401001 SUB Open_lib                      !Open the LIBRARY data base
401002 !
401003 !This subroutine illustrates DBOPEN -----!
401004 !
401005 !Define global variables -----!
401006     COM /Base/ Base$[196]             !Data base specifier
401007     COM /Password/ Password$[16]      !Data base password
401008     COM /Prog/ Prog$[16]              !Main program name
401009     COM /Status/ DOUBLE Status(1:10)   !IMAGE Status array
401010 !
401011 !Define local variables -----!
401012     DOUBLE Open_mode
401013 !
401014 !Initialize local constants -----!
401015     Password$="LIBRMGR"                 !Global password
401016     Open_mode=3                         !Exclusive r/w access
401017 !
401018 !Attempt to open the data base -----!
401019     DBOPEN (Base$,Password$,Open_mode,Status(*))
401020 !
401021 !If open attempt fails, then... -----!
401022     IF Status(1)<>0 THEN
401023         CALL Db_error(Status(*),Base$,Prog$,401019,"Open_lib",CRT)
401024     END IF
401025 !
401026 !In any case, return to caller -----!
401027     SUBEXIT
401028 !
401029 SUBEND

```

B-10 BASIC Programming Language Considerations and Examples

DBCLOSE Example

```
403001 SUB Close_lib                !Close LIBRARY data base
403002 !
403003 !This subprogram illustrates DBCLOSE by closing the !
403004 !LIBRARY data base.-----!
403005 !
403006 !Define global variables-----!
403007     COM /Base/ Base$[196]          !Data base specifier
403008     COM /Prog/ Prog$[16]          !Main program name
403009     COM /Status/ DOUBLE Status(1:10) !IMAGE Status array
403010 !
403011 !Define local variables-----!
403012     DOUBLE Close_mode
403013     DOUBLE Dont_care              !Dummy parameter
403014 !
403015 !Initialize variables-----!
403016     Close_mode=1                  !Close entire data base
403017 !
403018 !Attempt to close the data base-----!
403019     DBCLOSE (Base$,Dont_care,Close_mode,Status(*))
403020 !
403021 !If close attempt fails, then report the problem...!
403022     IF Status(1)<>0 THEN
403023         CALL Db_Error(Status(*),Base$,Prog$,403019,"Close_lib",CRT)
403024         DISP "Press CONT to give up, or after problem solved."
403025         PAUSE                      !Allow operator to correct
403026         DISP ""                    !Clear display
403027     END IF
403028 !
403029 !Return to caller in any case.-----!
403030     SUBEXIT
403031 !
403032 SUBEND
```

DBPUT Examples

```

407001 SUB Add_subject(Subject$)                !Add a new SUBJECT
407002 !
407003 !This subprogram illustrates DBPUT in a manual master !
407004 !-----!
407005 !Define global variables                      !
407006     COM /Base/ Base$[196]                    !Data base specifier
407007     COM /Log/ Log                            !Error log device selector
407008     COM /Prog/ Prog$[16]                    !Main program name
407009     COM /Status/ DOUBLE Status(1:10) !IMAGE status array
407010 !
407011 !Define local variables                      !
407012     DOUBLE Put_mode
407013 !
407014 !Initialize local constants                  !
407015     Put_mode=1
407016 !
407017 !Attempt to add new SUBJECT to data set      !
407018     DBPUT (Base$,"SUBJECT",Put_mode,Status(*),"SUBJECT",Subject$)
407019 !
407020     SELECT Status(1)                        !Get the Condition word
407021     CASE 0                                !No error, no-op
407022     CASE 43                               !SUBJECT Already exists
407023         OUTPUT Log;"SUBJECT=";Subject$;" already exists,"
407024     CASE ELSE                             !Any other error
407025         CALL Db_error(Status(*),Base$,Prog$,407018,"Add_subject",Log)
407026     END SELECT
407027 !
407028 !Return to caller                          !
407029     SUBEXIT
407030 !
407031 SUBEND

```

```

407101 SUB Add_book(Record$)                    !Add a new BOOK
407102 !
407103 ! This subprogram illustrates DBPUT in a detail set !
407104 ! It uses full record mode and a packing string.
407105 !-----!
407106 ! Define global variables                      !
407107     COM /Base/ Base$[196]                    !Data base specifier
407108     COM /Log/ Log                            !Error log device selector
407109     COM /Prog/ Prog$[16]                    !Main program name
407110     COM /Status/ DOUBLE Status(1:10) !IMAGE status array
407111 !
407112 ! Define local variables                      !
407113     DOUBLE Put_mode
407114 !
407115 ! Attempt to add new SUBJECT to data set      !
407116     DBPUT (Base$,"BOOK",1,Status(*),"@",Record$)
407117 !
407118     IF Status(1)<>0 THEN                      !If put failed.....
407119         CALL Db_error(Status(*),Base$,Prog$,407116,"Add_book",Log)
407120     END IF
407121 !
407122 ! Return to caller                          !
407123     SUBEXIT
407124 !
407125 SUBEND

```

B-12 BASIC Programming Language Considerations and Examples

CHAINED DBGET example

```
404001 SUB Find_book(DOUBLE Call_number,Record$) !Find & get a book.
404002 !
404003 !This subprogram illustrates a DBFIND followed by a !
404004 !DBGET. It locates and returns the BOOK entry !
404005 !specified by the Call_number. !
404006 !
404007 !Define global variables !
404008 COM /Base/ Base$[196] !Data base specifier
404009 COM /Log/ Log !Error log device selector
404010 COM /Prog/ Prog$[16] !Main Program name
404011 COM /Status/ DOUBLE Status(1:10) !IMAGE status array
404012 !
404013 !Set up path to detail set BOOK !
404014 DBFIND (Base$,"BOOK",1,Status(*),"CALL_NUMBER",Call_number)
404015 !
404016 IF Status(1)=0 THEN !FIND OK, get entry
404017 DBGET (Base$,"BOOK",5,Status(*),"@",Record$,0)
404018 !
404019 IF Status(1)<>0 THEN !If GET failed, then...
404020 CALL Db_error(Status(*),Base$,Prog$,404017,"Find_book",Log)
404021 END IF
404022 !
404023 ELSE !If FIND failed,....
404024 CALL Db_error(Status(*),Base$,Prog$,404014,"Add_book",Log)
404025 !
404026 END IF
404027 !
404028 !Return to caller !
404029 SUBEXIT
404030 !
404031 SUBEND
```

DBUPDATE example

```

406001 SUB New_price(DOUBLE Call_number,SHORT Price) !Change a book price.
406002 !
406003 !This subprogram illustrates a DBUPDATE. It locates      !
406004 !and changes the entry specified by the Call_number.    !
406005 !
406006 !Define global variables-----!
406007     COM /Base/ Base$[196]           !Data base specifier
406008     COM /Log/ Log                   !Error log device selector
406009     COM /Prog/ Prog$[16]           !Main program name
406010     COM /Status/ DOUBLE Status(1:10) !IMAGE status array
406011 !
406012 !Define local variables-----!
406013     DIM Record$[156]
406014 !
406015 !Locate the entry-----!
406016     CALL Find_book(Call_number,Record$)
406017 !
406018     IF Status(1)=0 THEN               !If record found...
406019         DBUPDATE (Base$,"BOOK",1,Status(*),"PRICE",Price)
406020 !
406021     IF Status(1)<>0 THEN               !If GET failed, then...
406022         CALL Db_error(Status(*),Base$,Prog$,406019,"New_price",Log)
406023     END IF
406024 !
406025     END IF
406026 !
406027 !Return to caller-----!
406028     SUBEXIT
406029 !
406030 SUBEND

```

B-14 BASIC Programming Language Considerations and Examples

DBDELETE example

```
408001 SUB Del_book(DOUBLE Call_number)           !Delete a BOOK
408002 !
408003 !-----!This subprogram illustrates DBDELETE. It locates and!
408004 !deletes the BOOK entry specified by the Call_number. !
408005 !
408006 !Define global variables-----!
408007 COM /Base/ Base$[196]           !Data base specifier
408008 COM /Log/ Log           !Error log device selector
408009 COM /Prog/ Prog$[16]       !Main program name
408010 COM /Status/ DOUBLE Status(1:10) !IMAGE status array
408011 !
408012 !Define local variables-----!
408013 DIM Record$[156]
408014 !
408015 !Locate the entry-----!
408016 CALL Find_book(Call_number,Record$)
408017 !
408018 IF Status(1)=0 THEN           !If record found...
408019 !Book located; delete it.-----!
408020 DBDELETE (Base$,"BOOK",1,Status(*))
408021 !
408022 IF Status(1)<>0 THEN           !If DELETE failed, then...
408023 CALL Db_error(Status(*),Base$,Prog$,408020,"Del_book",Log)
408024 END IF
408025 !
408026 END IF
408027 !
408028 !Return to caller-----!
408029 SUBEXIT
408030 !
408031 SUBEND
```

DBINFO example

```

402001 SUB Mod_count(DOUBLE Count)      !Get DB modification counter
402002 !
402003 !This subprogram illustrates DBINFO mode 504. It      !
402004 !returns the number of modifications made to the      !
402005 !data base.-----!
402006 !
402007 !Define global variables-----!
402008     COM /Base/ Base$[196]              !Data base specifier
402009     COM /Log/ Log                      !Error log device selector
402010     COM /Prog/ Prog$[16]              !Main program name
402011     COM /Status/ DOUBLE Status(1:10)  !IMAGE status array
402012 !
402013 !Define local variables-----!
402014     DIM Record$[20]
402015 !
402016 !Request the modification count.-----!
402017     DBINFO (Base$,0,504,Status(*),Record$)
402018 !
402019     IF Status(1)=0 THEN                !If DBINFO ok...
402020         UNPACK USING Count;Record$    !Get mod count
402021 Count:PACKFMT Count
402022 !
402023     ELSE                                !DBINFO failed...
402024         CALL Db_error(Status(*),Base$,Prog$,402017,"Mod_count",Log)
402025 !
402026     END IF
402027 !
402028 !Return to caller-----!
402029     SUBEXIT
402030 !
402031 SUBEND

```

Error Reporting Subprogram

```

400001 SUB Db_error(DOUBLE Stat(*),OPTIONAL Base$,Prog$,Line,Module$,Log$)
400002 !
400003 !Document passed parameters-----!
400004     OPTION BASE 1
400005 ! DOUBLE Stat(1:10)      !IMAGE Status array {required}
400006 ! DIM Base$[196]         !Data base specifier variable
400007 ! DIM Prog$[16]          !MAIN program name
400008 ! REAL Line              !Line number of failed DBxxxx call
400009 ! DIM Module$[16]        !Subprogram of failed DBxxxx call
400010 ! REAL Log               !Error message destination device sel.
400011 !-----!
400012 ! This subprogram outputs a report of an IMAGE data base error
400013 ! to the specified or defaulted Log device. Report form is:
400014 ! -----
400015 ! Program_name: An IMAGE Status error 999 has occurred
400016 ! in data base: Data_base_name:msus
400017 ! during a DBxxxx statement in line 123456 of module Sub_name
400018 ! The BASIC error is 999 in data set number 63
400019 ! -----
400020 !
400021 !Define and initialize local variables-----!
400022     DATA ????,OPEN,INFO,CLOSE,FIND,GET,UPDATE,PUT
400023     DATA DELETE,LOCK,UNLOCK,CONTROL,BUFFER
400024     DIM Intrinsic$(400:412)      !For DBxxxxx names
400025     REAL Condition,Intrinsic,Set,Basic,Dest
400026     READ Intrinsic$(*)          !Load names now
400027 !
400028 !Get data of interest from Status array-----!
400029     Condition=Stat(1)           !Condition word
400030     Intrinsic=Stat(6)           !DBxxxx call number
400031     IF (Intrinsic<401) OR (Intrinsic>412) THEN Intrinsic=400
400032     Set=Stat(8)                 !Set number, if any
400033     Basic=Stat(10)              !BASIC error, if any
400034 !
400035 !If "Log" device parameter was defaulted, use CRT-----!
400036     Dest=CRT
400037     IF NPAR>=6 THEN Dest=Log
400038 !
400039 !Begin output of error message-----!
400040     OUTPUT Dest USING Bars;CHR$(7)      !-----BEEP
400041 Bars:IMAGE 50("-"),A
400042     IF NPAR>2 THEN OUTPUT Dest;Prog$;": ";
400043     OUTPUT Dest;"An IMAGE Status error ";Condition;
400044     OUTPUT Dest;" has occurred"
400045     IF NPAR>1 THEN OUTPUT Dest;"in data base: ";Base$
400046     OUTPUT Dest;"during a DB";Intrinsic$(Intrinsic);" statement";
400047     IF NPAR>3 THEN OUTPUT Dest;" in line";Line;
400048     IF NPAR>4 THEN OUTPUT Dest;" of module ";Module$;
400049     OUTPUT Dest;"
400050     IF Condition=-1 THEN OUTPUT Dest USING Basic;Basic,Set
400051 Basic:IMAGE "The BASIC error is ",ZDD," in data set number ",ZD
400052     OUTPUT Dest USING Bars;CHR$(7)
400053 !
400054 !Return to caller-----!
400055     SUBEXIT
400056 SUBEND

```

Appendix C

C Programming Language Considerations and Examples

Data Base Item Types

The following table summarizes the correspondence between the IMAGE type designators, the terminology used in this manual and the C variable types.

Designator	Description	C variable type	Size in bytes
I2	16-bit integer	short	2
I4	32-bit integer	int	4
R4	32-bit real	float	4
R8	64-bit real	double	8
Xn	character string	char	n

C Library Calls

You do not need to predeclare calls to IMAGE-UX from the C language as externals.

You can identify the actual IMAGE-UX library routines (contained in */usr/lib/libim.a*) at compile time using the **-l** option. For example:

```
cc c/mysource.c -o bin/myobject -lim
```

Parameters

All parameters to IMAGE-UX must be call-by-reference; parameters must point to the actual data. For character strings and arrays, this is automatic. For simple numeric variables, the ampersand (&) is required. So, you could code a *dbopen* call as follows:

```
char base[60];           /* data base path name */
short mode;              /* dbopen mode parameter */
short status[10];        /* return status array - goes */
                          /* from elt 0 to elt 9 */
strcpy (base, " dbm/dbname"); /* remember leading 2 blanks */
mode = 3;                 /* exclusive dbopen */
dbopen (base, "mypassword", &mode, status);
if (status[0])

/* Print error message */
```

C Language Example

```

/*  browse - Browse through a data set.
 *
 *  Usage: browse dbpath dset [password]
 *  eg: browse dbm/dksam mymaster mypass
 */

union u_tag {
    short sval;    /* 16 bit integer */
    int ival;     /* 32 bit integer */
    float fval;    /* 32 bit float */
    double dval;   /* 64 bit float */
    char strg[8] } uval;

#include <stdio.h>
#include <errno.h>
extern int errno, errno;
#define LINESIZE 300
#define BASESIZE 300
char base[BASESIZE];
short mode, s[10];

main (argc, argv)
int  argc;
char *argv[];
{
    char pass[LINESIZE], more[LINESIZE], dset[LINESIZE];
    int  i;

    if (argc < 3 || argc > 4)
        fprintf (stderr, "\nUsage:  browse dbpath dset [password]\n");
    else {
        fprintf (stderr, "\n\nBrowse - browse through a data set\n\n");
        (argc == 2) ? strcpy (pass, ";") : strcpy (pass, argv[3]);

```

Dbopen Example

```

    strcpy (base, " ");
    strcpy (base+2, argv[1]);
    mode = 3;
    dbopen (base, pass, &mode, s);
    if (s[0]) {
        db_msg ("*** Error in dbopen: ", s);
        exit (1);
    }

    strcpy (dset, argv[2]);          /* Now loop on sets. */
    more[0] = 'y';
    while (more[0] == 'y' || more[0] == 'Y') {
        if (browse_set (base, dset))
            exit (1);
        fprintf (stderr, "Browse through another set? (y/n) ");
        gets (more);
        if (more[0] == 'y' || more[0] == 'Y') {
            fprintf (stderr, "Enter set name: ");
            gets (dset);
            fprintf (stderr, "\n");
        }
    }

    mode = 1;                        /*** All done. Close data base. ***/
    dbclose (base, pass, &mode, s);
    if (s[0])
        db_msg ("*** Error in dbclose: ", s);
    } /* else */
} /* main */

```

```

browse_set (base, dset)
char base[], dset[];
{
    short mode, s[10], fieldcnt;
    int    arg;
    char   buf104[300], buf105[1000], getbuf[2000], more[80];

```

Dbinfo Example

```

mode = 104;
dbinfo (base, dset, &mode, s, buf104);
if (s[0]) {
    db_msg ("*** Error in dbinfo mode 104: ", s);
    return (-1);
}
set_up_num (&fieldcnt, buf104, 0, 2);

mode = 105;
dbinfo (base, dset, &mode, s, buf105);
if (s[0]) {
    db_msg ("*** Error in dbinfo mode 105: ", s);
    return (-1);
}

```

Serial Dbget

```

s[0] = 0;
more[0] = 'y';
while (s[0] == 0 && more[0] != 'q') {
    mode = 2;
    dbset (base, dset, &mode, s, "@", getbuf, &mode);
    if (s[0] == 11) {
        fprintf (stderr, "*** End of file \n");
        mode = 4; arg = 0;
        dbset (base, dset, &mode, s, "@", getbuf, &arg);
        if (s[0]) {
            db_msg ("*** Error rewinding dset: ", s);
            return (-1);
        }
        mode = 2; /* Close set */
        dbclose (base, dset, &mode, s);
        if (s[0]) {
            db_msg ("*** Error closing dset: ", s);
            return (-1);
        }
        more[0] = 'q';
    }
    else
        if (s[0]) {
            db_msg ("*** Error in dbset: ", s);
            return (-1);
        }
    else { /* Got a valid record; go on */
        if (ertrec (base, s, fieldcnt, buf104, buf105, getbuf) == -1)
            return (-1);

        fprintf (stderr, "Press 'return' for more; Enter 'q' to quit: ");
        gets (more);
        fprintf (stderr, "\n");
        if (more[0] == 0) more[0] = 'y';
    } /* else */
} /* while loop */
} /* function browse_set */

```

Buffer Unpacking Routine

```

set_up_num (puval,buf,index,length)

/* Dbset returns a buffer. Now how do I get my numbers out? This routine */
/* will help. You must define a union externally first. Then:          */
/* INPUT:  puval   - ptr to union variable                             */
/*         buf     - buffer variable from dbset                        */
/*         index   - first byte to look at                             */
/*         length  - size of number                                     */
/* OUTPUT:  puval^ contains the number                                  */

union u_tag *puval;
char buf[];
{
    int i;
    for (i=0; i<length; ++i)
        puval->strs[i] = buf[index+i];
}

```

Print Error Message Routine

```

db_msg (buf,pstatus)

/* This function prints error messages from IMAGE-UX.                  */
/* Buf is a string containing specific error info; pstatus is a      */
/* ptr to the status array. For example, if you get a non-zero      */
/* return from dbset, you might call this function as follows:      */
/*         if (status[0])                                           */
/*             db_msg ("Error in dbset: ",pstatus);                */

char   buf[];
short  *pstatus;
{
    int   i;
    fprintf (stderr, "\n%-3s",buf);
    for (i=0; i<10; ++i)
        fprintf (stderr, "%hd ",*(pstatus+i));
    fprintf (stderr, "\n");
    perrorinfo();
    fprintf (stderr, "\n");
}

perrorinfo() /* Needs #include <errno.h> & #include <stdio.h> */
{
    fprintf (stderr, "*** Errno = %d   Errinfo = %d\n", errno, errinfo);
}

```

Buffer Pack Routine

```

set_buffer (puval, buf, index, length)

/* How do you put numbers into a buffer for dbput? This function will */
/* help.                                                                */
/* INPUT:  puval   - ptr to union uval                                */
/*         buf     - buffer for dbput                                  */
/*         index   - starting location in buffer                      */
/*         length  - size of number                                     */
/* For example, to put a 4-byte integer into the 6th thru 9th chars of */
/* buffer, do the following:                                             */
/*         uval.ival = 6;                                              */
/*         set_buffer (&uval,buffer,6,4);                             */
/* NOTE: buffer starts at 0 not 1.                                       */

```

```

union u_tag *puval;
char buf[];
{
    int i;
    for (i=0; i<length; ++i)
        buf[index+i] = puval->strg[i];
}

```

Print Element of Buffer Routine

```

Prtelt (elcnt, itemType, elemLen, offset, getbuf)

/* This routine prints out an element from a dbget buffer. (The element */
/* may actually be an array of elements. */
/* */
/* input: elcnt      - 1 if not an array */
/*          itemType - item type of element to be printed */
/*          elemLen   - # of bytes per element (if array, each entry has */
/*                    this # of bytes */
/*          offset    - byte offset (0 based) to element */
/*          getbuf    - buffer obtained from successful dbget */
/* */
char itemType[], getbuf[];
int elcnt, elemLen, offset;
{
    int i;

    if (elemLen < 9) set_up_num (&uval, getbuf, offset, elemLen);
    switch (itemType[0]) {
        case 'C':
        case 'I': if (elemLen == 2) {
                    printf ("%hd\n", uval.sval);
                }
                else
                    if (elemLen == 4) {
                        printf ("%d\n", uval.ival);
                    }
                    else {
                        fprintf (stderr, "Err in switch of Prtelt; I;
elemLen=%d\n", elemLen);
                        return (-1);
                    }
                break;
        case 'R': if (elemLen == 4) {
                    printf ("%f\n", uval.fval);
                }
                else
                    if (elemLen == 8) {
                        printf ("%lf\n", uval.dval);
                    }
                    else {
                        fprintf (stderr, "Err in switch of Prtelt; R;
elemLen=%d\n", elemLen);
                        return (-1);
                    }
                break;
        case 'X': for (i=0; i<elemLen; ++i)
                    printf ("%1s", getbuf+offset+i);
                    printf ("\n");
                    break;
        default: fprintf (stderr, "Err in switch in Prtelt: Unknown itemType: ");
                    fprintf (stderr, "%1s\n", itemType);
                    break;
    } /* switch */
} /* function Prtelt */

```

Terminal Conversion Routine

```

getvalue (value, itemType, elemLen, tmpBuf)

/* You can use this routine to convert terminal input values for */
/* an element to their correct numeric type.                      */
/*                                                                  */
/* Input: tmpBuf    - input string from terminal                  */
/*        itemType - item type of the element                   */
/*        elemLen   - # of bytes in element (if array, # of bytes in */
/*                  each element of array                         */
/* Output: value    - Ptr to desired return variable             */
/*                                                                  */
/*                                                                  */

char value[], itemType[], tmpBuf[];
int elemLen;
{
    int i;
    switch (itemType[0]) {
        case 'C':
        case 'I': if (elemLen == 2) {
                    sscanf (value, "%hd", tmpBuf);
                }
                else
                    if (elemLen == 4) {
                        sscanf (value, "%d", tmpBuf);
                    }
                else {
                    fprintf (stderr, "Err in switch of getvalue; I;
                                     elemLen=%d\n", elemLen);
                    return (-1);
                }
                break;
        case 'R': if (elemLen == 4) {
                    sscanf (value, "%f", tmpBuf);
                }
                else
                    if (elemLen == 8) {
                        sscanf (value, "%lf", tmpBuf);
                    }
                else {
                    fprintf (stderr, "Err in switch of getvalue; R;
                                     elemLen=%d\n", elemLen);
                    return (-1);
                }
                break;
        case 'X': for (i=0; i<elemLen; ++i)
                    tmpBuf[i] = ' ';
                    strcpy (tmpBuf, value);
                    tmpBuf[strlen(tmpBuf)] = ' ';
                    break;
        default: fprintf (stderr, "Err in switch in getvalue: Unknown itemType: ");
                  fprintf (stderr, "%1s\n", itemType);
                  break;
    } /* switch */
} /* function getvalue */

```

Print Data Base Entry Routine

```

prtrec (base, s, fieldcnt, buf104, buf105, getbuf)
short fieldcnt, s[];
char base[], buf104[], buf105[], getbuf[];
{
    int    recno, element, elemllen, offset, i, j;
    char   buf102[26];
    short  itemno;
    short  mode, *ps = &s[0];

    /* Get record # */
    set_up_num (&recno, s, 4, 4);
    printf ("Entry # %d\n", recno);

    /* Print fields */
    for (i=1; i<=fieldcnt; ++i) {
        set_up_num (&itemno, buf104+(i*2), 0, 2);
        if (itemno < 0) itemno *= (-1);
        mode = 102;
        dbinfo (base, &itemno, &mode, s, buf102);
        if (s[0]) {
            db_msg ("*** Error in dbinfo mode 102: ", ps);
            return (-1);
        }
        printf ("%16.16s", buf102);
        set_up_num (&offset, buf105, 8*(i-1)+8, 4);
        set_up_num (&elemllen, buf102, 20, 4);
        set_up_num (&element, buf102, 24, 4);

        for (j=1; j<=element; ++j) {
            if (element>1 && j==1) printf (" (%d): ", j);
            if (element>1 && j>1) printf ("          (%d): ", j);
            if (element == 1) printf ("          ");
            if (prtelt (j, buf102+16, elemllen, offset+((j-1)*elemllen), getbuf))
                return (-1);
        } /* for (j=1; ... */
    } /* i loop */

    printf ("\n");
    fflush (stdout);
} /* prtrec */

```




Appendix E

Errors

IMAGE Status Errors

These errors are returned by IMAGE intrinsics in element one of the Status array, the Condition word.

Result	Meaning
- 135	DBLOCK with wait not allowed while your partition/process already has lock(s) in effect
- 94	Corrupt data base, one or more data sets must be erased
- 93	Corrupt root file, purge and re-define/create the data base
- 92	One or more data sets are purged or not created
- 91	Root file structure not compatible with this revision of IMAGE
- 75	IMAGE-UX for single user systems only
- 74	Data base name mismatch
- 52	List specifier is invalid or item specified is not a key or search item in the specified data set
- 32	Data base already open in a mode incompatible with your DBOPEN
- 31	Improper mode parameter
- 24	DBPUT, DBDELETE or DBUPDATE not allowed on an Automatic master data set
- 23	Write access is required and your DBOPEN password does not have it in this data set
- 22	Specified set must be, and is not, a detail data set
- 21	Improper or nonexistent data set, data item, password or volume specified
- 14	DBPUT, DBDELETE and DBUPDATE not allowed in DBOPEN mode 8
- 12	Not all necessary data sets are locked
- 11	Data base variable invalid; leading blanks missing or no such data base
- 10	No data base number available; 100 DBOPENs already active
- 1	Host environment error, see Status(10) for system error
0	Successful execution - no error
10	Beginning of file (data set) encountered
11	End of file (data set) encountered
12	Negative record number specified, must be zero or positive
13	Specified record number greater than current set capacity
14	Beginning of chain encountered
15	End of chain encountered
16	The data set is full
17	No current record, no chain head, or specified record is empty
18	Broken chain encountered
20	One or more requested data sets locked to another data base variable
41	DBUPDATE of key, search or sort item not allowed
43	Specified key item value already exists in master data set
44	DBDELETE in master not allowed, one or more detail chains to this entry are not empty
50	Packing variable or string variable is smaller than the data item size(s)
51	Variable size and/or type does not match the data base item
52	The number of variables in the list does not match the requested number of variables in the list specifier
53	Argument parameter type or size incompatible with the key item type or size
94	DBOPEN successful, however, the data base is corrupt and the DBOPEN mode has been forced to read-only
95	Intrinsic not supported
1nn	There is no chain head for path nn
3nn	The Automatic master for path nn is full

IMAGE-Related Errors

The following is a partial list of BASIC errors which are related to IMAGE data base operations. Most of these errors are found in Status array element 10 when element 1, the condition word, is -1. BASIC error 2 may not be returned. It can cause a BASIC pause or ON ERROR branch. BASIC error 210 always causes an error pause or ON ERROR branch.

In data base utility statements, these errors are returned in the RETURN expression.

Result	Meaning
1	Missing OPTION BIN or configuration error
2	Memory overflow
8	Improper parameter matching
9	Improper number of parameters
62	Path to data base is protected
73	Incorrect device type in msus
80	Medium not present, off line or door open
81	Device and/or interface hardware failure
82	Device and/or interface hardware not present
88	Read data error
90	Mass storage system error
208	Volume not mounted
209	SDF directory format required
210	Bad status array, is not double integer or has less than 10 elements
213	Data base directory not found
214	One or more data sets purged or not created
215	One or more data sets are now corrupt
216	No available data base number, 100 DBOPENs currently active
217	Data base definition incomplete (corrupt root file)
220	Improper or illegal use of maintenance word
221	Data set purged or not created
222	Data base directory not created
225	Root file structure/contents not compatible with this revision of IMAGE
226	Corrupt root file - purge and re-define/create it
227	Corrupt data base - one or more data sets must be erased
229	Data base open, utility operation not allowed at this time
230	Improper set list or duplicate sets in the set list
231	Improper record count
232	Cannot purge root file with one or more sets unpurged
233	Root file not found
234	Referenced line is not a PACKFMT statement
236	Packing string too short for requested data

SCHEMA and QUERY/DEFINE Errors

The following are the error messages possible during the execution of SCHEMA or the DEFINE module of QUERY.

General Error Messages

ERROR nnn [:extended error message] – This standard BASIC error message can occur if improper information is input for the text file specifier, root file specifier or printer select code. It can also occur later in the process if an ERROR 2 occurs. The [:extended error message] is included only if the local language option BIN file, such as "ERRORS_ENG", is loaded.

<...> EXPECTED – The preceding statements or punctuation lead the processor to expect the syntax element <...> to appear at this point. The element actually found is therefore out of sequence. This event increments the error count, but the unexpected element is ignored and processing can continue.

<...> EXPECTED. (FATAL ERROR) – As above, except that the processor is unable to continue, regardless of the error count and limit.

*** FATAL ERROR nnn SCHEMA PROCESSOR TERMINATED. – This message is printed when any error occurs which prevents the processor from continuing, regardless of whether the error count has reached the value specified in the ERRORS = instruction.

<...> IN CHECKING STRUCTURE – The error reported with this message occurred during post-processing of the root file definition before actually writing the file contents to disc.

ROOT FILE HAS NOT BEEN GENERATED. – This message is printed whenever the processor has encountered one or more errors or you have included the schema instruction NOROOT.

*** TOO MANY ERRORS. PROCESSOR TERMINATED. – Whenever the number of errors exceeds the number you specify in the ERRORS = expression, the processor halts. The data base is not created.

Specific Error Messages

A VALID INTEGER EXPECTED – One or more of the items in the expression is either not an integer or is outside the allowable range of values.

A VALID REAL EXPECTED – One or more of the items in the expression is either not a real number or is outside the allowable range of values.

A VALID STRING EXPECTED – One or more of the items in the expression is either not in quotes or is too long.

BAD ERROR NUMBER SPECIFIED, MUST BE IN RANGE (0,999) – The maximum number of errors you can specify in the ERRORS = expression must be in the range 0 through 999.

BAD ITEM LENGTH – The length of a string item must be in the range 1 through 2 147 483 647.

E-4 Errors

BAD ITEM PICTURE – The specified PICTURE word must be an integer in the range –2 147 483 648 through 2 147 483 647, and if >0, should match one of the documented QUERY picture words for that data type. (See Appendix Q).

BAD LINE NUMBER SPECIFIED, MUST BE IN RANGE (20,999) – The number of lines per page in the LINES = instruction must be in the range 20 through 999.

BAD SET CAPACITY – The defined number of entries for a data set, specified by the CAPACITY: statement, must be an integer in the range 1 through 2 147 483 647. There must be room on the data set medium for this number of entries.

BASE VOLUME MUST NOT BE SPECIFIED HERE – Specify a directory name in the SET Part only if that set resides on a different volume than the root file.

BEGIN DATA BASE EXPECTED – The first schema statement other than a schema instruction must be the start of the DATA BASE Part.

CAPACITY EXPECTED – The definition of each data set must end with a CAPACITY: statement.

CODES > MAX # CODES/BASE – The maximum number of CODE item values in a single data base is 255.

CODES > MAX CODES/ITEM – The maximum number of CODE item values for a single data item is 32.

DUPLICATE CODE – Each code value for a CODE (C2) item must be a unique string.

DUPLICATE FIELD ITEM – You cannot specify the same item name more than once in the definition of a single data set.

DUPLICATE ITEM NAME – The item names in the ITEM PART must be unique.

DUPLICATE PASSWORD – The passwords in the PASSWORD PART must be unique.

DUPLICATE PSEUDO – Each pseudonym name in the data base must be unique and cannot be duplicated for the same or different items.

DUPLICATE SECONDARY VOLUME – Each directory name in the DIRECTORY PART must be unique. If you must use two existing volumes which presently have the same name, rename one of them.

DUPLICATE SET NAME – Each data set name in the SET PART must be unique.

END EXPECTED – After defining the last data set, the schema statement END must appear before the end of the text file.

ENTRY EXPECTED – After defining the NAME, location and access list for a data set, the next schema statement must be the ENTRY: statement to define the set's data entry.

ERROR <nnn> IN MAKING ROOT FILE – System error <nnn> occurred while constructing the root file. If possible, re-initialize the medium and try again. This error halts the processor. Typical errors in the BASIC Language System are: 54 (duplicate root file directory) and 64 (insufficient space on medium).

FIELDS > MAX # FIELDS ALLOWED – The maximum number of items in a data set is 1023.

GLOBAL PASSWORD NOT ACCESSED – If you specify a user class access list for any data set, you must specify at least the global password number (user class 1) in the write access list.

ILLEGAL ARRAY – Array items can have only 1 to 6 dimensions. The bounds of each dimension must be specified by integers in the range of a double integer. The difference between the upper and lower bounds must be less than or equal to 2 147 483 647.

ILLEGAL BASE NAME – The name selected for an IMAGE/9000 data base must be: an IMAGE name, not a reserved word, and a valid file name in the host operating system.

ILLEGAL CHARACTER ENCOUNTERED & IGNORED – There is a character in the schema text file which is not a valid component of schema language. If it is not visible, try listing the schema with the display or printer's DISPLAY FUNCTIONS enabled.

ILLEGAL CODE – A code value must be a quoted string consisting of 1 to 16 characters.

ILLEGAL CODE NUMBER – You can define only 32 unique code values for each CODE item.

ILLEGAL DEFINED TITLE – The string specified in the \$TITLE = schema instruction should be less than 60 characters long. The closing quote should appear on the same line as the instruction, and should appear before column 160. Quotes within the title string must appear as double quotes.

ILLEGAL FIELD NUMBER – An automatic master data set can contain only a single key item.

ILLEGAL IDENTIFIER – An identifier must be an IMAGE name and not a reserved word.

ILLEGAL ITEM – During the SET Part this error means that one or more referenced items are not defined in the ITEM Part. After the SET Part this error means that one or more items defined in the ITEM Part have not been referenced (used) in any set defined in the SET Part.

ILLEGAL ITEM NAME – The item name must be an IMAGE name (See the Glossary) and must not be a reserved word.

ILLEGAL ITEM TYPE – The type of a data base item must be C2, I2, I4, R4, R8 or X<n>, where <n> is an integer in the range 1 through 2 147 483 647.

ILLEGAL KEY ITEM – A key item must be a scalar (non-array) item. If it is a string item, it must have a length of 256 bytes or less. If referencing a key item in a detail data set, the search item must be the same type and size as the key item.

E-6 Errors

ILLEGAL LEX. ORDER – The **LEXICAL ORDER =** statement must specify one of the following lexical orders: ASCII, FRENCH, GERMAN, SPANISH, SWEDISH. Use ASCII for KATA-KANA.

ILLEGAL PASSWORD – A password must be a unique **IMAGE** name.

ILLEGAL PASSWORD ACCESS – Each password must have access to at least one set.

ILLEGAL PASSWORD NUMBER – A password number (user class) defined in the **PASSWORD** Part must be integer in the range 0 through 13.

ILLEGAL PATH NUMBER – The number of paths specified for a key item in a master set must be an integer in the range 1 through 31.

ILLEGAL PSEUDO – An item pseudonym name must be a unique **IMAGE** name. It must not be the same as any other pseudonym or item name.

ILLEGAL PSEUDO NUMBER – In **QUERY/DEFINE**, you must define at least one pseudonym name if you press the **PSEUDO** softkey.

ILLEGAL RANGE – The **RANGE =** expressing must specify a minimum and maximum value which are within the range of the type of the data item. An item's **NULL** value must be within this range. The **RANGE** of a search item must be equal to or within the **RANGE** of the key item to which it is linked.

ILLEGAL READ/WRITE LIST – A data set's access list must consist of zero, the global password number (1) or other password numbers defined in the **PASSWORD** Part.

ILLEGAL SEARCH ITEM – A detail set search item must be a scalar item (non-array) and if a string item, must have a length of 256 bytes or less. It must match its related key item in type and size.

ILLEGAL SET NAME – A data set name must be: unique **IMAGE** name, not a reserved word and a valid file name in the host operating system.

ILLEGAL SET TYPE – A data set type must be one of the following: A, AUTOMATIC, D, DETAIL, M, or MASTER. Other spellings and abbreviations are not allowed.

ILLEGAL SORT ITEM – A sort item must be a scalar (non-array) and if a string item, must have a length of 256 bytes or less.

ILLEGAL VOLUME NUMBER – The directory name specified for this set is not defined in the **DIRECTORIES** Part.

INVALID SCHEMA INSTRUCTION – A schema instruction must conform to the syntax diagrams of chapter 2 and must appear on a separate text line.

ITEM HAS ARRAY – You can specify the **DIMENSION =** expression only once for each data item.

ITEM HAS CODE – You can specify the CODE = expression only once for each CODE data item.

ITEM HAS NO CODE – For a CODE item, you must specify at least one code value.

ITEM HAS PSEUDO – You can specify the PSEUDONYM = expression only once for each data item.

ITEM HAS RANGE – You can specify the RANGE = expression only once for each data item.

ITEM NOT CODE – You can specify code values only for CODE (C2) items.

ITEMS EXPECTED – The next schema statement after the last semicolon in the PASSWORD Part must be an ITEMS: statement.

ITEMS > MAX # ITEMS/BASE – You can specify a maximum of 1023 items in an IMAGE/9000 data base.

KEY AND SEARCH ITEM NOT MATCHED – A search item and its related key item must have the same item type and size. They need not have the same name.

LEXICAL EXPECTED – After the BEGIN DATA BASE expression and before a semicolon, the only other schema statement allowed is LEXICAL ORDER = .

MASTER DETAIL CODE – The code values of a search item must be an ordered subset of the code values of the related key item.

MASTER SETS MUST COME BEFORE DETAIL SETS – In a data base which has both master and detail data sets, you must define the master data sets in the SET Part before you define the detail data sets.

NAME EXPECTED – The next schema statement after a SETS: statement is the NAME: statement defining the start of a set.

NO ITEMS PRESENT – The ITEM Part of the schema must define at least one data item.

NO SETS PRESENT – The SET Part of the schema must define at least one data set.

ORDER EXPECTED – The reserved word ORDER is expected after the reserved word LEXICAL in the DATA BASE Part.

PASSWORDS EXPECTED – The next schema statement after the DATA BASE Part is the PASSWORDS: statement. This section is required even if no passwords are defined.

PSEUDONYM.CODE.PICTURE.RANGE.NULL.DIMENSION EXPECTED – Until you delimit the definition of a single data base item with a semicolon, the processor expects only item extension expressions.

PATH > MAX # PATHS – A key item in a master data set can have a maximum of 31 paths. A detail data set can have a maximum of 31 search items.

E-8 Errors

PSEUDOS > MAX # PSEUDOS /ITEM – Each data item can have a maximum of four pseudonym names.

PSEUDOS > MAX # PSEUDOS/BASE – A data base can have a maximum of 255 total pseudonym names.

ROOT FILE CORRUPT – The contents of the root file are inconsistent. Purge the data base and re-define it.

SEARCH AND SORT ITEM MATCHED – The sort item for a detail path must not be the search item which defines that path.

SECONDARY VOLUME NOT SPECIFIED IN DIRECTORIES PART – A data set located on a different volume from the root file must reference a directory name specified in the DIRECTORIES Part.

SET HAS NO ACCESS – If one or more passwords are defined for the data base, each data set must specify an access list which has at least one read and/or write user class number.

SET HAS NO FIELD – You must specify at least one data item for each data set.

SET HAS NO PATH – Each detail data set which is not a stand-alone data set must be linked to at least one master set.

SET HAS PATH – All automatic master data sets and all manual master data sets which are not stand-alone must be linked to at least one detail data set.

SETS > MAX # SETS – An IMAGE/9000 data base can have a maximum of 63 data sets.

SET NOT DETAIL – The current data set is not a detail data set, but you have defined a search item in it.

SET NOT MASTER – The current data set is not a master data set, but you have defined a key item in it. The referenced master set must have been defined in the SET Part.

SETS EXPECTED – The next schema statement expected after the final semicolon of the ITEM Part is the SETS: statement.

SPECIFIED DIRECTORIES NOT ACCESSED – The DIRECTORIES Part specifies one or more directory names/paths which are not referenced in the SET Part.

STRUCTURE ABSENT – Due to a prior error, a needed part of the memory image of the root file is missing.

STRUCTURE PRESENT – QUERY/DEFINE cannot create a root file that already exists.

RANGE OF SEARCH ITEM > RANGE OF KEY ITEM – The range of a search must be equal to or within the range of the key item in the linked master data set.

VOLUMES > MAX # VOLUMES ALLOWED – An IMAGE/9000 data base can reside on a maximum of 20 different volumes.

WRONG UTIL VERSION – The revision level of the IMAGE_DBM option BIN file is incompatible with the revision of the BASIC Language System.

Appendix F

Fortran Programming Language Considerations and Examples

Data Base Item Types

The following table summarizes the correspondence between the IMAGE type designators, the terminology used in this manual and the Fortran variable types.

Designator	Description	Fortran variable type	Size in bytes
I2	16-bit integer	integer*2	2
I4	32-bit integer	integer*4	4
R4	32-bit real	real*4	4
R8	64-bit real	real*8	8
Xn	character string	character*n	n

Fortran Library Calls

You do not need to predeclare calls to IMAGE-UX from the Fortran language as externals. However, some linker performance improvement may be seen if you use the Fortran external statement. For example:

```
external dbopen, dbclose, dbset
```

You can identify the actual IMAGE-UX library routines (contained in */usr/lib/libim.a*) at compile time using the **-l** option. For example:

```
fc f/mysource.f -o bin/myobject -lim
```

Fortran Parameters

All parameters to IMAGE-UX must be call-by-reference; parameters must point to the actual data. For simple numeric variables in Fortran, this is automatic. Care must be taken with character strings. IMAGE-UX expects a pointer to the first character of the string, but the Fortran character*n type gives a pointer to a general string marker. The easiest way to set up the correct pointer is to use an EQUIVALENCE statement as in the following example for *dbget*.

```
C Mode must be 2-byte integer; status must be ten 2-byte element
C array.

integer*2 status(10), mode

C dbget arg parameter must be a 4-byte integer.

integer*4 arg

C Declare these parameters as integer arrays so that you have
C pointers directly to the first character of the arrays.

integer*2 ibase(40), idset(10), ilist(40), igetbuf(100)

C Also declare these parameters as character arrays to simplify
C assignment statements.

character*80 base, list
character*20 dset
character*200 getbuf

C The character arrays are used for assignments; the integer
C arrays are used for IMAGE-UX calls.

equivalence (base(1:1), ibase(1)), (dset(1:1), idset(1))
equivalence (list(1:1), ilist(1)), (getbuf(1:1), igetbuf(1))

C Character values must be terminated with a semicolon, a blank
C or a null character.

list = '@;'
dset = 'mydataset;'

C Direct mode dbget.

mode = 4

C Get entry #14.

arg = 14

C Actual dbget call uses ibase, idset, etc. to have pointers to
C first element of array and not pointers to string descriptor.

dbget (ibase, idset, mode, status, ilist, igetbuf, arg)
if (status[1] .ne. 0)

C Print error message
```

Fortran Language Example

```

C LIB,fdemo.f - sample Fortran program, using the LIBRARY data base
C               to read entries from data set BOOK and print some
C               of the item fields.
C Sample compile: fc -lim f/LIB,fdemo.f -o bin/LIB,fdemo

      program libfdemo
      external dbopen, dbclose, dbset

```

Set Up Arrays

```

C
C The following tricks equivalencing integer and character types are needed
C to insure that IMAGE gets a pointer to the actual value of the character
C array. Without this, if a character type were passed to IMAGE, it would
C be a pointer to a STRING MARKER rather than to the actual value area.

      integer*2 ibase(40), ipass(10), idset(10), ilist(40)
      integer*2 s(10), mode, igetbuf(100)
      integer*4 arg, real*4 rprice, character*80 base, list
      character*20 pass, dset
      character*200 getbuf
      character*20 title, author, price
      equivalence (base(1:1), ibase(1)), (dset(1:1), idset(1))
      equivalence (pass(1:1), ipass(1)), (list(1:1), ilist(1))
      equivalence (getbuf(1:1), igetbuf(1)), (rprice, igetbuf(1))

```

Dbopen Example

```

C Set up variables and open data base. NOTE: character arrays are used in
C initializing the strings. Also, ibase and NOT base must be used in the
C IMAGE call.

      base = ' LIBRARY;'
      pass = 'LIBRMGR;'
      mode = 8
      call dbopen (ibase, ipass, mode, s)
      if (s(1).ne. 0) goto 9000

C Set up header information.

      write (6,*)
      write (6,*) 'Fortran/IMAGE Demo Program'
      write (6,*) '*****'
      write (6,*)
      write (6,50) 'Title', 'Author', 'Price'
      write (6,50) '-----', '-----', '-----'
50  format (a5,45x,a6,17x,a5)
      write (6,*)

```

F-4 FORTRAN Programming Language Considerations and Examples

Serial dbget

```
C NOTE: The order of the list parameter is important to
C insure that PRICE is on a full word boundary. If we had set the
C list = 'TITLE,AUTHOR,PRICE;' then PRICE would NOT have been on a full
C word boundary. Hence rprice would not have been equivalenced to the
C right set of bits. It is always safest to make string lengths multiples
C of 4 when defining the data base!

100 mode = 2
   dset = 'BOOK;'
   list = 'PRICE,TITLE,AUTHOR;'
   call dbget (ibase, idset, mode, s, ilist, igetbuf, arg)
   if (s(1).eq. 11) goto 140
   if (s(1).ne. 0) goto 9020
```

Print Results Routine

```
C Print results and continue in serial read loop.

120 write (6,122) getbuf(5:54), getbuf(55:79), rprice
122 format (a50, a23, f5.2)
   goto 100
```

Dbclose Example

```
140 mode = 1
   call dbclose (ibase, ipass, mode, s)
   if (s(1).ne. 0) goto 9040
   stop
```

Error Return Routine

```
9000 write (6,*) '*** Error in dbopen: '
      goto 9900
9020 write (6,*) '*** Error in dbget: '
      goto 9900
9040 write (6,*) '*** Error in dbclose: '
9900 do 9910 i=1,10
      write (6,9905) s(i)
9905 format (nn, i5)
9910 continue
      write (6,*)
      stop
      end
```

Appendix G

Glossary

access method – the method by which data entries are retrieved from a data base. In IMAGE/9000 there are seven: re-read, serial forward, serial backward, directed, chained forward, chained backward, and calculated.

automatic master data set – a data set which contains only one data item, the key item. It is related to at least one detail data set. When a new key item value is added to a related detail set, a new entry is automatically added to the master. When the last entry containing that key item value is deleted from all related detail sets, the master entry is automatically deleted.

binary-coded decimal – a system of representing numbers in which each digit (and sign) of a decimal number is represented by a group of four binary digits, abbreviated BCD.

buffer – a segment of contiguous random-access memory locations used for temporary storage of input and output data.

byte – a unit of data storage consisting of 8 bits. A byte can represent any HP 9000 character, an unsigned numeric value in the range 0 thru 255 or a signed numeric value in the range – 128 thru 127.

calculated access – a method for locating an entry in a master data set. The value of the key item is transformed via a hashing algorithm to produce a primary data entry record number.

capacity – the defined capacity of a data base data set is the number of data entries specified for the set in SCHEMA or DEFINE. The actual capacity differs if the set has been extended with DBEXTEND. The capacity is an integer value in the range of a DOUBLE. The defined capacity of a master data set should be a prime number for optimum performance.

chain – a series of paired record numbers (pointers) stored in the header of a data entry's media record. Each pair of pointers designates the previous and next entry on the chain. Entries on chains are related by one of three characteristics: free records (empty entry), synonym (duplicate hash value), and same search value (detail chain).

chained access – a method of locating detail data entries which retrieves the next or previous entry having the same synonym (in a master data set) or search item value (in a detail data set).

code data type – an enumerated data item type with a code value equal to one of 32 user-defined 16-character strings. Each defined value corresponds to a code number in the range 1 to 32. The strings are specified in SCHEMA and DEFINE, and are stored in a code table in the root file. The code table can be accessed via DBINFO mode 108. Only the code number is physically stored in a data entry.

current record – the last entry to be read or written in each data set.

G-2 Glossary

data base — a collection of logically related files. A root file contains structural information. Master and detail data set files contain the data and relational information.

data entry — that portion of a data set media record which contains the values for the data items which are stored in that data set.

data item — the smallest named element in a data base. It is stored in a field of a data entry and can contain a value which corresponds to a program variable. An item can be an entire array, but is always accessed as a single unit.

data set — a collection of data entries. All data entries in the same set have identical structure, but not necessarily identical data. There are two types of data sets, master and detail. See *data set file*.

data set file — a file of type DSET containing all of the media records for a single data set. Each media record can contain one data entry.

detail data set — a data set containing entries for one or more search and/or detail items. If the data set is linked to one or more master data sets, the detail entries containing the same key value are chained. Detail data sets support only re-read, serial, chained and directed access methods.

directed access — a method of retrieving a data entry in which the entry's record number is specified.

hashing — see *calculated access*.

IMAGE name — used to name data bases, data sets, data items, and data item pseudonyms. In BASIC, it is a literal, 1 to 16 characters long, consisting of characters from the following table. The name is terminated by a blank, semicolon, or end of string. If fewer than 16 characters are specified, trailing blanks are presumed.

Characters	Character Codes	Leading	Embedded	Trailing
space	32	NO	NO	YES
0 thru 9	48 thru 57	NO	YES	YES
A thru Z	65 thru 90	YES	YES	YES
—	95	NO	YES	YES
a thru z	97 thru 122	NO	YES	YES
À thru Ò	161 thru 167	YES	YES	YES
É thru Ò	173 thru 174	YES	YES	YES
Ã thru ã	177 thru 178	YES	YES	YES
ç thru ñ	180 thru 183	YES	YES	YES
ä thru ß	192 thru 222	YES	YES	YES

In HP-UX it is a literal. Set names and data base names can be from 1 to 14 characters long. Other names can be up to 16 characters long. The names can consist of characters from the following table.

Character	Character Codes	Leading	Embedded	Trailing
0 thru 9	48 thru 57	NO	YES	YES
A thru Z	65 thru 90	YES	YES	YES
—	95	NO	YES	YES
a thru z	97 thru 122	NO	YES	YES

intrinsic – an HP-UX operating system function available to the user through a high-level language (such as FORTRAN, Pascal, or C), or a BASIC Language statement.

key item – in a master data set, a data item which can be located by calculated access. Each key item in a master data set has a unique value.

maintenance word – a data base protection code defined by DBCREATE and required for all subsequent DBCREATE, DBERASE, DBEXTEND and DBPURGE utility operations on the same data base. The maintenance word is 16 characters long and is composed of any characters in the HP 9000 set except the semicolon. Characters after a blank or semicolon are ignored. If less than 16 are specified, trailing blanks are assumed.

manual master data set – a master data set which can contain detail items in addition to the key item. A manual master need not be linked to a detail data set. Entries in a master set must be explicitly added or deleted.

master data set – a data set whose entries contain one key item and zero or more detail data items. A master set is generally used as an index to one or more detail data sets. Master data sets support all data base access methods.

media – the physical material on which data is stored (as distinct from the device, which does the reading and writing). Media fall into two categories; removable and non-removable. Typical removable media supported by the HP 9000 are:

- HP p/n 9164-0105 8 inch 0.5Mb Flexible Discs
- HP 92190A 5-1/4 inch .25Mb Flexible Discs
- HP 92195A 8 inch 1.2Mb Flexible Discs

Typical non-removable, or Fixed media are:

- :MEMORY volumes
- the disc platters of the internal HP 97093A (system option 610) internal disc
- the disc platters of the peripheral HP 7908/11/12/33 discs

media record – a record in a data set file (type DSET) which contains one data entry. Each media record contains a header (with flag and pointer fields), and the data entry which contains fields for data item values.

modification counter – the number of DBPUT, DBDELETE, and DBUPDATE operations which have been performed on a data base since the counter was last reset. The counter is returned by DBINFO mode 501 as an integer in the range of a double integer. Data base operations which reset the counter include: creation, erasure, and purging of the entire data base, and backup of the data base with an HP-supplied utility.

password – provides access to a protected file, volume or IMAGE data set. It is a non-listable string kept with the file or data set description in the file's directory or data base root file. In HP-UX, a password is also a string of characters used to verify the identity of a user. Passwords can be associated with users and groups.

G-4 Glossary

path — in a data base, the links between master data set entries and the heads of chains of detail set entries which contain the same key item values.

path count — a number specified in a data base schema which specifies the number of paths a key item has from a master data set to linked detail data sets.

path specifier — a sequence of directory names separated by slashes (/), and ending in a file name (any type).

primary entry — in a data base, the record number produced by the calculated access method hashing algorithm is the primary location. If the key item value stored in that entry hashed to that location, it is the primary entry. A different key item value which also hashes to the same record is called a synonym and is stored in a secondary entry.

primary path — in a data base, the path represented by the first defined master data set which is linked to the first search item in the specified detail data set. If no DBFIND is performed, a chained DBGET in a detail data set uses the primary path.

pseudonym — one of zero to four alternate names for a data base data item. The list of pseudonyms for an item is returned by DBINFO mode 107.

schema — a description in IMAGE/9000 schema language which defines the structure of a data base. The schema is normally created in a file. The SCHEMA processor program reads the schema text file as data to create the root file.

schema instructions — optional keywords in a schema, used to provide page control and select options.

schema processor — a program that processes the schema text file and produces the root file.

SDF — an abbreviation for Structured Directory Format. SDF provides tree-structured access to files through the root directory of the volume. Unlike LIF and 9845, SDF supports file type DIR (directory). Each DIR file contains its own catalog of files, which can include additional DIR files.

search item — an item in a detail data set having associated pointers which link its entry to a chain of detail entries containing the same value for that item. The chain is linked to one or more master data sets in which that item is key item.

secondary entry — a master data set record occupied by a data entry which cannot be stored (DBPUT) at its primary location because that record is already occupied by an entry having a different key item value. A secondary entry is linked to its primary location by a synonym chain.

sector — the smallest unit of data written or read by mass storage device drivers. A sector is typically 256 bytes long, and is sometimes referred to as a “physical record”.

serial access — the reading or writing of records in a file sequentially. In a data base, the reading (DBGET) or writing (DBUPDATE) of entries sequentially.

sort item — a data item in a detail data set which is used by IMAGE to sort a chain in DBPUT. New entries are inserted in the current chain in ascending order according to the value of the sort item. Duplicate sort item values are inserted in chronological order.

status array — an integer array with at least ten elements, into which status information is returned after most DBM operations.

synonym — a key item value which hashes to a record already occupied by a different key item value. A new synonym is stored in the next available free record, which is then linked to the primary location by a synonym chain. This is called a secondary entry.

volume — a complete and separate unit of mass storage which represents an independent file system. It is often a unique mountable medium such as a floppy disc. It is the largest organizational unit of mass storage on a medium.

volume label — a literal which uniquely identifies a volume. The name consists of HP 9000 characters excluding the slash (/), less-than (<), colon (:), and semicolon (;). A single period (.) or double period (..) cannot be used as a label. Leading and embedded blanks are deleted. Any name with a length less than the maximum for the directory format is considered to have trailing blanks.

In a Structured Directory (**SDF**), labels are 1 to 16 characters long.

In an HP 9835/9845 directory (**9845**), labels are 1 to 6 characters long. If strict compatability with the HP9835/45 is desired, you should also restrict 9845 labels to exclude the ASCII NUL (CHR\$(0)), comma (,), space (), and the DEL (CHR\$(127)) characters.

Appendix I

HP IMAGE Data Base Compatibility

Introduction

This Appendix compares the specifications and capabilities of IMAGE/9000 to IMAGE/250, IMAGE/1000, IMAGE/3000 and IMAGE/45. It is intended to summarize the considerations involved in transporting data bases between Hewlett-Packard computer systems, and in the writing of data base programs intended to execute on multiple systems. The information here may not represent the most recent specifications for products other than IMAGE/9000.

Specifications

Data Bases

	HP 9000					
	BASIC	HP-UX	HP 250	HP 1000	HP 3000	HP 9845
Data bases per computer	note ¹	note ¹	note ¹	note ¹	note ¹	note ¹
Data base name length in bytes	16	14	6	6	6	4
Data bases open to one program/process	100	100	5	note ²	63	5
Data base open at one time	100	100	30	note ²	note ²	5
Volumes per data base	20	20	4	n.a.	n.a.	24
Directory structure	SDF	SDF	250	1000	3000	9835/45.

Passwords

	HP 9000					
	BASIC	HP-UX	HP 250	HP 1000	HP 3000	HP 9845
Data base files protected	YES	YES	YES	YES	YES	YES
Maintenance word length in bytes	16	16	YES	n.a.	8	6
Number of DBOPEN passwords	13	13	31	15	63	31
Password length in bytes	16	16	8	6	8	8
Password character restrictions	IMAGE name	none	none	none	ASCII	none
Password grants access to ... level	Set	Set	Set	Item	Item	Set

¹ Limited only by the total disc space on the system.

² Limited by available system memory.

I-2 HP IMAGE Data Base Compatibility

Data Sets

	HP 9000		HP 250	HP 1000	HP 3000	HP 9845
	BASIC	HP-UX				
Data sets per data base	63	63	50	50	99	32
Sets span volumes?	NO	NO	NO	NO	NO	NO
Data set name length in bytes	16	14	8	6	16	15
Lower case allowed?	YES	YES	NO	NO	NO	YES
Set types	A,M,D	A,M,D	A,M,D	A,M,D	A,M,D	A,M,D

Data Paths

	HP 9000		HP 250	HP 1000	HP 3000	HP 9845
	BASIC	HP-UX				
Paths per master set	31	31	8	16	16	16
Paths per detail set	31	31	8	16	16	16
Optional sorted chains	YES	YES	NO	YES	YES	NO
Extended sort on equal sort items	NO	NO	n.a.	NO	YES	n.a.
Sort LEXICAL orders	5	ASCII	ASCII	ASCII	ASCII	ASCII

Data Entries

	HP 9000		HP 250	HP 1000	HP 3000	HP 9845
	BASIC	HP-UX				
Entries per data base	note ¹	note ¹	note ¹	note ¹	note ¹	note ¹
Entries per data set	note ³	note ³	65534	note ³	note ³	32767
Data entry length in bytes	note ³	note ³	1022	4092	8388608	1022
Media record length in bytes	note ³	note ³	1024	4096	8388612	1024
Migrating secondaries	NO	NO	YES	NO	YES	YES

¹ Limited only by the total disc space on the system.

³ Limited only by disc volume size.

⁴ This feature fully supported only by QUERY.



Data Items

	HP 9000		HP 250	HP 1000	HP 3000	HP 9845
	BASIC	HP-UX				
Items per data base	1023	1023	255	255	255	255
Items per data set	1023	1023	127	127	255	127
Item name length in bytes	16	16	15	6	16	15
Lower case in names	YES	YES	NO	NO	NO	YES
Item types:						
Enumerated CODE	C2	-	-	-	-	I
16-bit integer	I2	I2	I	I1	I,J	I
16-bit unsigned	-	-	-	-	K1	-
32-bit integer	I4	I4	-	-	I2,J2	-
32-bit floating	R4	R4	S	R2	R2	S
64-bit floating	R8	R8	L	-	R4	L
Packed decimal	-	-	-	-	Pn	-
Zoned-decimal str.	-	-	-	Xn ⁴	Zn	-
String, uppercase	-	-	-	-	Un	-
String, any 8-bit	Xn	Xn	Xn	Xn	Xn	Xn
Xn: max n in bytes	note ³	note ³	1022	255	4094	1022
Floating pt. format	IEEE	IEEE	BCD	1000	3000	BCD
Search item length	256	256	255	255	4094	255
Sort item length	256	256	n.a.	255	254	n.a.

Data Item Extensions

	HP 9000		HP 250	HP 1000	HP 3000	HP 9845
	BASIC	HP-UX				
Compound (array) items	YES	YES	YES	YES	YES	YES
Number of dimensions	6	6	1	1	1	1
Max element count	note ³	note ³	50	2048	255	511
CODE value string length, bytes	16	n.a.	n.a.	n.a.	n.a.	15 ⁴
CODE values per item	32	n.a.	n.a.	n.a.	n.a.	35 ⁴
CODE values per data base	255	n.a.	n.a.	n.a.	n.a.	250 ⁴
Default NULL values	YES	YES	NO	NO	NO	YES ⁴
PICTURE/CONTROL word	YES ⁴	NO	NO	NO	NO	YES ⁴
Pseudonym name length, bytes	16	n.a.	n.a.	n.a.	n.a.	15 ⁴
Pseudonym names per item	4	n.a.	n.a.	n.a.	n.a.	5 ⁴
Pseudonym names per data base	255	n.a.	n.a.	n.a.	n.a.	250 ⁴
RANGE limits	YES ⁴	NO	NO	NO	NO	YES ⁴

³ Limited by disc volume space

⁴ This feature fully supported only in QUERY.

IMAGE Intrinsic

Implementation

	HP 9000		HP 250	HP 1000	HP 3000	HP 9845
	BASIC	HP-UX				
Software in...	RAM	RAM	DROM	RAM	RAM	ROM

Host Languages

	HP 9000		HP 250	HP 1000	HP 3000	HP 9845
	BASIC	HP-UX				
Assembly	n.a.	n.a.	n.a.	YES	SPL	NO
BASIC, compiled	YES	n.a.	n.a.	YES	n.a.	n.a.
BASIC, interpreted	n.a.	n.a.	YES	YES	YES	YES
C	n.a.	YES	n.a.	YES	n.a.	n.a.
COBOL	n.a.	n.a.	n.a.	YES	YES	n.a.
FORTRAN	n.a.	YES	n.a.	YES	YES	NO
Pascal	n.a.	YES	n.a.	YES	YES	NO
RPG	n.a.	n.a.	n.a.	n.a.	YES	n.a.

Syntax

	HP 9000		HP 250	HP 1000	HP 3000	HP 9845
	BASIC	HP-UX				
Status array size in bytes	40	20	20	20	20	20
Status array variable type	DOUBLE	any	INTEGER	any	any	INTEGER
Full record list mode	"@"	"@"	"@"	"@"	"@"	"@"
Item name list mode	YES	YES	YES	YES	YES	NO
Item number array list mode	NO	YES	NO	YES	YES	NO
Current list mode	"**"	"**"	NO	NO	"**"	NO
Null list (no write/no read)	YES	YES	NO	YES	YES	NO
Single buffer for data	YES	YES	YES	YES	YES	NO
Variable list for data or "IN DATA SET"	YES	NO	YES	YES ⁵	YES ⁵	NO

DBBEGIN

	HP 9000		HP 250	HP 1000	HP 3000	HP 9845
	BASIC	HP-UX				
Define beginning of logged transaction	NO	NO	NO	NO	YES	NO

⁵ This feature available only in BASIC

DBBUFFER

Modes	HP 9000		HP 250	HP 1000	HP 3000	HP 9845
	BASIC	HP-UX				
Control I/O buffer size	YES	NO	NO	NO	NO	NO
Control I/O buffer count	YES	NO	NO	NO	NO	NO

DBCLOSE

Modes	HP 9000		HP 250	HP 1000	HP 3000	HP 9845
	BASIC	HP-UX				
Close entire data base	1	1	1	1	1	1
Close a specified data set	2	2	-	2	2	-
Rewind a specified data set	3	3	3	-	3	-
Post a specified data set's buffers	4	4	4	-	-	4
Automatic DBCLOSE on program stop/ abort	YES	YES	NO	NO	NO	NO

DBCONTROL

Modes	HP 9000		HP 250	HP 1000	HP 3000	HP 9845
	BASIC	HP-UX				
Deferred or immediate posting	1,2	1,2	-	-	1,2	-
Load CODE table into memory	3,4	-	-	-	-	-
Load PSEUDONYM table into memory	5,6	-	-	-	-	-

DBCREATE

	HP 9000		HP 250	HP 1000	HP 3000	HP 9845
	BASIC	HP-UX				
Create empty data set	YES	YES	YES	NO	DBUTIL	YES

DBDELETE

	HP 9000		HP 250	HP 1000	HP 3000	HP 9845
	BASIC	HP-UX				
Delete current entry	YES	YES	YES	YES	YES	YES

DBEND

	HP 9000		HP 250	HP 1000	HP 3000	HP 9845
	BASIC	HP-UX				
Define the end of a logged transaction	NO	NO	NO	NO	YES	NO

DBERASE

	HP 9000		HP 250	HP 1000	HP 3000	HP 9845
	BASIC	HP-UX				
Return a data set to the empty state	YES	YES	YES	NO	DBUTIL	YES

DBERROR

	HP 9000		HP 250	HP 1000	HP 3000	HP 9845
	BASIC	HP-UX				
Get short text error message	NO	NO	NO	NO	YES	NO

DBEXPLAIN

	HP 9000		HP 250	HP 1000	HP 3000	HP 9845
	BASIC	HP-UX				
Send long text error message to log device	NO	NO	NO	NO	YES	NO

DBEXTEND

	HP 9000		HP 250	HP 1000	HP 3000	HP 9845
	BASIC	HP-UX				
Increase capacity of existing data set	YES	YES	NO	NO	NO	NO

DBFIND

	HP 9000		HP 250	HP 1000	HP 3000	HP 9845
	BASIC	HP-UX				
Set up detail chain	YES	YES	YES	YES	YES	YES

DBGET

	HP 9000		HP 250	HP 1000	HP 3000	HP 9845
	BASIC	HP-UX				
Re-read current record	1	1	-	1	1	-
Serial forward	2	2	2	2	2	2
Serial backward	3	3	-	3	3	-
Directed	4	4	4	4	4	4
Chained forward	5	5	5	5	5	5
Chained backward	6	6	-	6	6	-
Calculated	7	7	7	7	7	7
Primary calculated	-	-	-	-	8	-

DBINFO

Modes: item info	HP 9000		HP 250	HP 1000	HP 3000	HP 9845
	BASIC	HP-UX				
Item number	101	101	101	101	101	101
item name, type and size	102	102	102	102	102	102
Item count and item numbers	103	103	-	103	103	-
Set's item count and item numbers	104	104	104	104	104	104
Set's item count and fields	105	105	-	-	-	501
Item NULL values	106	n.a.	-	-	-	-
Pseudonym count and pseudonym values	107	n.a.	-	-	-	-
CODE count, values	108	n.a.	-	-	-	-
PICTURE word, NULL and RANGE values	109	n.a.	-	-	-	102
Item dimensions, size and bounds	110	110	-	-	-	-

DBINFO

Modes: Set info	HP 9000		HP 250	HP 1000	HP 3000	HP 9845
	BASIC	HP-UX				
Set number	201	201	201	201	201	201
Set name, entry and capacity	202	202	202	202	202	202
Set count and set number	203	203	203	203	203	203
Set in which item is found	204	204	204	204	204	204
Set list and open status	205	205	-	-	-	-
Media record and buffer info	206	-	-	-	-	-

DBINFO

Miscellaneous modes	HP 9000		HP 250	HP 1000	HP 3000	HP 9845
	BASIC	HP-UX				
Paths, set, key and sort item info	301	301	301	301	301	301
Key item and set number	302	302	302	302	302	302
Modification count and lexical order	504	504	-	-	-	-
Volume number	601	601	401	-	-	401
Directory specifier	602	602	402	-	-	402
Directory path count and volume number	603	603	403	-	-	403
Set count and set numbers	604	604	404	-	-	404
Directory path specifier	605	605	-	-	-	-
Get chain info	-	-	-	401	-	-
Restore chain info	-	-	-	402	-	-
Get log info	-	-	-	-	401	-
Check access	-	-	-	-	501	-

DBLOCK

Modes	HP 9000		HP 250	HP 1000	HP 3000	HP 9845
	BASIC	HP-UX				
Odd = with wait, Even = without wait	YES	YES	YES	YES	YES	n.a.
Entire data base	1,2	1,2	1,2	1,2	1,2	-
Single data set	3,4	3,4	3,4	-	3,4	-
Record (predicate)	-	-	5,6	-	5,6	-
List of data sets	7,8	7,8	-	-	-	-
Data base, read-only	-	-	11,12	-	-	-
Data set, read-only	-	-	13,14	-	-	-
Record, read-only	-	-	15,16	-	-	-

DBMEMO

	HP 9000		HP 250	HP 1000	HP 3000	HP 9845
	BASIC	HP-UX				
Send message to log device	NO	NO	NO	NO	YES	NO

DBOPEN

Modes	HP 9000		HP 250	HP 1000	HP 3000	HP 9845
	BASIC	HP-UX				
Shared read/write, locking required	1	1	1	1	1	-
Shared read/write, without locking	-	-	-	-	2	-
Exclusive read/write	3	3	3	3	3	3
Exclusive write, shared read	-	-	-	-	4	-
Shared read with other mode 1,5,9	-	-	-	-	5	-
Shared read with other mode 4,8,9	-	-	-	-	6	-
Exclusive read	-	-	-	-	7	-
Shared read	8	8	8	8	8	8
Mode 3 with deferred buffer posting	11	-	-	-	-	11
Open a remote data base	-	-	NO	YES	YES	NO
Share a remote data base	-	-	n.a.	YES	YES	n.a.

DBPURGE

	HP 9000		HP 250	HP 1000	HP 3000	HP 9845
	BASIC	HP-UX				
Purge existing data set	YES	YES	YES	NO	DBUTIL	YES

DBPUT

	HP 9000		HP 250	HP 1000	HP 3000	HP 9845
	BASIC	HP-UX				
Add new data entry	YES	YES	YES	YES	YES	YES

DBSTATUS

	HP 9000		HP 250	HP 1000	HP 3000	HP 9845
	BASIC	HP-UX				
List data base status report	YES	YES	NO	DBSPA	DBUTIL	NO

DBUNDO

	HP 9000		HP 250	HP 1000	HP 3000	HP 9845
	BASIC	HP-UX				
Cancel uncommitted transaction log	NO	NO	NO	NO	YES	NO

DBUNLOCK

Modes	HP 9000		HP 250	HP 1000	HP 3000	HP 9845
	BASIC	HP-UX				
Release all locks owned	1	1	1	1	1	n.a.
Release only specified data sets	2	2	-	-	-	n.a.

DBUPDATE

	HP 9000		HP 250	HP 1000	HP 3000	HP 9845
	BASIC	HP-UX				
Alter current record	YES	YES	YES	YES	YES	YES

Data Base Backup

	HP 9000					
	BASIC	HP-UX	HP 250	HP 1000	HP 3000	HP 9845
Unload/reload set data to/from disc	QUERY	NO	YES	DBLOAD	YES	DBLOD
Unload/reload set data to/from tape	NO	NO	NO	DBULD	YES	DBUNLD
Save/restore sets to/from disc as sets	QUERY	NO	YES	DBSTR	YES	YES
Save/restore sets to/from tape as sets	NO	NO	NO	DBRST	YES	NO
Save/restore sets to/from disc as files.	COPY	tar	COPY	FCOPY,	YES	COPY
Save/restore sets to/from tape as files	YES	cp	NO	others	YES	YES
Save/restore volume to/from disc	YES	tar	YES	FCOPY,	YES	YES
Save/restore volume to/from tape	YES	cp	NO	others	YES	YES

Miscellaneous

	HP 9000					
	BASIC	HP-UX	HP 250	HP 1000	HP 3000	HP 9845
DBBLD: build data set from ASCII file	NO	NO	NO	YES	NO	NO
RECOV: close a data base left open	n.a.	n.a.	n.a.	YES	YES	NO

QUERY

	HP 9000		HP 250	HP 1000	HP 3000	HP 9845
	BASIC	HP-UX				
Availability	YES	NO	YES	YES	YES	YES
Interactive	YES	n.a.	YES	YES	YES	YES
Batch	NO	n.a.	NO	YES	YES	NO
Commands from files	YES ⁶	n.a.	YES ⁶	YES	YES	NO
Output to files	NO	n.a.	SPOOL	YES	YES	NO
Opens data base shared or exclusive	EXC	n.a.	EXC	EXC	Opt.	n.a.
Forms mode	YES	n.a.	YES	NO	NO	YES
Report writer	NO	n.a.	NO	YES	YES	NO
Run user subprogram	YES	n.a.	n.a.	n.a.	n.a.	YES
Create data base (DEFINE module)	YES	n.a.	NO	NO	NO	YES
List schema	YES	n.a.	YES	YES	YES	YES
Diagram schema	YES	n.a.	NO	NO	NO	YES
Threads (multi-set searches)	YES	n.a.	YES	NO	YES	YES
Item type extensions:						
CODE	YES	n.a.	NO	NO	NO	YES
CURRENCY	NO	n.a.	YES	NO	NO	NO
DATE	YES	n.a.	YES	NO	NO	YES
NAME	YES	n.a.	NO	NO	NO	YES
PICTURE numeric	YES	n.a.	NO	NO	NO	YES
RANGE	YES	n.a.	NO	NO	NO	YES
REQUIRED	NO	n.a.	NO	NO	NO	YES
TIME (CLOCK)	YES	n.a.	YES	NO	NO	YES
TIMEDATE	YES	n.a.	YES	NO	NO	YES
USER/UNDEFINED	YES	n.a.	YES	NO	NO	YES
WRITE inhibit	NO	n.a.	YES	NO	NO	NO
Sort data	YES	n.a.	YES	YES	YES	NO
Alter data base	YES	n.a.	YES	NO	NO	YES

⁶ Formal commands only.

1-12 HP IMAGE Data Base Compatability

Appendix P

Pascal Programming Language Considerations and Examples

Data Base Item Types

The following table summarizes the correspondence between the IMAGE type designators, the terminology used in this manual and the Pascal variable types.

Designator	Description	Pascal variable type	Size in bytes
I2	16-bit integer	-32768..32767	2
I4	32-bit integer	integer	4
R4	32-bit real	real	4
R8	64-bit real	longreal	8
Xn	character string	packed array [1..n] of char	n

Pascal Library Calls

You need to predeclare calls to IMAGE-UX from the Pascal language as externals. For example, a Pascal program using *dbopen* could have the following:

```

type base_type = packed array [1..80] of char;
    pass_type = packed array [1..16] of char;
    i16_type = -32768..32767;
    mode_type = i16_type;
    st_type = packed array [1..10] of i16_type;

procedure dbopen (var base : base_type;
                  var pass : pass_type;
                  var mode : mode_type;
                  var st : st_type);
external;
```

You could identify the actual IMAGE-UX library routines (contained in */usr/lib/libim.a*) at compile time using the **-l** option. For example:

```
PC P/mysource.P -o bin/myobject -lim
```

Pascal Aliasing

Suppose a program is doing dbputs to two different data sets. Only one type declaration can be given for the buffer parameter for the external procedure declaration for *dbput*. How can we express two different dbput buffer parameters? The simplest way is to use the Pascal compiler's ALIAS option. This allows you to refer to a routine by a name in the source code that is different from the name used in the object code. You define several routines with different parameter lists that represent the same *dbput* library call. For example:

```
{* PUT_BUF_SET_A and PUT_BUF_SET_B are the buffer *}
{* structures for the two sets we plan to use, *}

type PUT_BUF_SET_A = packed array [1..120] of chars;

      PUT_BUF_SET_B = packed record of
          cnt : integer;
          str : packed array [1..20] of char;
      end;

procedure PUT_TO_SET_A      $ALIAS 'dbput'$
    ( var db_namr : database_name;
      var data_set : dset_name;
      var put_mode : single_integer;
      var status : image_put_status_type;
      var put_list : lr_string2;
      var put_buf_a : PUT_BUF_SET_A);
external;

procedure PUT_TO_SET_B      $ALIAS 'dbput'$
    ( var db_namr : database_name;
      var data_set : dset_name;
      var put_mode : single_integer;
      var status : image_put_status_type;
      var put_list : lr_string2;
      var put_buf_b : PUT_BUF_SET_B);
external;
```

Instead of using *dbput* in your program, you can now use either of the following:

```
PUT_TO_SET_A (db_namr,data_set,put_mode,status,put_list,put_buf_a);
PUT_TO_SET_B (db_namr,data_set,put_mode,status,put_list,put_buf_b);
```

Pascal Parameters

All parameters to IMAGE-UX must be call-by-reference; parameters must point to the actual data. In Pascal, you must declare each parameter as a "var" parameter as in the previous examples.

Pascal Language Example

```

(* LIB.Pdemo.P - demo program on using Pascal/IMAGE                                *)
(* Given a Key value, this program does a dbfind                                  *)
(* and chained dbsets. The LIBRARY data base is used.                            *)
*)

(* AIM: BASIC allows some packing of information which is                        *)
(* not allowed in PASCAL/9000. This program will exhibit                          *)
(* the problem.                                                                    *)
(* To avoid the problem, if PASCAL is to be the language of                      *)
(* choice, character fields in data sets should all have a                       *)
(* multiple of 4 # of characters.                                                 *)
*)

(* Here's one possible way to compile this program:                              *)
(* PC P/LIB.Pdemo.P -lim -o bin/LIB.Pdemo                                         *)
*)

Program assign(input, output);

```

Type Declarations

```

type
  single_integer = -32768..32767;
  base_type      = packed array [1..18] of char;
  pass_type      = packed array [1..16] of char;
  status_type    = array [1..10] of single_integer;
  dset_type      = packed array [1..16] of char;
  item_type      = packed array [1..16] of char;
  list_type      = packed array [1..250] of char;
  key_type       = packed array [1..40] of char;

  badPackedbuf_type = packed record
    title       : packed array [1..50] of char;
    callno      : integer;
    author       : packed array [1..25] of char;
    subject      : packed array [1..30] of char;
    pub_date     : longreal;
    publisher    : packed array [1..30] of char;
    price        : real;
  end;

  ok1Packedbuf_type = packed record
    title       : packed array [1..50] of char;
    author       : packed array [1..25] of char;
    subject      : packed array [1..30] of char;
    publisher    : packed array [1..30] of char;
  end;

  ok2Packedbuf_type = packed record
    callno      : integer;
    pub_date     : longreal;
    price        : real;
  end;

  buffer_type    = record case integer of
    1: (badPackedbuf : badPackedbuf_type);
    2: (ok1Packedbuf : ok1Packedbuf_type);
    3: (ok2Packedbuf : ok2Packedbuf_type);
  end;

```

Variable Declarations

```
var
  base : base_type;
  pass : pass_type;
  mode : single_integer;
  dset : dset_type;
  stat : status_type;
  list : list_type;
  item : item_type;
  dum1 : integer;
  buf  : buffer_type;
  dum2 : integer;
  buf1 : buffer_type;
  dum3 : integer;
  buf2 : buffer_type;
  key  : key_type;
  arg  : integer;
```

External Declarations

```
{ NOTE: The dbset buffer uses a variant record type to account for the }
{      different buffer types. The Pascal ALIAS option could be used }
{      instead.                                                         }

procedure dbopen (var base : base_type;
                  var pass : pass_type;
                  var mode : single_integer;
                  var stat : status_type); external;

procedure dbfind (var base : base_type;
                  var dset : dset_type;
                  var mode : single_integer;
                  var stat : status_type;
                  var item : item_type;
                  var key  : key_type); external;

procedure dbset (var base : base_type;
                 var dset : dset_type;
                 var mode : single_integer;
                 var stat : status_type;
                 var list : list_type;
                 var buf  : buffer_type;
                 var arg  : integer); external;

procedure dbclose (var base : base_type;
                   var qual : pass_type;
                   var mode : single_integer;
                   var stat : status_type); external;
```

Error Message Routine

```
procedure db_msg (list : list_type;
                 stat : status_type);

var
  i : integer;

begin
  write (list);
  for i := 1 to 10 do
    write (' ', stat[i]:3);
  writeln; writeln;
  halt (1);
end;
```

Print Entry Routine

```

procedure pentry (buf  : buffer_type;
                  buf1 : buffer_type;
                  buf2 : buffer_type);
begin
  writeln ('Entry #: ', stat[4]:5);

  with buf, badpackedbuf do
  begin
    writeln;
    writeln ('*** Misaligned buffer ***');
    writeln ('      Title: ', title);
    writeln ('      Call #: ', callno:10);
    writeln ('      Author: ', author);
    writeln ('      Subject: ', subject);
    writeln ('      Publisher: ', publisher);
    writeln ('      Price: ', price:5:2);
    writeln;
  end;

  writeln ('*** Properly aligned buffer ***');
  writeln ('      Title: ', buf1.ok1packedbuf.title);
  writeln ('      Call #: ', buf2.ok2packedbuf.callno:8);
  writeln ('      Author: ', buf1.ok1packedbuf.author);
  writeln ('      Subject: ', buf1.ok1packedbuf.subject);
  writeln ('      Publisher: ', buf1.ok1packedbuf.publisher);
  writeln ('      Price: ', buf2.ok2packedbuf.price:5:2);

  writeln; writeln ('*****'); writeln;
end;

```

Main Program

```

begin
{ Print header information. }
  writeln;
  write ('                      Pascal/IMAGE Demo Program');
  writeln;
  write ('                      *****');
  writeln; writeln;

  writeln ('  Given a particular AUTHOR name (SAGE,EDWIN), this program');
  writeln ('  does a dbfind on the detail set BOOK to find all books');
  writeln ('  in the LIBRARY data base written by this author.');
```

```

  writeln;

  writeln ('  NOTE:  Results of properly and improperly aligning set')
  writeln ('  buffers are also given.');
```

```

  writeln;

```

Dbopen Example

```

base := ' LIBRARY';
base[10] := chr(0); {Terminator must be semicolon, blank or null char}

pass := 'LIBRMGR';
pass[8] := ' ';

mode := 1;
dbopen (base, pass, mode, stat);
if (stat[1] <> 0)
  then db_msg ('*** Error on dbopen: ', stat);

```

Dbfind Example

```
{ Set up chain in dset BOOK for AUTHOR=EDWIN SAGE. }
mode := 1;
dset := 'BOOK';
dset[5] := ' '; {For bkward compatibility, blank is a valid terminator.}

item := 'AUTHOR'; item[7] := chr(0);
key := 'SAGE,EDWIN';
dbfind (base, dset, mode, stat, item, key);
if (stat[1] <> 0)
    then db_msg ('*** Error on dbfind: ', stat);
```

Chained dbget Example

```
while (stat[1] = 0) do
begin
    list[1] := '@'; list[2] := chr(0);
    mode := 5;
    dbget (base, dset, mode, stat, list, buf, arg);
    if (stat[1] <> 0)
        then if (stat[1] = 15)
            then writeln ('End of chain on dbget.')
            else db_msg ('*** Error on dbget: ', stat)
        else { Got a good record }
        begin
            mode := 1;
            list := 'TITLE,AUTHOR,SUBJECT,PUBLISHER';
            dbget (base, dset, mode, stat, list, buf1, arg);
            if (stat[1] <> 0)
                then db_msg ('*** Error on dbget: ', stat);

            mode := 1;
            list := 'CALL_NUMBER,PUBLISHED_DATE,PRICE';
            dbget (base, dset, mode, stat, list, buf2, arg);
            if (stat[1] <> 0)
                then db_msg ('*** Error on dbget: ', stat)
                else entry (buf, buf1, buf2)
            end; { Else of good record }

        end; { while loop }
    writeln;
```

Dbclose Example

```
mode := 1;
dbclose (base, pass, mode, stat);
if (stat[1] <> 0)
    then db_msg ('*** Error on dbclose: ', stat);

END;
```

Appendix Q

QUERY Type Extensions

Introduction

In addition to the normal numeric and string types, IMAGE provides your application program access to data type CODE and type extensions NULL, PICTURE, PSEUDONYM, and RANGE. IMAGE supports the NULL extension. Only QUERY transparently supports the other types.

Each of these extensions is implemented as an entry or table in the data base root file. You can retrieve this information in your application program via DBINFO. You can interpret this information in any manner appropriate to your application, but it is suggested that you use the QUERY definitions.

The QUERY definition for all of these types except PICTURE is fully described in Chapter 2 of this manual, and in the description of DBINFO in your Reference manual.

SCHEMA only partially supports the use of the PICTURE type attribute. You can specify a double integer constant in the PICTURE expression. DBINFO can return the constant to your program, but does not interpret it. This appendix describes the QUERY interpretation of the PICTURE constant.

QUERY defines five PICTURE interpretations: TIMEDATE, DATE, NAME, NUMERIC and TIME. Each implies that the data is in a certain form and may imply a conversion which is performed on the data when it is stored in or retrieved from the data base.

PICTURE Interpretations and IMAGE Data Types

Not all PICTURE interpretations are defined for all data types. The following table summarizes PICTURE useability.

Data Types	QUERY PICTURE interpretation					
	TIMEDATE	DATE	NAME	numeric	TIME	USER
CODE	NO	NO	NO	YES	NO	YES
String	NO	NO	YES	NO	NO	YES
Integer	NO	NO	NO	YES	NO	YES
Double integer	NO	NO	NO	YES	NO	YES
Shortreal	NO	NO	NO	YES	NO	YES
Longreal	YES	YES	NO	YES	YES	YES

The PICTURE constant is a double integer, but is treated by QUERY as a 32-bit entity which is sub-divided into fields consisting of single bits, 2-bit dibits, 4-bit halfbytes, and 8-bit bytes. The following sections describe each PICTURE word in terms of the “fields” which comprise it.

Default PICTURE Word

If you do not specify a PICTURE extension when you define the data base, the value of the item's PICTURE word defaults to zero. A value of zero means that QUERY assigns no interpretation to the data item.

bits	31 thru 28	27 thru 00
field	Type	Not Used
range	0	0

PICTURE Words TIMEDATE, DATE and TIME

TIMEDATE, DATE and TIME signify that the longreal item value represents a Julian time and date. The fields of the word denote how the values are displayed. In the DATE word, the time portion of the Julian value is ignored. In the TIME word, the date value is ignored.

bits	31	30...28	27 thru 00
field	–	Type	Not Used
range	1	below	X

Type field:

Type	Binary	Decimal
TIMEDATE	100	4
TIME	101	5
DATE	110	6

PICTURE word values stored in the root file by QUERY/DEFINE:

Type specified	Decimal value	Hexadecimal value
TIMEDATE	–1073741824	C0000000
TIME	–805306368	D0000000
DATE	–536870912	E0000000

PICTURE Word NAME

NAME signifies that the string item contains a person's last and first names followed by an optional title. A middle name or initial is also optional. The ordering of the name is:

Lastname,Firstname Middle,Title

bits	31	30...28	26 thru 00
field	–	Type	Not Used
range	1	6	X



PICTURE word value stored in root file by QUERY/DEFINE

Type specified	Decimal value	Hexadecimal value
NAME	–536870912	E0000000

PICTURE Word Numeric

The numeric PICTURE word specifies the form in which the application program should display the value of the integer, double integer, shortreal or longreal data item type.

bits	31	30...28	27	26	25...24	23...20	19 ... 16
field	–	Type	Fixed Float	Sign	Not used	Digits	Fraction digits
range	1	6	below	below	X	below	below

Fixed/Float:

Sign: 0 = Display in fixed point format.

1 = Display in floating point format.

0 = Display minus or blank

1 = Display minus or plus

Digits:

Fixed Format: 0...15 Number of optional digits to left of radix indicator; one is required.

Float Format: 0...2 Number of optional digits in exponent; one is required.

Fraction Digits:

Fixed Format: Number of zeros to right of radix indicator.

Float Format: Number of optional zeros to right of radix indicator; one is required.

bits	15 thru 00
field	Not Used
range	X

Q-4 QUERY Type Extensions

PICTURE word values stored by QUERY/DEFINE for various BASIC image specifiers:

Image specifier	Decimal value	Hexadecimal value	Image specifier	Decimal value	Hexadecimal value
SD	-335544320	EC000000	SD.16DESZ	-470745088	E3F10000
SD.DESZ	-469762048	E4000000	SD.16DES2Z	-471793664	E3E10000
SD.DES2Z	-470810624	E3F00000	SD.16DES3Z	-472842240	E3D10000
SD.DES3Z	-471859200	E3E00000	S2D	-336592896	EBF00000
SD.2D	-335765392	EBFE0000	S2D.2D	-336723968	EBEE0000
SD.2DESZ	-469827584	E3FF0000	S2D.14D	-337510400	EBE20000
SD.2DES2Z	-470876160	E3EF0000	S3D	-337641472	EBE00000
SD.2DES3Z	-471924736	E3DF0000	S3D.2D	-337772544	EBDE0000
D.3DESZ	-469893120	E3FE0000	S3D.13D	-338493440	EBD30000
SD.3DES2Z	-470941696	E3EE0000	S4D	-338690048	EBD00000
SD.3DES3Z	-471990272	E3DE0000	S4D.2D	-338821120	EBCE0000
SD.4DESZ	-469958656	E3FD0000	S4D.12D	-339476480	EBC40000
SD.4DES2Z	-471007232	E3ED0000	S5D	-339738624	EBC00000
SD.4DES3Z	-472055808	E3DD0000	S5D.2D	-339869696	EBBE0000
SD.5DESZ	-470024192	E3FC0000	S5D.11D	-340459520	EBB50000
SD.5DES2Z	-471072768	E3EC0000	S6D	-340787200	EBB00000
SD.5DES3Z	-472121344	E3DC0000	S6D.2D	-340918272	EBAE0000
SD.6DESZ	-470089728	E3FB0000	S6D.10D	-341442560	EBA60000
SD.6DES2Z	-471138304	E3EB0000	S7D	-341835776	EBA00000
SD.6DES3Z	-472186880	E3DB0000	S7D.2D	-341966848	EB9E0000
SD.7DESZ	-470155264	E3FA0000	S7D.9D	-342425600	EB970000
SD.7DES2Z	-471203840	E3EA0000	S8D	-342884352	EB900000
SD.7DES3Z	-472252416	E3DA0000	S8D.2D	-343015424	EB8E0000
SD.8DESZ	-470220800	E3F90000	S8D.8D	-343408640	EB880000
SD.8DES2Z	-471269376	E3E90000	S9D	-343932928	EB800000
SD.8DES3Z	-472317952	E3D90000	S9D.2D	-344064000	EB7E0000
SD.9DESZ	-470286336	E3F80000	S9D.7D	-344391680	EB790000
SD.9DES2Z	-471334912	E3E80000	S10D	-344981504	EB700000
SD.9DES3Z	-472383488	E3D80000	S10D.2D	-345112576	EB6E0000
SD.10DESZ	-470351872	E3F70000	S10D.6D	-345374720	EB6A0000
SD.10DES2Z	-471400448	E3E70000	S11D	-346030080	EB600000
SD.10DES3Z	-472449024	E3D70000	S11D.2D	-346161152	EB5E0000
SD.11DESZ	-470417408	E3F60000	S11D.5D	-346357760	EB5B0000
SD.11DES2Z	-471465984	E3E60000	S12D	-347078656	EB500000
SD.11DES3Z	-472514560	E3D60000	S12D.2D	-347209728	EB4E0000
SD.12DESZ	-470482944	E3F50000	S12D.4D	-347340800	EB4C0000
SD.12DES2Z	-471531520	E3E50000	S13D	-348127232	EB400000
SD.12DES3Z	-472580096	E3D50000	S13D.2D	-348258304	EB3E0000
SD.13DESZ	-470548480	E3F40000	S13D.3D	-348323840	EB3D0000
SD.13DES2Z	-471597056	E3E40000	S14D	-349175808	EB300000
SD.13D3S3Z	-472645632	E3D40000	S14D.2D	-349306880	EB2E0000
SD.14DESZ	-470614016	E3F30000	S14D.2D	-349306880	EB2E0000
SD.14DES2Z	-471662592	E3E30000	S15D	-350224384	EB200000
SD.14DES3Z	-472711168	E3D30000	S15D.D	-350289920	EB1F0000
SD.15D	-336527360	EBF10000	S16D	-351272960	EB100000
SD.15DESZ	-470679552	E3F20000	MD	-268435456	F0000000
SD.15DES2Z	-471728128	E3E20000	MD.DESZ	-402653184	E8000000
SD.15DES3Z	-472776704	E3D20000	MD.DES2Z	-403701760	E7F00000

Image specifier	Decimal value	Hexadecimal value	Image specifier	Decimal value	Hexadecimal value
MD.DES3Z	-404750336	E7E00000	M2D	-269484032	EFF00000
MD.2D	-268566528	EFEE0000	M2D.2D	-269615104	EFEE0000
MD.2DESZ	-402718720	E7FF0000	M2D.14D	-270401536	EFE20000
MD.2DES2Z	-403767296	E7EF0000	M3D	-270532608	EFE00000
MD.2DES3Z	-404815872	E7DF0000	M3D.2D	-270663680	EFDE0000
MD.3DESZ	-402784256	E7FE0000	M3D.13D	-271384576	EFD30000
MD.3DES2Z	-403832832	E7EE0000	M4D	-271581184	EFD00000
MD.3DES3Z	-404881408	E7DE0000	M4D.2D	-271712256	EFCE0000
MD.4DESZ	-402849792	E7FD0000	M4D.12D	-272367616	EFC40000
MD.4DES2Z	-403898368	E7ED0000	M5D	-272629760	EFC00000
MD.4DES3Z	-404946944	E7DD0000	M5D.2D	-272760832	EFBE0000
MD.5DESZ	-402915328	E7FC0000	M5D.11D	-273350656	EFB50000
MD.5DES2Z	-403963904	E7EC0000	M6D	-273678336	EFB00000
MD.5DES3Z	-405012480	E7DC0000	M6D.2D	-273809408	EFA60000
MD.6DESZ	-402980864	E7FB0000	M6D.10D	-274333696	EFAE0000
MD.6DES2Z	-404029440	E7EB0000	M7D	-274726912	EFA00000
MD.6DES3Z	-405078016	E7DB0000	M7D.2D	-274857984	EF9E0000
MD.7DESZ	-403046400	E7FA0000	M7D.9D	-275316736	EF970000
MD.7DES2Z	-404094976	E7EA0000	M8D	-275775488	EF900000
D.7DES3Z	-405143552	E7DA0000	M8D.2D	-275906560	EF8E0000
MD.8DESZ	-403111936	E7F90000	M8D.8D	-276299776	EF880000
MD.8DES2Z	-404160512	E7E90000	M9D	-276824064	EF800000
MD.8DES3Z	-405209088	E7D90000	M9D.2D	-276955136	EF7E0000
MD.9DESZ	-403177472	E7F80000	M9D.7D	-277282816	EF790000
MD.9DES2Z	-404226048	E7E80000	M10D	-277872640	EF700000
MD.9DES3Z	-405274624	E7D80000	M10D.2D	-278003712	EF6E0000
MD.10DESZ	-403243008	E7F70000	M10D.6D	-278265856	EF6A0000
MD.10DES2Z	-404291584	E7E70000	M11D	-278921216	EF600000
MD.10DES3Z	-405340160	E7D70000	M11D.2D	-279052288	EF5E0000
MD.11DESZ	-403308544	E7F60000	M11D.5D	-279248896	EF5B0000
MD.11DES2Z	-404357120	E7E60000	M12D	-279969792	EF500000
MD.11DES3Z	-405405696	E7D60000	M12D.2D	-280100864	EF4E0000
MD.12DESZ	-403374808	E7F50000	M12D.4D	-280231936	EF4C0000
MD.12DES2Z	-404422656	E7E50000	M13D	-281018368	EF400000
MD.12DES3Z	-405471232	E7D50000	M13D.2D	-281149440	EF3E0000
MD.13DESZ	-403439616	E7F40000	M13D.3D	-281214976	EF3D0000
MD.13DES2Z	-404488192	E7E40000	M14D	-282066944	EF300000
MD.13D3S3Z	-405536782	E7D40000	M14D.2D	-282198016	EF2E0000
MD.14DESZ	-403505152	E7F30000	M14D.2D	-282198016	EF2E0000
MD.14DES2Z	-404553728	E7E30000	M15D	-283115520	EF200000
MD.14DES3Z	-405602304	E7D30000	M15D.D	-283181056	EF1E0000
MD.15D	-269418496	EFF10000	M16D	-284164096	EF100000
MD.15DESZ	-403570688	E7F20000			
MD.15DES2Z	-404619264	E7E20000			
MD.15DES3Z	-405667840	E7D20000			
MD.16DESZ	-403636224	E7F10000			
MD.16DES2Z	-404684800	E7E10000			
MD.16DES3Z	-405733376	E7D10000			

User-Defined PICTURE Word

Values of the PICTURE word in the range 1 through 2 147 483 647 are treated the same as 0 by QUERY. This range is reserved for user defined interpretations.

bits	31	26 thru 00
field	–	User - Defined
range	0	X

Appendix T

Useful Tables

HP 9000 Character Codes

The following tables list the characters available on the HP 9000 internal printer and display. The representation on an external printer or terminal may differ. For each character, the tables show its graphic representation on the printer, its description, and its binary, octal, decimal, and hexadecimal code value.

If it is a control character, it may not print or display unless the DISPLAY FUNCTIONS mode of the device is enabled. The ASCII control characters can be generated by simultaneously pressing **CTRL**, **CNTL** or **CONTINUE** and the key listed in the CTRL Char column of the table. On most HP terminals, the BEL character is represented as a different symbol.

HP 9000 ASCII Control Character Codes

Graphic	CTRL Char	ASCII Character Abbreviation, Description	Binary	Oct	CHR\$ Dec	Hex
NUL	@	NUL, Null	00000000	000	000	00
SOH	A	SOH, Start Of Header	00000001	001	001	01
STX	B	STX, Start Of Text	00000010	002	002	02
ETX	C	ETX, End Of Text	00000011	003	003	03
EOT	D	EOT, End Of Transmission	00000100	004	004	04
ENQ	E	ENQ, Enquiry	00000101	005	005	05
ACK	F	ACK, Acknowledge	00000110	006	006	06
BEL	G	BEL, Bell, attention signal	00000111	007	007	07
BS	H	BS, Backspace	00001000	010	008	08
HT	I	HT, Horizontal Tab	00001001	011	009	09
LF	J	LF, Line-feed	00001010	012	010	0A
VT	K	VT, Vertical Tab	00001011	013	011	0B
FF	L	FF, Form-feed	00001100	014	012	0C
CR	M	CR, Carriage-return	00001101	015	013	0D
SO	N	SO, Shift Out	00001110	016	014	0E
SI	O	SI, Shift In	00001111	017	015	0F
DLE	P	DLE, Data Link Escape	00010000	020	016	10
DC1	Q	DC1, Device Control 1, X-ON	00010001	021	017	11
DC2	R	DC2, Device Control 2	00010010	022	018	12
DC3	S	DC3, Device Control 3, X-OFF	00010011	023	019	13
DC4	T	DC4, Device Control 4	00010100	024	020	14
NAK	U	NAK, Negative Acknowledge	00010101	025	021	15
SYN	V	SYN, Synchronous idle	00010110	026	022	16
ETB	W	ETB, End Transmission Block	00010111	027	023	17
CAN	X	CAN, Cancel	00011000	030	024	18
EOM	Y	EOM, End Of Medium	00011001	031	025	19
SUB	Z	SUB, Substitute	00011010	032	026	1A
ESC	[	ESC, Escape	00011011	033	027	1B
FS	\	FS, File Separator	00011100	034	028	1C
GS	]	GS, Group Separator	00011101	035	029	1D
RS	^	RS, Record Separator	00011110	036	030	1E
US	_	US, Unit Separator	00011111	037	031	1F

HP 9000 ASCII Displayable Character Codes

Graphic	ASCII Character Description	Binary	Oct	CHR# Dec	Hex
	Space, Blank	00100000	040	032	20
!	Exclamation point	00100001	041	033	21
"	Quotation mark	00100010	042	034	22
#	Pound sign, Number sign	00100011	043	035	23
\$	Dollar sign	00100100	044	036	24
%	Percent sign	00100101	045	037	25
&	Ampersand	00100110	046	038	26
'	Apostrophe, Acute accent	00100111	047	039	27
(	Opening parenthesis	00101000	050	040	28
)	Closing parenthesis	00101001	051	041	29
*	Asterisk, Star	00101010	052	042	2A
+	Plus	00101011	053	043	2B
,	Comma, Cedilla	00101100	054	044	2C
-	Hyphen, Minus, Dash	00101101	055	045	2D
.	Period, Decimal Point	00101110	056	046	2E
/	Slant, Slash	00101111	057	047	2F
0	Zero	00110000	060	048	30
1	One	00110001	061	049	31
2	Two	00110010	062	050	32
3	Three	00110011	063	051	33
4	Four	00110100	064	052	34
5	Five	00110101	065	053	35
6	Six	00110110	066	054	36
7	Seven	00110111	067	055	37
8	Eight	00111000	070	056	38
9	Nine	00111001	071	057	39
:	Colon	00111010	072	058	3A
;	Semicolon	00111011	073	059	3B
<	Less than	00111100	074	060	3C
=	Equals	00111101	075	061	3D
>	Greater than	00111110	076	062	3E
?	Question mark	00111111	077	063	3F
@	Commercial at	01000000	100	064	40
A	Uppercase A	01000001	101	065	41
B	Uppercase B	01000010	102	066	42
C	Uppercase C	01000011	103	067	43
D	Uppercase D	01000100	104	068	44
E	Uppercase E	01000101	105	069	45
F	Uppercase F	01000110	106	070	46
G	Uppercase G	01000111	107	071	47
H	Uppercase H	01001000	110	072	48
I	Uppercase I	01001001	111	073	49
J	Uppercase J	01001010	112	074	4A
K	Uppercase K	01001011	113	075	4B
L	Uppercase L	01001100	114	076	4C
M	Uppercase M	01001101	115	077	4D
N	Uppercase N	01001110	116	078	4E
O	Uppercase O	01001111	117	079	4F

HP 9000 ASCII Displayable Character Codes (con't)

Graphic	ASCII Character Description	Binary	Oct	CHR\$ Dec	Hex
P	Uppercase P	01010000	120	080	50
Q	Uppercase Q	01010001	121	081	51
R	Uppercase R	01010010	122	082	52
S	Uppercase S	01010011	123	083	53
T	Uppercase T	01010100	124	084	54
U	Uppercase U	01010101	125	085	55
V	Uppercase V	01010110	126	086	56
W	Uppercase W	01010111	127	087	57
X	Uppercase X	01011000	130	088	58
Y	Uppercase Y	01011001	131	089	59
Z	Uppercase Z	01011010	132	090	5A
[	Opening bracket	01011011	133	091	5B
\	Reverse slant, Backslash	01011100	134	092	5C
]	Closing bracket	01011101	135	093	5D
^	Circumflex, Caret, Hat	01011110	136	094	5E
_	Underscore	01011111	137	095	5F
`	Grave accent	01100000	140	096	60
a	Lowercase a	01100001	141	097	61
b	Lowercase b	01100010	142	098	62
c	Lowercase c	01100011	143	099	63
d	Lowercase d	01100100	144	100	64
e	Lowercase e	01100101	145	101	65
f	Lowercase f	01100110	146	102	66
g	Lowercase g	01100111	147	103	67
h	Lowercase h	01101000	150	104	68
i	Lowercase i	01101001	151	105	69
j	Lowercase j	01101010	152	106	6A
k	Lowercase k	01101011	153	107	6B
l	Lowercase l	01101100	154	108	6C
m	Lowercase m	01101101	155	109	6D
n	Lowercase n	01101110	156	110	6E
o	Lowercase o	01101111	157	111	6F
p	Lowercase p	01110000	160	112	70
q	Lowercase q	01110001	161	113	71
r	Lowercase r	01110010	162	114	72
s	Lowercase s	01110011	163	115	73
t	Lowercase t	01110100	164	116	74
u	Lowercase u	01110101	165	117	75
v	Lowercase v	01110110	166	118	76
w	Lowercase w	01110111	167	119	77
x	Lowercase x	01111000	170	120	78
y	Lowercase y	01111001	171	121	79
z	Lowercase z	01111010	172	122	7A
<	Opening (left) brace	01111011	173	123	7B
	Vertical line	01111100	174	124	7C
>	Closing (right) brace	01111101	175	125	7D
~	Tilde	01111110	176	126	7E
⌘	DEL, Delete, Rubout	01111111	177	127	7F

HP 9000 BASIC Extension Control Character Codes

The following characters are fully supported only in the BASIC Language System on the HP 9000 Model 520. In the HP-UX operating system, these characters print and display as blank. On external printers and displays. These codes may be interpreted as ASCII characters with one's parity, as spaces, nil, or characters from the device's own extension set.

Graphic	HP9000 Character Description	Binary	Oct	CHR\$ Dec	Hex
␣	Clear all enhancements	10000000	200	128	80
␣	Begin Inverse Video	10000001	201	129	81
␣	Begin Blinking	10000010	202	130	82
␣	Begin Inverse Blinking	10000011	203	131	83
␣	Begin Underline	10000100	204	132	84
␣	Begin Underline Inverse	10000101	205	133	85
␣	Begin Underline Blinking	10000110	206	134	86
␣	Begin Underline Inv. Blnk.	10000111	207	135	87
␣	Begin Color White	10001000	210	136	88
␣	Begin Color Red	10001001	211	137	89
␣	Begin Color Yellow	10001010	212	138	8A
␣	Begin Color Green	10001011	213	139	8B
␣	Begin Color Cyan	10001100	214	140	8C
␣	Begin Color Blue	10001101	215	141	8D
␣	Begin Color Magenta	10001110	216	142	8E
␣	Begin Color Black	10001111	217	143	8F
␣	Reserved for future use	10010000	220	144	90
␣	Reserved for future use	10010001	221	145	91
␣	Reserved for future use	10010010	222	146	92
␣	Reserved for future use	10010011	223	147	93
␣	Reserved for future use	10010100	224	148	94
␣	Reserved for future use	10010101	225	149	95
␣	Reserved for future use	10010110	226	150	96
␣	Reserved for future use	10010111	227	151	97
␣	Reserved for future use	10011000	230	152	98
␣	Reserved for future use	10011001	231	153	99
␣	Reserved for future use	10011010	232	154	9A
␣	Reserved for future use	10011011	233	155	9B
␣	Reserved for future use	10011100	234	156	9C
␣	Reserved for future use	10011101	235	157	9D
␣	Reserved for future use	10011110	236	158	9E
␣	Reserved for future use	10011111	237	159	9F

HP 9000 BASIC Extension Displayable Character Codes

The following characters in the range of codes from 160 through 222 are fully supported only by the BASIC Language System. HP terminals and printers which have their "Roman Extension" character set and "8-bit" mode enabled may correctly represent only some of these characters.

Graphic	HP9000 Character Description	Binary	Oct	CHR\$ Dec	Hex
	Space	10100000	240	160	A0
À	Uppercase A accent acute	10100001	241	161	A1
Í	Uppercase I accent acute	10100010	242	162	A2
Ó	Uppercase O accent acute	10100011	243	163	A3
Ú	Uppercase U accent acute	10100100	244	164	A4
Ä	Uppercase A accent grave	10100101	245	165	A5
Ê	Uppercase E accent grave	10100110	246	166	A6
Ô	Uppercase O accent grave	10100111	247	167	A7
´	Acute accent mark	10101000	250	168	A8
`	Grave accent mark	10101001	251	169	A9
^	Circumflex	10101010	252	170	AA
¨	Umlaut, trema, diaeresis	10101011	253	171	AB
~	Tilde	10101100	254	172	AC
É	Uppercase E circumflex	10101101	255	173	AD
Ö	Uppercase O circumflex	10101110	256	174	AE
£	Line currency symbol	10101111	257	175	AF
¯	Overbar, macron	10110000	260	176	B0
Ā	Uppercase A tilde (macron)	10110001	261	177	B1
ā	Lowercase a tilde (macron)	10110010	262	178	B2
°	Degrees, superscript 0	10110011	263	179	B3
Ç	Uppercase C cedilla	10110100	264	180	B4
ç	Lowercase c cedilla	10110101	265	181	B5
Ñ	Uppercase N tilde (macron)	10110110	266	182	B6
ñ	Lowercase n tilde (macron)	10110111	267	183	B7
¡	Leading exclamation	10111000	270	184	B8
¿	Leading question mark	10111001	271	185	B9
¤	Currency symbol	10111010	272	186	BA
£	Pound currency symbol	10111011	273	187	BB
²	Superscript underlined o	10111100	274	188	BC
§	Paragraph, section mark	10111101	275	189	BD
²	2 superscript	10111110	276	190	BE
+	Small +, 5x7 cell centered	10111111	277	191	BF
â	Lowercase a circumflex	11000000	300	192	C0
ê	Lowercase e circumflex	11000001	301	193	C1
ô	Lowercase o circumflex	11000010	302	194	C2
û	Lowercase u circumflex	11000011	303	195	C3
à	Lowercase a accent acute	11000100	304	196	C4
é	Lowercase e accent acute	11000101	305	197	C5
ó	Lowercase o accent acute	11000110	306	198	C6
ú	Lowercase u accent acute	11000111	307	199	C7
à	Lowercase a accent grave	11001000	310	200	C8
è	Lowercase e accent grave	11001001	311	201	C9
ò	Lowercase o accent grave	11001010	312	202	CA
ù	Lowercase u accent grave	11001011	313	203	CB
ä	Lowercase a umlaut	11001100	314	204	CC
ë	Lowercase e umlaut	11001101	315	205	CD
ö	Lowercase o umlaut	11001110	316	206	CE
ü	Lowercase u umlaut	11001111	317	207	CF
Å	Uppercase Scand. vowel ~AA	11010000	320	208	D0

HP 9000 BASIC Extension Displayable Character Codes

The characters in the code range 223 through 255 are fully supported only in the HP 9000 Model 520 BASIC Language System.

Graphic	HP9000 Character Description	Binary	Oct	CHR\$ Dec	Hex
ï	Lowercase i circumflex	11010001	321	209	D1
ø	Uppercase Scand. vowel wOE	11010010	322	210	D2
œ	Uppercase Scand. vowel wAE	11010011	323	211	D3
ä	Lowercase Scand. vowel waa	11010100	324	212	D4
í	Lowercase i accent acute	11010101	325	213	D5
ø	Lowercase Scand. vowel woe	11010110	326	214	D6
æ	Lowercase Scand. vowel wae	11010111	327	215	D7
Ä	Uppercase A umlaut	11011000	330	216	D8
ï	Lowercase i accent grave	11011001	331	217	D9
Ö	Uppercase O umlaut	11011010	332	218	DA
Ü	Uppercase U umlaut	11011011	333	219	DB
É	Uppercase E accent acute	11011100	334	220	DC
ï	Lowercase i umlaut	11011101	335	221	DD
ß	Sharp S, lowercase ss	11011110	336	222	DE
	Reserved	11011111	337	223	DF
↑	Up arrow	11100000	340	224	E0
┃	Bold vertical & left end	11100001	341	225	E1
┃┃	Bold top left corner	11100010	342	226	E2
┃┃┃	Bold horizontal & top end	11100011	343	227	E3
┃┃┃┃	Bold horizontal	11100100	344	228	E4
┃┃┃┃┃	Bold top right corner	11100101	345	229	E5
┃┃┃┃┃┃	Bold vertical & right end	11100110	346	230	E6
┃┃┃┃┃┃┃	Bold vertical	11100111	347	231	E7
┃┃┃┃┃┃┃┃	Bold horiz. Fine vertical	11101000	350	232	E8
┃┃┃┃┃┃┃┃┃	Fine vertical & left end	11101001	351	233	E9
┃┃┃┃┃┃┃┃┃┃	Fine top left corner	11101010	352	234	EA
┃┃┃┃┃┃┃┃┃┃┃	Fine horizontal & top end	11101011	353	235	EB
┃┃┃┃┃┃┃┃┃┃┃┃	Fine horizontal	11101100	354	236	EC
┃┃┃┃┃┃┃┃┃┃┃┃┃	Fine top right corner	11101101	355	237	ED
┃┃┃┃┃┃┃┃┃┃┃┃┃┃	Fine vertical & right end	11101110	356	238	EE
┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃	Bold vert. Fine horiz.	11101111	357	239	EF
→	Right arrow	11110000	360	240	F0
┃┃┃┃┃┃┃┃┃┃┃┃┃┃	Bold vert. Fine left end	11110001	361	241	F1
┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃	Fine bottom left corner	11110010	362	242	F2
┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃	Fine horiz. & bottom end	11110011	363	243	F3
┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃	Fine horizontal & vertical	11110100	364	244	F4
┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃	Fine bottom right corner	11110101	365	245	F5
┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃	Bold vert. Fine right end	11110110	366	246	F6
↓	Down arrow	11110111	367	247	F7
←	Left arrow	11111000	370	248	F8
┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃	Bold horiz. Fine bottom end	11111001	371	249	F9
┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃	Bold bottom left corner	11111010	372	250	FA
┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃	Bold horiz. & bottom end	11111011	373	251	FB
┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃	Bold vertical & horizontal	11111100	374	252	FC
┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃	Bold lower right corner	11111101	375	253	FD
┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃┃	Bold horiz. Fine top end	11111110	376	254	FE
※	Smudge, ※ raster complement	11111111	377	255	FF

Prime Numbers

Here is a table of selected prime numbers between 37 and 25250003. Space does not permit a complete list of all the primes in this range. If your master data set's capacity is not near one of these primes, use the program listed below for generating primes.

37	67	101	131	211	257
307	401	521	601	701	809
907	1031	1511	2053	3001	4099
5003	6007	7001	8209	9001	10007
12007	14009	16411	18013	20011	25013
30011	40009	50021	65537	80021	100003
131101	150001	20003	262147	300007	350003
400009	524309	700001	900001	10448593	1500007
2097169	3000017	4194319	5000011	83888617	11000027
12500003	16777259	18000041	20000003	22500011	25250003

PRIME-FINDER Program

```

10 !Program PRIME_FIND: Next Prime number [RJN] <821014.1319>
20 !
30 ! This Program requests your master data set's estimated
40 ! capacity and displays the nearest Prime number greater
50 ! than that capacity.
60 !
70 Start:DOUBLE Capacity,Test,Prime,Limit
80 !
90 INPUT "Enter desired CAPACITY (or 0 to STOP)",Capacity
100 IF Capacity<0 THEN GOTO Start
110 IF Capacity=0 THEN GOTO Stop
120 !
130 Prime=Capacity !For low numbers,
140 IF Capacity<4 THEN GOTO Prime !capacity is always Prime
150 !
160 FOR Prime=Capacity TO 2147483647 !For cap to max_double
170 IF (Prime MOD 2) THEN !If not an even number..
180 Limit=SQR(Capacity) !Loop limit
190 FOR Test=3 TO Limit STEP 2 !For 3 to sqrt(cap)....
200 IF NOT (Prime MOD Test) THEN GOTO Next_Prime
210 NEXT Test
220 GOTO Prime !Not divisible, must be Prime
230 END IF
240 Next_Prime:NEXT Prime
250 !
260 Stop:PRINT "PRIME_FIND: END"
270 STOP
280 !
290 Prime:PRINT "CAPACITY =" ;Capacity; ", Nearest Prime =" ;Prime
300 GOTO Start
310 !
320 END

```

Determining File Sizes

You can use the following formulas to determine the size of IMAGE media records and files. All quantities are in bytes. All file sizes are rounded up to the nearest multiple of one sector. In the BASIC Language System, a sector is always 256 bytes. In HP-UX a sector defaults to 256 bytes.

These formulas are only approximations. Use them to estimate the disc space required for your data base. You can determine the exact file size after they are created. In the BASIC Language System the CAT statement lists the record length and number of records in each file.

ROOT File Directory

$$\begin{aligned}
 & 24 \text{ bytes for the ROOT file} \\
 & + 24 \text{ bytes for each DSET file} \\
 & + 120 \text{ bytes for spare directory entries (primarily for QUERY)} \\
 & \hline
 & = \text{directory size before sector rounding}
 \end{aligned}$$

The root directory size is in the range 256 through 1792 bytes after rounding. If you create additional files in the directory, the directory is extended if necessary.

ROOT File

$$\begin{aligned}
 & 944 \text{ bytes (includes some internal sector round-up)} \\
 & + 232 \text{ bytes per data set volume} \\
 & + 72 \text{ bytes per data set} \\
 & + 40 \text{ bytes per item} + 8 \text{ bytes per item occurrence in each set} \\
 & + 12 \text{ bytes per array item} \\
 & + 8 \text{ bytes per array dimension } 2 \dots 6 \\
 & + 24 \text{ bytes per numeric item for NULL/RANGE (worst case)} \\
 & + 16 \text{ bytes per CODE value} \\
 & + 16 \text{ bytes per PSEUDONYM} \\
 & + 4 \text{ bytes per path per set} \\
 & \hline
 & = \text{ROOT file size before rounding}
 \end{aligned}$$

The ROOT file size is in the range 1280 through 701696 bytes after rounding.

Data Set File Directory

Data sets which are not in the ROOT file's directory have their own directory. This directory has the same name as the data base.

$$\begin{aligned}
 & + 24 \text{ bytes per data set (DSET) in the directory} \\
 & + 96 \text{ bytes (spare directory entries) for QUERY} \\
 & \hline
 & = \text{directory size before rounding}
 \end{aligned}$$

The set directory size is in the range 256 through 1792 bytes after rounding. If you create other files in this directory, it is extended as necessary.

Master Data Set

$$\begin{array}{r}
 \text{ENTRY length (sum of data ITEM sizes)} \\
 + 4 \text{ bytes for hash field} \\
 + 8 \text{ bytes for free/synonym chain pointers} \\
 + 12 \text{ bytes for each path} \\
 \hline
 = \text{media record size} \\
 \\
 \text{media record size} \\
 \times \text{master set capacity} \\
 \hline
 = \text{Master data set file size before rounding}
 \end{array}$$

The minimum master data set file size after rounding is 256 bytes. The maximum file size is limited by disc capacity.

Linked Detail Data Sets

$$\begin{array}{r}
 \text{greater of: ENTRY length, or 8 bytes} \\
 + 4 \text{ bytes for free/empty record flag.} \\
 + 8 \text{ bytes for each path} \\
 \hline
 = \text{media record size} \\
 \\
 \text{media record size} \\
 \times \text{detail set capacity} \\
 \hline
 = \text{DSET file size before rounding}
 \end{array}$$

The minimum detail data set file size after rounding is 256 bytes. The maximum file size is limited by disc capacity.

Stand Alone Detail Data Sets

$$\begin{array}{r}
 \text{media record size} \\
 \times \text{detail set capacity} \\
 \hline
 = \text{DSET file size before rounding}
 \end{array}$$

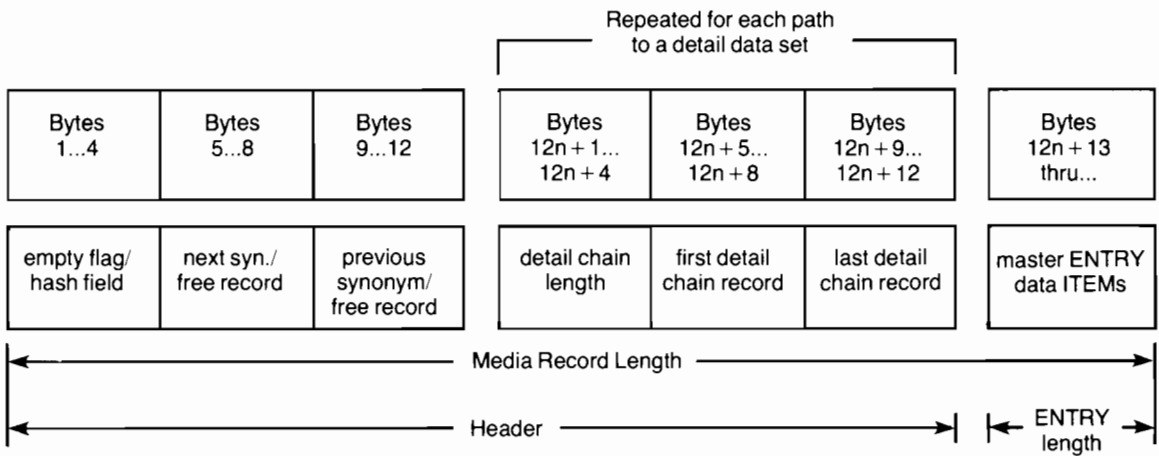
The minimum detail data set file size after rounding is 256 bytes. The maximum file size is limited by disc capacity.

Media Records

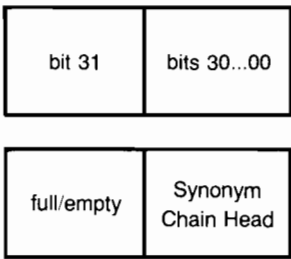
The following sections describe the structure of master and detail data set media records. These structures are described in terms of fields. A field is one or more contiguous bits, bytes, or double integer locations in the media record. There is a description for each field except the ENTRY data ITEMS field. This field is your data. Its structure is the simply the binary representation of all the data items in the entry. The maximum length of all media records is 2 147 483 647 bytes, or the maximum capacity of the storage medium, whichever is less.

Master Data Set Records

The media record structure for Automatic and Manual Master Data Sets is illustrated below. The formulas for byte locations are based on n paths from the master to linked detail sets. If it is a stand-alone master, the detail chain fields are not present.



The Hash Field occupies 31 bits of a double integer. The sign bit (bit 31) is the full/empty indicator. The bit is 0 if this record is empty of data, and 1 if it is full.



The synonym chain head is a 32 bit unsigned double integer. The sign bit is ignored. This is the record number of the first master set record in the chain of master data set entries which are synonyms of this entry. If there are no synonyms, or this record is the head of the chain, this field contains its own record number. It is possible for an empty entry to have synonyms.

The synonym/free fields are double integer record numbers. If the record is empty, and has no synonyms, the synonym/free fields are the record numbers of the next and previous records in the free record chain. If this record is the last or first record in that chain, the next or previous field is zero.

The detail chain fields are omitted if the set is a stand-alone master.

If the set is linked to any detail sets (has paths), there are three 4-byte (double integer) fields for each path. The detail chain length is the number of entries in the detail set which have a search item value which is the same as this master entry's key item value. The chain length can be zero if this is a manual master data set.

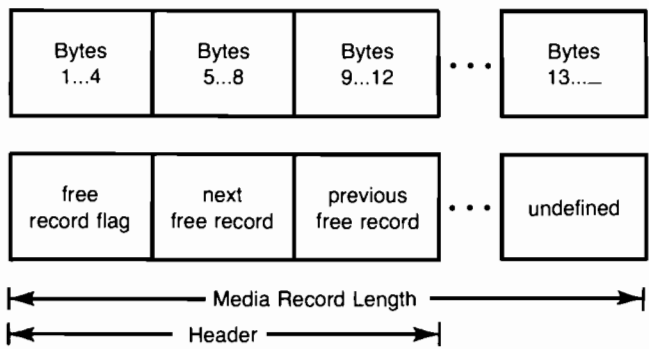
If the detail set has one or more entries on this chain, the first and last detail chain fields contain the record numbers of the first (head) and last entries on the chain. If the chain is empty (length = 0), these fields are undefined.

Detail Data Set Records, Chained

This section illustrates structures for media records in a detail data set having n paths to related master sets.

Empty Records

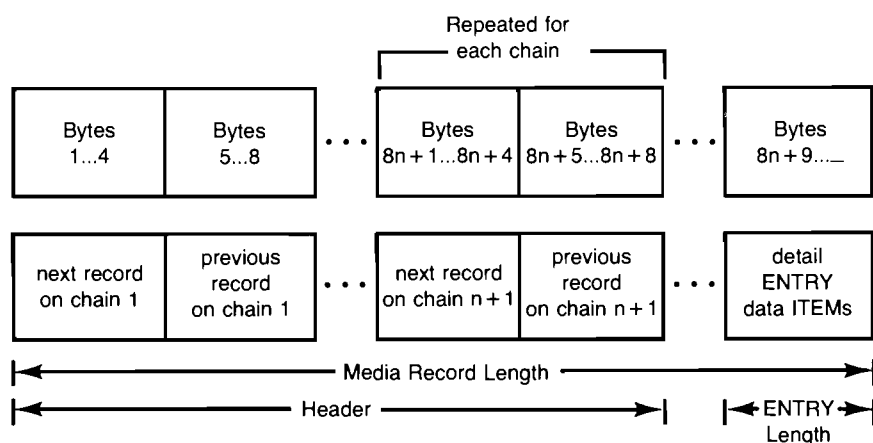
The following diagram is for a record on the set's free record chain. A pointer in the root file stores the number of the highest occupied record in each detail set. Records above this position have no structure.



Empty Chained Detail Record

If this record is the last or first record in that chain, the next or previous field is zero. All other chain fields are undefined.

Occupied Records



Occupied Chained Detail Record

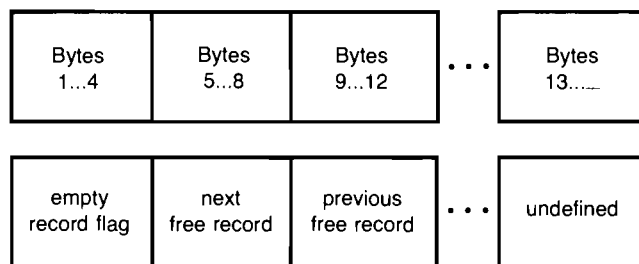
The next chain fields point to the next record along each chain. If this is the last record along any chain, that chain's next field is zero. If the record is occupied, the previous chain fields point to the previous record along each chain. If this is the first record along any chain, that chain's next field is zero.

Detail Data Set Records (stand_alone)

This section illustrates record structures for Detail Data Sets which have no paths to a master set.

Empty Records

The following diagram is for a record on the data set's free record chain. A pointer in the root file stores the number of the highest occupied record in each detail set. Records above this position have no structure.



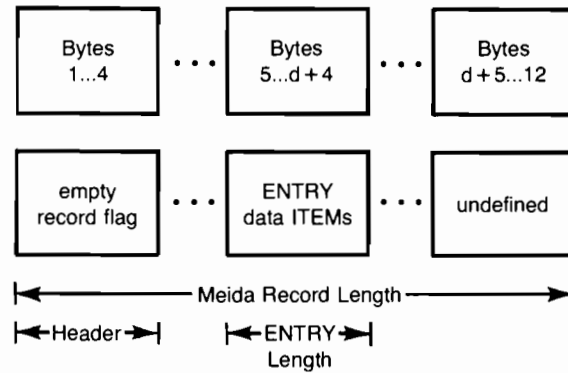
Empty Stand-alone Detail Record

The free record flag in a empty record is -2 147 483 648. The next and previous free record fields point to the next and previous records in the free record chain. If this is the last or first free record, the corresponding field is zero. These fields over-write the ENTRY or chain data in the record.

If the record was the highest occupied record when deleted, this structure does not apply. The highest occupied pointer is decremented, and the record structure is undefined.

Occupied Records

The formulas for byte locations are based on d data bytes in the ENTRY data ITEMS portion of the media record.

**Occupied Stand-alone Detail Record**

The free record flag in an occupied record is zero. Due to the structure of deleted records, the media record must be at least 12 bytes long. Therefore, if the ENTRY length is 7 bytes or less in length, there are undefined trailing bytes in the record.

Appendix U

Upgrading From IMAGE/45 to IMAGE/9000

Introduction

You cannot access an IMAGE/45 data base from an IMAGE/9000 program. If you have an IMAGE/45 data base which you want to convert to IMAGE/9000, you must unload the data from the IMAGE/45 data base, transport the data to your HP 9000 and load the data into your IMAGE/9000 data base.

In the HP 9000 BASIC Language System, the IMAGE product includes two utility programs for performing this task. The first program, DBXPRT, runs on the HP 9845B/C, and unloads the data into 9845 DATA files. The second program, DBIMPORT, runs on the HP 9000. The HP 9000 supports the 9845 directory format and the DBIMPORT program accesses the unloaded data files as type BCD. If you do not have a disc supported by both systems, you can transfer the data electronically.

The rest of this Appendix discusses the use of the DBXPRT and DBIMPORT programs.

The Export/Import Utilities

In general, you can only use DBXPRT and DBIMPORT if there is an exact structural correspondence between the IMAGE/45 and the IMAGE/9000 data bases. You can change the type of numeric items, such as INTEGER to DOUBLE.

One data type must be changed. In IMAGE/45, a data item designated as a DATE is a SHORT real. In IMAGE/9000 the QUERY type DATE must be a long REAL. The DBXPRT/DBIMPORT programs automatically perform the data conversion, but DBIMPORT expects the item to be type REAL in the HP 9000 data base.

DBXPRT/DBIMPORT also automatically determines if a numeric value is a QUERY/45 null value and converts it to the NULL value defined in the IMAGE/9000 data base.

DBXPRT Instructions

1. The DBXPRT program is supplied in a BCD file on an SDF flexible disc. To use the program on your HP 9845, you need to transfer the program to a 9845 format disc. To initialize a disc in 9845 format you need to load the "9845_DISC" BIN option file. If you are using an external flexible disc as the transport medium, you also need to load the "HPIB_FLEX" or "HP9885" BIN option files.
2. Once you have an initialized 9845 format disc, transfer the DBXPRT program using the COPY command. Do not use a GET/SAVE or GET/STORE. Some of the DBXPRT program does not syntax correctly in HP 9000 BASIC and those lines are changed to comments during a GET. Also, the SAVE and STORE commands do not create a BCD file, although RE-SAVE and RE-STORE can access BCD files.
3. Mount the 9845 format disc in a disc drive of your HP 9845 system and load the DBXPRT program with a GET command.
4. Insert an initialized 9845 format disc in the drive to be used for the unload files and RUN the DBXPRT program. It is interactive and asks you for the name of the IMAGE/45 data base and a read password. It lists the data sets worth exporting and ask you which to export. You also designate the names and locations of the destination file names.
5. DBXPRT opens each data set and writes the entries to the set's unload file. The DBXPRT program estimates the size of file needed to export the data and ask you to verify this size. You can make the file larger or smaller. If the file is not large enough, DBXPRT asks you for another file name to continue exporting.

The listing for the DBXPRT program contains information on the structure of the unloaded data files.

DBIMPORT Instructions

1. Using SCHEMA or QUERY/DEFINE, define the HP 9000 data base and create its data sets with DBCREATE.
2. If not already loaded, load the option BIN file "9845_DISC" or a later version. If necessary, load the "HP9885" or "HPIB_FLEX" driver for the disc to be used.
3. Load the DBIMPORT program with a GET command.
4. Mount the 9845 format disc in a disc drive of your HP 9000 system.
5. RUN the DBIMPORT program. It is interactive and asks you for the name of the IMAGE/9000 data base and a write password. It lists the data sets and asks you for the file names of the unloaded data files. You should load the manual master data sets first.
6. DBIMPORT opens each data file and data set and writes the data to the set.
7. If the data base structures do not match, you can modify the DBEXPRT or DBIMPORT programs.

Subject Index

a

Accessing Data Entries 1-28
Accessing the Packing String B-4
Adding Data 3-14
Array Data Type 1-17
Attributes 1-13
Automatic Master 1-25, 1-26

b

Backup 4-3
Backup Techniques 4-5
BASIC 1-1
BASIC Language B-1
BASIC Language Examples B-7
BASIC Schema Example 2-19
BASIC Schema Process 2-18
BASIC Systems 1-34

c

C 1-1
C Language C-1
C Language Examples C-2
C Library Calls C-1
Calculated Access 1-30
Capacities 5-2
Chained Access 1-29
Chained Reads 3-18
Changing the Current Path 3-17
Character Codes T-1
Closing the Data Base 3-6
CODE Data Type 1-17
CODEs 3-28
Collisions 1-12
Compatibility 1-1
Concepts 1-8
Coordinating Access 3-24
Corrupt Data Base 1-33, 4-9
Creating the Data Base 2-1
Creating the Data Set Files 2-25
Creating the ROOT File 2-19, 2-23
Creating the Schema 2-1, 2-2
Current Path, Changing 3-17
Current Record 3-21

d

Daily Use 1-7
Data Base 1-2, 1-23
Data Base Access 3-1
Data Base Backup 1-33
Data Base Creating 2-1
Data Base Definition 5-1
Data Base Design 1-2
Data Base Diagram 1-27
Data Base Directory 1-30
Data Base Files 1-30
Data Base Maintenance 4-1
Data Base Number 3-3
Data Base Operations 5-1
Data Base Part 2-5
Data Base Restructure 4-11
Data Base ROOT File 1-30
Data Base Variable 3-2
Data Chain 1-21
Data Entry 1-19
Data Item 1-15
Data Path 1-24
Data Records 1-9
Data Security 1-32
Data Set 1-19
Data Set Files 1-31
Data Set Types 1-23
Data Size 1-9
Data Storage 1-17
Data Types 1-15, 1-9
DBCLOSE 3-1, 3-6
DBCLOSE Modes 3-6
DBDELETE 3-1, 3-22
DBEXTEND 4-11
DBFIND 3-1, 3-16
DBGET 3-1, 3-18
DBINFO 3-1, 3-28
DBLOCK 3-1, 3-24
DBOPEN 3-1, 3-3
DBOPEN Modes 3-5
DBOPEN Password 3-4
DBPUT 3-1, 3-14
DBPUT in a Detail Data Set 3-15
DBPUT in a Manual Master 3-14
DBSTATUS 4-2
DBUNLOCK 3-1, 3-24
DBUPDATE 3-23
Deleting a Data Entry 3-22
Deleting a Detail Data Set 3-22

Deleting a Master Data Set	3-22
Determining File Size	T-8
Direct Access	1-28
Directory Part	2-6

e

Entity	1-13
ERRORS	2-16
Errors	E-1

f

File Management	1-9
File Names	1-31
File Size, Determining	T-8
Force Buffer Posting	3-9
Forced Close	3-7
FORTRAN	1-1
Fortran Language	F-1
Fortran Language Example	F-3
Fortran Library Calls	F-1
Frequency of Backups	4-6

g

Getting Started in BASIC	1-7
Getting Started in HP-UX	1-8
Glossary	G-1

h

Hashing	1-11
High Performance Access	5-4
HP-UX	1-1
HP-UX Schema Example	2-23
HP-UX Schema Process	2-23
HP-UX Systems	1-34

i

Identifying the Data Items	3-9
IMAGE Errors	E-2
IMAGE/9000	1-1

IMAGE_DBM	1-7
Increasing Data Set Capacity	4-11
Indexed Access	1-10
Installing QUERY	1-8
Integer Data Type	1-15
Invoking Statements, BASIC	B-2
Item List Mode	3-10
Item Part	2-8

k

Key Items	1-19
Keyed Access	1-10

l

Language Considerations	3-2
Lexical Order	1-22
LINES	2-16
LIST	2-16
List Specifier	3-9
Locating a Chain	3-16
Locking with Wait	3-25

m

Maintenance	4-1
Manual Master	1-25, 1-26
Master Data Sets	1-25
Minimizing Processing	5-4
Modifying Data	3-23
Mutiple-keyed Access	1-11

n

NO LIST	2-16
NOROOT	2-16
NOTABLE	2-16
NULL Extension	1-17

o

Opening the Data Base	3-3
Overview	1-3

p

PACK USING	B-6
PACKFMT	B-5
Parameter Data Types	3-2
Parameters	B-2
Partial Close	3-8
Pascal	1-1
Pascal Aliasing	P-2
Pascal Language	P-1
Pascal Language Example	P-3
Pascal Library Calls	P-1
Password Part	2-7
Passwords	1-31
Permission Bits	1-31
PICTURE Extension	1-18
PICTUREs	3-30
Primary Address	1-12
Primary Path	3-17
Prime Numbers	T-7
Program Variables	3-12
PSEUDONUM Extension	1-18
PSEUDONYMs	3-30

q

QUERY	Q-1
QUERY/DEFINE Errors	E-3

r

Random Access	1-10
RANGE	3-30
RANGE Extension	1-18
Re-entry of Data	4-3
Reading a Key Item's Entry	3-19
Reading Data	3-9
Reading Every Entry	3-20
Reading the ROOT File	3-28
Real Data Type	1-16
Records	1-9
Recovery of Corrupt Data Base	4-9
Relating Data	1-13
Representing Data	1-14
Reserved Words	2-3
Restructuring a Data Base	4-11
Retrieving Data	3-18
Rewind	3-20
Rewind a Data Set	3-8
Rewinding a Data Set	1-29
ROOT	2-16

s

Sample Data Base	1-4
Schema	2-1
Schema Errors	E-3
Schema Example, BASIC	2-19
Schema Example, HP-UX	2-23
Schema Instructions	2-15
Schema Language Conventions	2-2
Schema Process, BASIC	2-18
Schema Process, HP-UX	2-23
Schema Structure	2-2
Schema Syntax	2-4
Search Items	1-20
Secondary Location	1-12
Select	3-20
Serial Access	1-28, 1-9
SET	2-16
Set Part	2-11
Shared Data Base Access	1-32
Software Installation	B-3
SORT Items	1-22
Specifying the Data Base	3-4
Status Array	3-2
Status Errors	E-1
Storing a Data Set	1-22
Storing Data Entries	1-27
String Data Type	1-16
Synonym	1-12

t

TABLE	2-16
Terms	1-4
Type Extensions	1-17

u

UNPACK USING	B-6
Utilities	4-1

v

Validation	4-1
------------------	-----

w

Writing Data	3-9
--------------------	-----

Manual Comment Sheet Instruction

If you have any comments or questions regarding this manual, write them on the enclosed comment sheets and place them in the mail. Include page numbers with your comments wherever possible.

If there is a revision number, (found on the Printing History page), include it on the comment sheet. Also include a return address so that we can respond as soon as possible.

The sheets are designed to be folded into thirds along the dotted lines and taped closed. Do not use staples.

Thank you for your time and interest.