



AT&T

999-801-3021S

**AT&T UNIX[®] PC
ISAM**

Programmer's Guide

©1985 AT&T
All Rights Reserved
Printed in USA

NOTICE

The information in this document is subject to change without notice.
AT&T assumes no responsibility for any errors that may appear in
this document.

UNIX is a registered trademark of AT&T

PREFACE

This manual describes the AT&T UNIXTM PC Indexed Sequential Access Method (ISAM) software product, running under the UNIX System V Operating System. UNIX PC ISAM accesses fixed-length records in ISAM data sets. ISAM identifies the records using record keys. Programs written in the UNIX PC programming languages Interpreted BASIC and C can use ISAM. UNIX PC ISAM also provides utilities for maintaining ISAM data sets.

This manual is oriented toward the ISAM applications programmer and the ISAM administrator. The manual contains the following sections:

Section 1, Overview, presents the ISAM features.

Section 2, Concepts, describes the important concepts necessary to understand UNIX PC ISAM.

Section 3, ISAM Applications, is an ISAM tutorial. The section contains seven program examples that access ISAM procedures.

Section 4, ISAM Procedures, describes the ISAM procedures in detail.

Section 5, System Administration, presents the background information of interest to the ISAM administrator and describes the utilities designed for ISAM maintenance.

Appendix A, Status Codes, lists the ISAM errors in detail.

Appendix B, Programming Examples, contains three examples of programs that use ISAM.

Installing UNIX PC ISAM from the User Agent

To install ISAM on your UNIX PC, see Installing Optional Application Software in the AT&T UNIXTM PC Installation Guide.

Using UNIX PC ISAM with Interpreted Basic

To make ISAM accessible from Interpreted BASIC, the UNIX PC Development Set software must be installed to relink the BASIC Interpreter. See Section 7 of the AT&T UNIX™ PC BASIC Interpreter Programmer's Guide for more information.

CONTENTS

SECTION 1: OVERVIEW 1-1

 FEATURES 1-1

SECTION 2: CONCEPTS 2-1

 KEYS 2-1

 Simple and Composite Keys 2-1

 Key Types 2-2

 Binary 2-2

 Byte String 2-2

 Character String 2-3

 Integer 2-4

 Long/Short IEEE 2-4

 Long/Short Real 2-4

 Summary 2-4

 INDICES 2-5

 FILE STRUCTURE 2-6

 Data Store Section 2-6

 Index Section 2-6

 ISAM DESCRIPTION BLOCK 2-7

 UNIQUE RECORD IDENTIFIER 2-11

 ISAM HANDLE 2-11

 ISAM PROCEDURES 2-11

 Storing 2-11

 Reading 2-11

 Modifying 2-12

 Deleting 2-12

 DATA SET SHARING 2-12

 Access Modes 2-12

 Transactions 2-14

 Transaction-Related Constraints 2-16

 Locking a Record or Data Set 2-18

 Conflicts and Deadlocks 2-20

 Time-Outs 2-22

 ISAM UTILITIES 2-23

 DATA SECURITY 2-23

 DATA INTEGRITY 2-23

SECTION 3: ISAM APPLICATIONS	3-1
EXAMPLE 1: CREATING SIMPLE DATA SETS	3-1
EXAMPLE 2: CREATING DATA SETS (MULTIPLE INDICES)	3-6
EXAMPLE 3: OPENING A DATA SET	3-7
EXAMPLE 4: STORING A NEW RECORD	3-9
EXAMPLE 5: READING A SINGLE RECORD	3-10
EXAMPLE 6: READING MULTIPLE RECORDS	3-12
EXAMPLE 7: USING COMPOUND INDICES FOR EFFICIENT RETRIEVAL	3-14
SECTION 4: ISAM PROCEDURES	4-1
HEADER FILES	4-2
THE LIBRARY	4-2
FILE NAMES	4-2
NOTE ON PARAMETER NAMES	4-2
STATUS BLOCK	4-3
BeginTransaction	4-4
CloseISAM	4-5
CommitTransaction	4-6
CreateISAM	4-7
DeleteByKeyISAMRecord	4-8
DeleteISAM	4-10
DelISAMRecord	4-11
GetISAMRecords	4-12
GetHoldISAMRecords	4-12
HoldISAMDataSet	4-15
iserror	4-17
ISAMHoldRecord	4-18
ModifyByKeyISAMRecord	4-19
ModifyISAMRecord	4-21
OpenISAM	4-23
QueryTransactionParams	4-25
RdNextISAMRecord	4-26
RdHoldNextISAMRecord	4-26
RdUniqueISAMRecord	4-28
RdHoldUniqueISAMRecord	4-28
ReadISAMRecordByUri	4-30
ReadHoldISAMRecordByUri	4-30
ReleaseDataSet	4-32
ReleaseISAMRecord	4-33
RenameISAM	4-34
SetISAMProtection	4-35
SetTransactionParams	4-36
SetUpLimitsISAMIteration	4-37
SetUpPrefixIteration	4-40
StoreISAMRecord	4-41
VerifyMultiUserISAM	4-42

SECTION 5: SYSTEM ADMINISTRATION	5-1
ADMINISTRATOR'S HOME DIRECTORY	5-1
THE SERVER	5-1
SERVER CONFIGURATION	5-2
Installation Block Size	5-3
Number of Server Buffers	5-3
Maximum Number of User Handles	5-4
Maximum Number of UNIX Open Data Sets	5-4
Number of Open Files Retained	5-4
Maximum Number of Server Clients	5-4
Write-Through Cache Option	5-5
ISAM UTILITY PROGRAMS	5-6
IsamCreate	5-7
IsamProtect	5-8
IsamReorg	5-9
IsamStop	5-10
APPENDIX A: STATUS CODES	A-1
APPENDIX B: PROGRAMMING EXAMPLES	B-1
INDEX	I-1

LIST OF TABLES

Table 2-1.	UNIX PC ISAM Key Types	2-3
Table 2-2.	ISAM Description Block	2-8
Table 2-3.	ISAM Index Specification Block	2-9
Table 2-4.	ISAM Key Types and Programming Language Representations	2-10
Table 2-5.	Restrictions on Multiple Access to a Data Set	2-13
Table 2-6.	Transaction-Related Constraints	2-17
Table 2-7.	Transaction Parameters Block Format	2-22
Table 3-1.	Fields of the Elements of IndexSpec	3-7
Table 3-2.	Fields of Order Line Number	3-11
Table 3-3.	Fields of Item Number Due Date	3-15
Table 4-1.	ISAM Procedures by Function	4-1
Table 4-2.	Status Block Format (pStatusBlockRet Parameter)	4-3
Table 4-3.	Buffer Structure for GetISAMRecords and GetHoldISAMRecords When Records Are Read (46-Byte Records)	4-14
Table 4-4.	MatchKind Words for Record Retrieval	4-38
Table 5-1.	ISAM Utilities	5-6

LIST OF FIGURES

Figure 2-1. Index for a Data Set 2-5
Figure 2-2. Series of Transactions 2-15
Figure 2-3. Examples of Deadlock 2-21

SECTION 1: OVERVIEW

UNIX PC Indexed Sequential Access Method (ISAM) is a library of C routines that access fixed-length records in UNIX data sets, by identifying the records using key fields. Programs written in the UNIX PC programming languages Interpreted BASIC and C can access ISAM routines. UNIX PC ISAM also provides utilities designed to simplify ISAM maintenance and administration.

FEATURES

ISAM includes the following features:

- o Random access to data identified by multiple keys.
- o Up to 20 keys per ISAM data set.
- o Key types for many different representations of character and numeric data.
- o Compatibility with the Motorola 68000 family of processors.
- o Shared or exclusive access to ISAM data sets, with transactions and record or data set locking.
- o ISAM server.
- o Easy way to configure ISAM.
- o Utilities for maintaining and modifying data sets:
 - IsamCreate
 - IsamProtect
 - IsamReorg
 - IsamStop

Overview

SECTION 2: CONCEPTS

ISAM accesses fixed-length data records contained in ISAM data sets using fixed-length keys. Each ISAM data set holds one type of data record. At the creation time of the data set, the user specifies the record length and key fields. This section describes concepts important to UNIX PC ISAM.

KEYS

A record key is a field within a record. The content of this field is used to order the records in the data set. When a key is specified in a read operation, the record corresponding to that key value can be accessed directly.

Several parameters describe each key: its position in the record--that is, the offset from the first byte of the record; the key length (size, in bytes); and the key type (such as character, integer, real, and so on).

In addition, the following parameters are specified for each key at the creation time of an ISAM data set:

- o Whether the key values are stored in ascending or descending order.
- o Whether indexing of null value keys is suppressed in order to reduce the size of the index.
- o Whether duplicate key values are allowed in the index.

A record can have up to 20 keys.

Simple and Composite Keys

Keys can be simple or composite. A simple key is based on one field. ISAM sorts an index composed of simple keys in the natural order for the key type. This allows applications to sequentially access records in order by key value. A composite key is based on multiple fields. ISAM sorts composite keys in the same way as simple keys. ISAM accesses identical records in the order they were created. If a composite key is unique, ISAM does not allow duplication.

Concepts

In a composite key,

- o The total key length must not exceed 64 bytes.
- o All fields must be either character strings or byte strings.
- o The sort order must be either ascending or descending for all fields.
- o The fields must be adjacent in the records and appear in their order of significance.

Example 7 in Section 3 illustrates the use of composite keys.

Key Types

UNIX PC ISAM handles 9 different key types (data types). Byte string and character string key types represent character data. Integer, binary, packed decimal, display, and several forms of real numbers represent numeric data. A description of each key type follows. Table 2-1 lists the various key types and the size in bytes of each key type.

Binary

A binary key is an unsigned 1- to 8-byte integer. The low-address byte is the most significant.

Byte String

A byte string key is an uninterpreted fixed-length string of 1 to 64 binary bytes. The low-address byte is the most significant for determining sort order. ISAM distinguishes between uppercase and lowercase ASCII characters. Byte strings have the same representation in all programming languages.

Table 2-1. UNIX PC ISAM Key Types

Type	isam.h name	Description
Binary (0)	TY_BINARY	The size of the key must be 1-8 bytes.
Byte (1)	TY_BYTES	The size of the key must be 1-64 bytes.
Character (2)	TY_CHAR	The size of the key must be 1-64 bytes.
LongReal (4)	TY_LREAL	The size of the key must be 8 bytes.
ShortReal (5)	TY_SREAL	The size of the key must be 4 bytes.
Integer (7)	TY_INTEGER	The size of the key must be 1-8 bytes.
LongIEEE (8)	TY_LIEEE	The size of the key must be 8 bytes (same as long real.)
ShortIEEE (9)	TY_SIEEE	The size of the key must be 4 bytes (same as short real.)

Character String

A character string key is a fixed-length string of 1 to 64 binary bytes. ISAM sorts character strings with the low-address byte as the most significant. In addition, ISAM sorts character string keys with no distinction between uppercase and lowercase ASCII characters. Character strings have the same representation in all programming languages.

Concepts

Integer

An integer key is a signed 1- to 8-byte integer. The low-address byte of an integer key is the most significant for determining sort order.

Long/Short IEEE

Under UNIX, long and short IEEE are equivalent to long and short real, respectively (see below).

Long/Short Real

ISAM uses long real and short real keys in UNIX PC language applications. A long real key is an 8-byte real number, and a short real key is a 4-byte real number. (For information regarding the number of bits of precision and range of values for these keys, refer to the appropriate language manuals.)

Summary

Byte strings, character strings, decimal, and display key types have the same representations in all of the programming languages.

INDICES

For each key field defined for the records of a data set, ISAM builds an index structure within the data set. Using the index, ISAM can quickly access records with different key values. There is one index for record retrieval for each key. ISAM retrieves records in key-order sequence by any key field, starting with any key value. When records are stored or modified, ISAM automatically updates the index. Figure 2-1 shows an index for a data set.

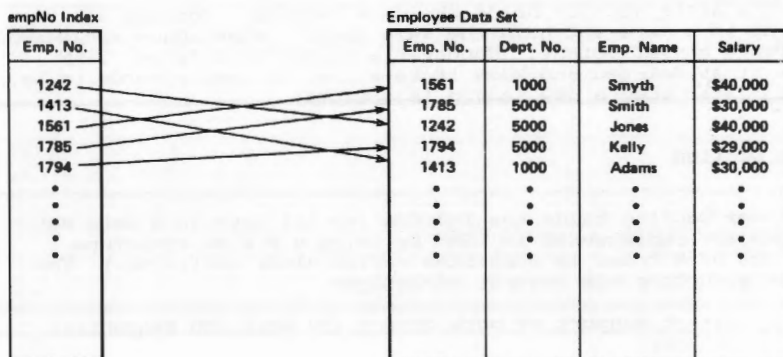


Figure 2-1. Index for a Data Set

A program uses an index to read records in key order or to read a single record directly by a unique key. An example of sequential access is reading the employee records in order by employee number from employee number 100 to employee number 999. An example of direct access is reading the employee record for employee number 15. Both kinds of access read the records using an employee number index. To access records directly, the index must be composed of unique keys. Unique keys mean that only one record exists for each key value, and duplicate keys are not allowed.

Concepts

FILE STRUCTURE

ISAM stores each data set in a single UNIX file. This file contains three sections. The first section is the header, which describes the various attributes of the data set such as record size and keys. The data store section contains the data records for the data set, and the index section contains the indices for all the keys.

Data Store Section

The data store section holds the data records. Because all the records in a data set have the same length, disk space management is simple and efficient. Whenever a record is deleted, ISAM marks it as deleted and adds it to a list of free records to be reused later when a new record is created.

Index Section

The index section holds the indices for all keys in a data set. Indices are implemented in ISAM by using a B-tree structure. (The use of B-trees is sometimes called block splitting.) The B-tree structure has several advantages:

- o Direct support of both direct (by key) and sequential access.
- o Efficiency:
 - Always takes the same number of I/O operations to reach a record.
 - Never has to follow long overflow chains.
 - Typically uses two or three reads to locate a record and one to actually read the record.
- o Very fast sequential access: three or four reads to locate and read the first record and usually a single read to read each subsequent record.
- o Self-reorganization: automatic expansion to accommodate new keys.
- o Automatic compression when keys are removed to improve disk use and efficiency of access.

ISAM DESCRIPTION BLOCK

When ISAM creates a data set, the user supplies an ISAM Description Block which contains a description of the data set, including the record size, key description, and disk allocation policies. Table 2-2 shows the structure of this block. Table 2-3 shows the structure of the Index Specification Block. Table 2-4 shows key types.

Concepts

Table 2-2. ISAM Description Block

Offset	Field	Size (bytes)	Description
0	InitIndexSize	4	Initial size of the index section of the data set.**
4	GrowIndex	4	Size in bytes by which the index section grows when the existing memory is exhausted.**
8	Reserved	2	Currently not used.
10	InitDataSize	4	Initial size of the data store section of the data set.**
14	GrowData	4	Size in bytes by which the data store section grows when the existing file space is exhausted.**
18	RecordSize	2	Size in bytes of the records in the data set. Records must be at least four bytes long.
20	Indices	2	Number of indices for the data set.
22	IndexSpec	Indices*20	Set of Index Specification Blocks that describe a particular index for the data set. There is one ISAM Index Specification Block for each index. Table 2-3 shows the structure of an ISAM Index Specification Block.
**Size defaults to 16384 (16 blocks) if the field is 0. The size may be rounded up based on the data set block size. For more information, see Section 5, <i>System Administration</i> .			

Table 2-3. ISAM Index Specification Block

Offset	Field	Size (bytes)	Description
0	KeyOffset	2	This is the offset of the key component in each record of the data set.
2	KeySize*	2	This is the size of the key component in bytes.
4	Type*	2	This is one of the key type codes (0-11) that represent a key type. (See Table 2-4.)
6	fAscending	2	This is TRUE (not 0) if keys for this index are in ascending order, or FALSE (0) if the keys are in descending order.
8	fNullIndexed	2	This is TRUE (not 0) if null values (binary 0s) are indexed, or FALSE (0) if null values are not indexed.
10	fDuplicates	2	This is TRUE (not 0) if duplicate values are valid for this index. It is FALSE (0) if ISAM is to prevent a record from being stored or modified when the key value duplicates the key of an existing record in the data set. When fDuplicates is FALSE, the index can be used to directly access the records of the data set, because each key is unique.
12	pad	8	Reserved.

*The fields KeySize and Type together specify the type and size of the key component in each ISAM Index Specification Block. See Table 2-4 for key type codes.

Concepts

Table 2-4. ISAM Key Types and Programming Language Representations

LANGUAGE	Index Specification	
Key Type	KeySize	Type
BASIC Interpreter		
Integer (%)	4	7
ShortReal (!)	4	5
LongReal (#)	8	4
FORTRAN-77		
INTEGER*2	2	7
INTEGER*4	4	7
REAL*4	4	5
REAL*8	8	4
DOUBLE PRECISION	8	4
Pascal		
Double	8	8
Integer	2	7
Long integer	4	7
Real	4	9
String	-	1(2)
C		
Double	8	8
Short integer	2	7
Integer/Long integer	4	7
Real	4	9
String	-	1(2)

UNIQUE RECORD IDENTIFIER

ISAM uniquely identifies each record in a data set by a 4-byte quantity called the Unique Record Identifier (URI). The URI identifies the record to be processed.

Unique record identifiers are valid only while the data set is open. They cannot be saved to identify records at a later point when the data set has been closed and reopened.

ISAM HANDLE

An ISAM handle is a 16-bit word that identifies an open data set. The procedure that opens the data set (OpenISAM) returns the ISAM handle. Subsequent ISAM procedures use the ISAM handle for the parameter ISAMHandle. An ISAM handle is valid until the data set is closed or deleted.

ISAM PROCEDURES

Most ISAM procedures perform an operation that falls into one of four categories: storing, reading, modifying, or deleting.

Storing

When an application program stores a new record, ISAM inserts the record into the data set and indexes the record according to the values in all the record's key fields. Store procedures return the URIs of the records that ISAM stores.

Reading

When an application program reads an existing record, ISAM can retrieve any of the following:

- o A single record with a given unique key or unique record identifier.
- o All records with keys of a specific value (that is, an exact match).
- o All records with key values residing in a specified range (that is, a range match).

Concepts

- o All records in which the beginning of a byte or character string key matches a particular value (that is, a prefix match).

ISAM can retrieve either specified records or a sequence of 4-byte URIs. Both types of information are returned by read procedures. If a program obtains URIs, it can later obtain the corresponding records by using a special form of the read operation that does not reaccess the index.

Modifying

When a program modifies an existing record, ISAM removes it from each index for which the key field is being changed and indexes it under the new key. Modify procedures can use either the URI or one of the keys to identify the record to be processed.

Deleting

When an application program deletes an existing record, ISAM removes the record from the data set and from each index. Delete procedures use either the URI or one of the keys to identify the record to be processed.

DATA SET SHARING

Since applications can share data sets, ISAM must carefully control data set access.

Access Modes

ISAM has three modes for accessing a data set: administrator mode, batch mode, and transaction mode. Administrator mode performs data set activities such as deleting, renaming, and setting protection. Only the ISAM administrator can open a data set in administrator mode (see Section 5, System Administration).

Note

Do not confuse ISAM administrator mode with UNIX administrator mode.

Batch mode opens a data set for applications that require exclusive access. Opening a data set in transaction mode allows applications to access and modify a data set concurrently. Table 2-5 summarizes the restraints controlling data set access for each mode.

Table 2-5. Restrictions on Multiple Access to a Data Set

Existing Access	Valid Modes for New Open
Administrator	None
Batch read	Batch read Transaction read
Batch modify	None
Transaction read	Batch read Transaction read Transaction modify
Transaction modify	Transaction read Transaction modify

If ISAM opens a data set in batch or transaction mode, the data set can be either read or modified. The extent to which other applications share the data set depends on the read or modify performed.

If ISAM opens a data set in batch read mode, other users can open it for read-only access. ISAM denies a request to open the data set in administrator, batch modify, or transaction modify mode if the data set is open in batch read mode. A request to open the data set in batch read mode does not execute if the data set is already open in administrator, batch modify, or transaction modify mode.

A data set open in batch modify mode is open exclusively. No other user can open it. A request to open the data set in batch modify mode does not execute if another user already has it open in any mode.

Other users can open a data set opened in transaction read mode in batch read, transaction read, or transaction modify modes. A request to open the data set in transaction read mode does not execute if another user has the data set open in administrator or batch modify mode.

Concepts

A data set open in transaction modify mode can be opened by other users in either transaction read or transaction modify mode. A request to open the data set in transaction modify mode cannot execute if another user has the data set open in administrator, batch read, or batch modify mode.

A data set open in administrator mode is open exclusively. A request to open the data set in administrator mode does not execute if another user has already opened it in any mode.

A program can open a data set in batch mode to read or modify records. In batch modify mode, an application has exclusive access to the data set. Since other applications do not have access to the data set at the same time, it is not possible to make multiple updates simultaneously. Modification of the records and data sets does not affect any other application.

Transactions

In ISAM, transactions are the coordination mechanism that prevent errors that can result from simultaneous access to a data set. In transaction mode (that is, when a data set is open for shared access), many applications can access the same records and data sets. Any modification of a record or data set by one application affects the other applications. Errors can arise if simultaneous access to a data set is uncoordinated.

In transaction mode, applications designed to permit multiuser access to a data set divide their processing into a series of transactions. Each transaction is a unit of work, as shown in Figure 2-2. Changes to a data set must be made within a transaction.

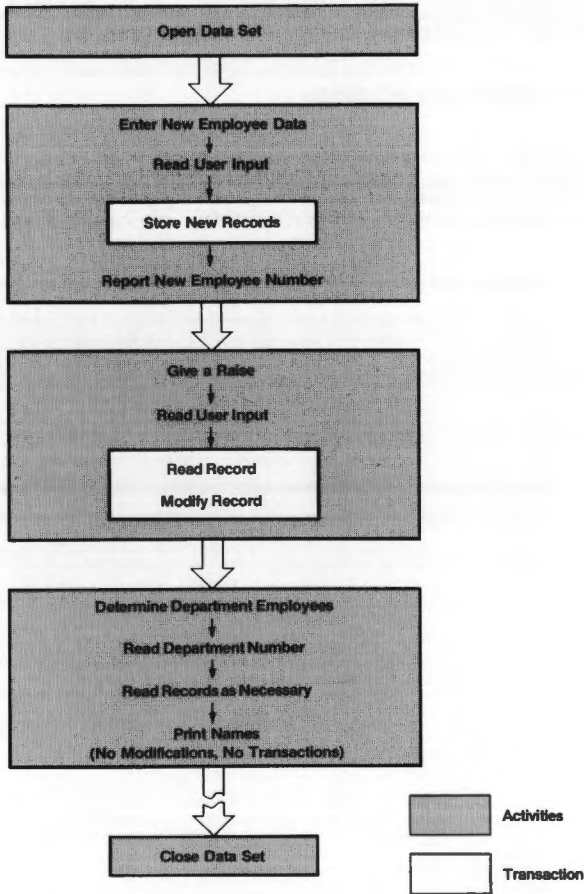


Figure 2-2. Series of Transactions

Concepts

The beginning of a transaction is the BeginTransaction procedure. The end of a transaction is a CommitTransaction procedure.

While some procedures can be performed whether or not an application is in a transaction, any procedure that locks or modifies a record (or locks a data set) must take place while the application is in a transaction, as summarized in Table 2-5.

Transaction-Related Constraints

Table 2-6 shows the constraints associated with each ISAM procedure used during transaction processing. These constraints do not apply to procedures on data sets open for batch access, that is, in administrator, batch modify, or batch read modes.

Table 2-6. Transaction-Related Constraints

Must Be Used in a Transaction	May Be Used Any Time
CommitTransaction	CloseISAM
DeleteByKeyISAMRecord	CreateISAM*
DelISAMRecord	DeleteISAM**
GetHoldISAMRecords	BeginTransaction***
HoldISAMDataSet	GetISAMRecords
ISAMHoldRecord	OpenISAM
ModifyISAMRecord	QueryTransactionParams
ModifyByKeyISAMRecord	RdNextISAMRecord
RdHoldNextISAMRecord	RdUniqueISAMRecord
ReadHoldISAMRecordByUri	ReadISAMRecordByUri
RdHoldUniqueISAMRecord	RenameISAM**
ReleaseDataSet	SetISAMProtection**
ReleaseISAMRecord	SetTransactionParams
StoreISAMRecord	SetUpLimitsISAMIteration
	SetUpPrefixIteration
	VerifyMultiUserISAM
*Not applicable because a data set is not open when it is created. **Not applicable because a data set must be open in administrator mode to use this operation. ***Not applicable because BeginTransaction cannot be called within a transaction.	

Concepts

Locking a Record or Data Set

During a transaction, an application can make several changes to a data set. When an application is in the midst of modifying data, the data set is not in a consistent state. Other applications must be prevented from accessing the changed data when it is in an inconsistent state. By using transaction modify mode and locking records and data sets, an application prevents other applications from accessing inconsistent data. To prevent concurrent modification by multiple applications, ISAM allows an application to **lock** a record or data set, thereby giving that application exclusive access to the record or data set.

To illustrate the importance of transactions and locking, consider a bank account system that processes several financial transactions simultaneously. Jill and John Kelly have a joint savings account with a balance of \$1000. John Kelly withdraws \$500 and Jill Kelly deposits \$500.

John Kelly's withdrawal is accomplished by

1. Looking up the balance in the ledger
2. Subtracting \$500 from the balance
3. Writing the new balance to the account

Jill Kelly's deposit is accomplished by

1. Looking up the balance in the ledger
2. Adding \$500 to the balance
3. Writing the new balance to the account

In manual banking systems, this method works well because the teller holds the passbook during an entire transaction. Only one transaction can occur at a time. Jill Kelly can make a deposit and then John Kelly can make a withdrawal; or John Kelly can make a withdrawal and then Jill Kelly can make a deposit.

With an electronic banking system in which parallel transactions can occur, the following could happen:

Concepts

<u>John Kelly's</u> <u>Transaction</u>	<u>Jill Kelly's</u> <u>Transaction</u>
1. Look up the balance: \$1000.	
2.	Look up the balance: still \$1000.
3. Subtract \$500 from the \$1000 balance: new balance, \$500.	
4.	Add \$500 to the \$1000 balance: new balance, \$1500.
5. Write the new balance, \$500, to the account.	
6.	Write the new balance, \$1500, to the account, overwriting the \$500 balance written by John Kelly's transaction.

These procedures would be disastrous from the bank's point of view, because John Kelly was given \$500, but the debit to his account was "forgotten." Some way must be found to avoid such a situation.

In an electronic banking system the two transactions could be correctly processed as follows:

John Kelly's withdrawal is accomplished by

1. Starting the transaction.
2. Reading the account balance (\$1000) and locking the account record.
3. Subtracting \$500 from the balance: new balance, \$500.
4. Writing the new balance to the account.
5. Ending the transaction, thereby releasing the account record.

Concepts

Jill Kelly's deposit is accomplished by

1. Starting the transaction.
2. Reading the account balance (\$500) and locking the account record.
3. Adding \$500 to the balance: new balance, \$1000.
4. Writing the new balance to the account.
5. Ending the transaction, thereby releasing the account record.

With this method, the error outlined previously cannot occur, because the account record is locked after it is read. John Kelly's transaction cannot read the account record from the time Jill Kelly's transaction first reads the record until her transaction ends. Similarly, Jill Kelly's transaction cannot read the account record from the time John Kelly's transaction first reads the record until his transaction ends.

If an application locks a record, only that application has access to the record until it is released or unlocked. If an application locks a data set, only that application has access to the data set until it is unlocked.

Locking a record allows other applications to access the remaining records of the data set. Locking a data set should be done only when necessary because other applications are prevented from accessing any of the data set's records.

Conflicts and Deadlocks

An application using ISAM should lock all records it needs before modifying the data sets. However, conflicts and deadlocks occur when it is not possible for an application to lock all necessary records.

Conflict arises when more than one application attempts to lock the same record or data set. The first application to request the record or data set obtains it, and requests from other applications are placed in a queue while the record or data set is locked.

There is a special case of conflict called a **deadlock**. A deadlock occurs when one application has locked a record or data set and attempts to lock another record or data set that is already locked by a second application waiting to lock the record or data set locked by the first application. Each application is queued for the record or data set locked by the other and thus both can wait forever. A deadlock can become quite complicated when a number of programs and requests are involved, as shown in Figure 2-3.

Concepts

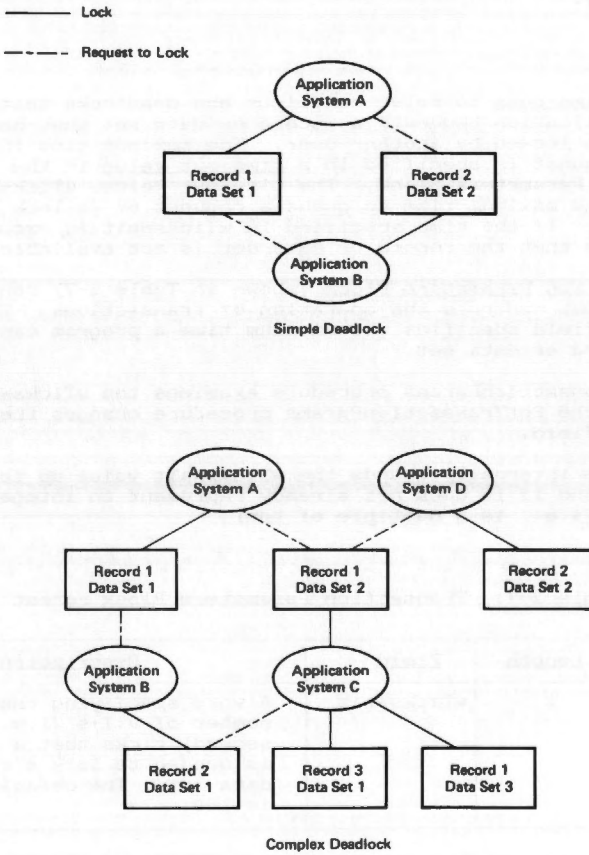


Figure 2-3. Examples of Deadlock

Concepts

A similar problem occurs when an application with a locked record or data set is waiting for an external event (for example, waiting for an offline printer to come online or for a user to enter data) while a second application is waiting for the locked record or data set. The second application may have to wait indefinitely.

Time-Outs

ISAM uses time-outs to solve conflicts and deadlocks that occur when an application requests a record or data set that has already been locked by another user. The maximum time ISAM queues a request is specified in a time-out value in the Transaction Parameters Block. The time-out value, `wTicksWait`, specifies the maximum time to queue a request or to lock a record or data set. If the time specified in `wTicksWait` is exceeded, ISAM reports that the record or data set is not available.

The Transaction Parameters Block (shown in Table 2-7) contains the value that controls the operation of transactions. The `wTicksWait` field specifies the maximum time a program can wait to lock a record or data set.

The `QueryTransactionParams` procedure examines the `wTicksWait` field, and the `SetTransactionParams` procedure changes the `wTicksWait` field.

UNIX PC ISAM internally rounds the `wTicksWait` value up to the nearest second if it does not already represent an integer number of seconds (i.e., is a multiple of ten).

Table 2-7. Transaction Parameters Block Format

Offset	Length	Field	Description
0	2	<code>wTicksWait</code>	A word specifying the maximum number of 0.1-s (i.e., 0.1 second) ticks that a request is queued to lock a record or data set. The default value is 100.

ISAM UTILITIES

ISAM provides utilities for maintaining and modifying ISAM data sets. The utilities are:

- o IsamCreate
- o IsamProtect
- o IsamReorg
- o IsamStop

For a detailed description of these utilities, see Section 5, **System Administration**.

DATA SECURITY

Data security at each ISAM data set level is available through two associated passwords: a read password and a modify password. ISAM protects all data sets it creates so that only the ISAM administrator can access them from UNIX.

DATA INTEGRITY

ISAM includes features for maintaining the integrity of data on disk files. ISAM maintains and monitors the integrity of data through:

- o Error logging. Errors discovered in the file structures maintained by ISAM are logged in the ErrorLog file in the ISAM administrator's home directory.. The ISAM administrator should examine this file frequently to determine if there are problems with any of the data sets.
- o Internal consistency checking. Hardware or software errors can occasionally introduce anomalies into a data set. ISAM algorithms minimize these anomalies by detecting them where possible and preventing them from becoming worse.

If an ISAM data set is damaged by hardware or software failure, use the IsamReorg utility to reconstruct the ISAM data set.

Concepts

SECTION 3: ISAM APPLICATIONS

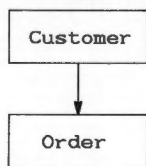
This section contains seven examples written in C that access ISAM. The examples illustrate:

- o Creating a data set
- o Opening an existing data set
- o Storing a new record
- o Reading single and multiple records
- o Using compound indices

The examples refer to procedures, which are described in detail in Section 4, **ISAM Procedures**.

EXAMPLE 1: CREATING SIMPLE DATA SETS

Consider an example in which an application maintains a set of customers and a set of current orders for each customer. The normal access pattern is to process all current orders for a given customer. The following data structure diagram shows the relationship between customers and orders:



One record type is used for information relevant to customers, and another is used for information relevant to orders:

```
typedef struct {  
    char custName[32];  
    char custNo[5];  
    /* additional fields for the application */  
} custType;
```

Applications

```
typedef struct {  
    char orderNo[8];  
    char custNo[5];  
    char orderDate[10];  
} orderType;
```

In the customer record, both custName and custNo are keys. Typically, an application accesses a customer record by the custName index. After accessing the customer, the application can obtain the custNo. The field custNo is also a key in the order record. To obtain all the order records corresponding to a given customer, the application retrieves order records according to the custNo key.

Either the IsamCreate utility or the application system using the CreateISAM procedure can create the data set of records.

If CreateISAM is used, the application must supply the following information to CreateISAM for each record type:

- o UNIX file name for the data set
- o Password required to create data set
- o An ISAM Description Block of struct ISAMDesc provided in /user/include/isam.h
- o An ISAM Status Block of struct ISAMStatusBlock, also in isam.h
- o An Index Specification Block for each index, of struct IndexSpec, also in isam.h

Applications

Following is a typical isam.h entry for these structures. (Your software version may differ in some details.)

```
/*
 *IndexSpec is the structure which describes a particular
 *index for a data set.
 */
struct IndexSpec {
    ushort   KeyOffset;    /* offset of key field into data
                           record */
    ushort   KeySize;      /* size of key field */
    ushort   Type;         /* data type of key */
    ushort   fAscending;   /* TRUE if keys are in ascending
                           order, FALSE if keys are
                           descending. */
    ushort   fNullIndexed; /* TRUE if null keys are
                           included. */
    ushort   fDuplicates;  /* TRUE if duplicates keys are
                           allowed. */
    long     filler[2];
};

struct ISAMDesc {
    long     InitIndexSize; /* Initial index file size */
    long     GrowIndex;     /* Amount to grow index file when
                           it fills up */
    ushort   SectorNode;   /* number of 512 byte sectors per
                           node. (Not Used) */
    long     InitDataSize; /* Initial size of data store
                           file */
    long     GrowData;     /* Amount to grow data file when
                           it fills up */
    ushort   RecordSize;   /* number of bytes in record */
    ushort   Indices;      /* number of indices for this
                           file */
    struct   IndexSpec IndSpec[ISAMIndCnt];
};

struct ISAMStatusBlock {
    ushort   ercISAM;
    ushort   ercDetail;
};
```

Applications

The application calls CreateISAM with the following statement:

```
erc = CreateISAM(
    FileName, strlen(FileName),
    0,0,
    0,0,
    0,0,
    &DescBlock, sizeof (DescBlock),
    &StatusBlock);
```

where

erc	is a word into which the status code returned by CreateISAM is stored.
FileName	is a character array, containing the data set file name.
&DescBlock	points to a struct of type ISAMDesc (as shown above) that gives the remaining information used by CreateISAM, such as record size. It must include one IndexSpec for each index field.
&StatusBlock	points to a 4-byte struct of type ISAMStatusBlock (as shown above) into which the status codes from the procedure are placed.

(This example assumes no password is required. For more information, see the full description of CreateISAM in Section 4.)

All ISAM procedures take the address of an ISAM Status Block as their last argument. The two status code fields return the procedure status. The ercISAM field is either 0 (OK) or is one of the ISAM status codes (see Appendix A). When the ercISAM field is not 0, the ercDetail field may contain additional status information (refer to intro(2) in the AT&T UNIX™ PC UNIX System V Manual).

In the following example, the data set for the "custType" records is created.

Note

If there is more than one index, the #define ISAMIndCnt statement must specify the number. If this statement is not included, the file /usr/include/isam.h will set a default value of 1.

```

#define ISAMIndCnt 2
#include <isam.h>
#include <stdio.h>
main()
{
    struct ISAMStatusBlock StatusBlock;
    struct ISAMDesc IsamDesc;
    unsigned short erc;

    /* Uses default values for first five elements of
    IsamDesc. */

    IsamDesc.RecordSize = 40;
    IsamDesc.Indices = 2;

    /* Specifications for index "custName[32]" */

    IsamDesc.IndSpec[0].KeyOffset=0;
    IsamDesc.IndSpec[0].KeySize=32;
    IsamDesc.IndSpec[0].Type=2; /*character*/
    IsamDesc.IndSpec[0].fAscending=1;
    IsamDesc.IndSpec[0].fNullIndexed=1;
    IsamDesc.IndSpec[0].fDuplicatesAllowed=1;

    /* Specifications for index "custNo[5]" */

    IsamDesc.IndSpec[1].KeyOffset=32;
    IsamDesc.IndSpec[1].KeySize=5;
    IsamDesc.IndSpec[1].Type=2; /*character*/
    IsamDesc.IndSpec[1].fAscending=1;
    IsamDesc.IndSpec[1].fNullIndexed=1;
    IsamDesc.IndSpec[1].fDuplicatesAllowed=1;

    /* Calling CreateISAM to create "customer.isam" data
    set. */

    erc= CreateISAM("customer.isam",13,
    0,0,0,0,0,0,
    &IsamDesc, sizeof(IsamDesc),
    &StatusBlock);

    if (erc != 0) {
        printf("errors are %d, %d
        StatusBlock.ercDetail);
    } else printf ("No errors in CreateISAM
    exit(0);

}

```

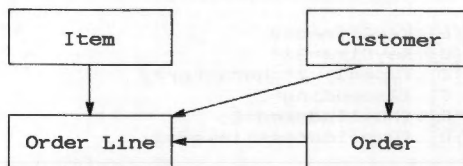
Applications

EXAMPLE 2: CREATING DATA SETS (MULTIPLE INDICES)

This example extends the application of Example 1 above. In addition to customers, there are items and orders. Customers, items, and orders are each associated with a set of order lines. Correspondingly, there are four different record types:

- o Customers
- o Items
- o Orders
- o Order lines

The following data structure diagram shows the relationships:



The structures for the items, customers, and orders, respectively, contain the fields:

```
char itemNo[6];  
char custNo[5];  
char orderNo[8];
```

Order line records are stored in a data set, with ISAM maintaining the indices that link item, customer, and order records to order line records. Each order line record is indexed under the item, customer, and order number so that the application can access the order lines associated with any item, customer, or order:

```
struct orderLine {  
    char itemNo[6];  
    char dueDate[10];  
    char custNo[5];  
    char orderNo[8];  
    int short quantity;  
}
```

Either the IsamCreate utility or the application using the CreateISAM procedure can create the data set of order records.

Applications

Calling CreateISAM to create the data set of order line records is essentially the same as shown in Example 1 above except that there are three indices. The IndexSpec field of ISAMDesc must have three elements ([0..2]). Table 3-1 shows the values for the fields of these elements.

Table 3-1. Fields of the Elements of IndexSpec

	Index 0 Item No.	Index 1 Cust No.	Index 2 Order No.
KeyOffset	0	16	21
KeySize	6	5	8
Type	2	2	2
fAscending	1	1	1
fNullIndexed	1	1	1
fDuplicates	1	1	1

That is, the item number index is a string of 6 bytes starting at byte 0 of the record; the customer number index is a string of 5 bytes starting at byte 16; and the order number index is a string of 8 bytes starting at byte 21.

EXAMPLE 3: OPENING A DATA SET

To open a data set, supply the data set name and password, the record size for the data set, and an open mode. (See the subsection called "Data Set Sharing" in Section 2.) ISAM returns an ISAM handle that identifies the data set in subsequent procedures. The ISAM handle remains valid until the data set is closed.

The OpenISAM procedure opens the data set at the operating system level. OpenISAM also verifies the password and checks data set integrity.

To open the data set of Example 2 above for transaction modification, the application system calls OpenISAM with the following statement:

```
erc = OpenISAM(  
    &ISAMHandle,  
    &Filespec, strlen(Filespec),  
    &Password, strlen>Password),  
    mode, sRecord,  
    &StatusBlock);
```

Applications

where

erc	is a word into which the status code returned by OpenISAM is stored.
&ISAMHandle	points to a word set by OpenISAM. It is used in subsequent operations to identify the open data set.
&Filespec	points to an array containing the name of the data set.
&Password	points to an array containing the modify password of the data set.
mode	is a word containing the MD_TMOD (from isam.h).
sRecord	is the record size (in bytes) for this data set.
&StatusBlock	points to a 4-byte record of struct ISAMStatusBlock, as described in Example 1 above, into which the status codes from the procedure are placed.

This example uses transaction modify mode because it is most typical for multiuser ISAM applications. This mode ensures that any interactions with the data set are not nullified by other users' interactions.

Procedures performed on the data set must occur within a transaction. This means that the BeginTransaction and CommitTransaction calls must bracket all operations.

Note

The decimal and hexadecimal values for the various modes defined in isam.h are:

mode	name	dec	hex
batch read	MD_BREAD	25202	6272x
batch modify	MD_BMOD	25197	626dx
transaction read	MD_TREAD	29810	7472x
transaction modify	MD_TMOD	29805	746dx
administrator mode	MD_ADMIN	24932	6164x

EXAMPLE 4: STORING A NEW RECORD

To store a new record in the data set of Example 2 above, the application calls the StoreISAMRecord procedure, supplying a record it has constructed.

```
erc = StoreISAMRecord (
    ISAMHandle,
    &IorderLine, sizeof (IorderLine),
    &uriOrderLine,
    &StatusBlock);
```

where

erc	is a word into which the status code returned by StoreISAMRecord is stored.
ISAMHandle	identifies the open data set, as described in Example 3 above.
&IorderLine	points to a variable of struct orderLine (see Example 2 above), that has been filled with the desired data.
&uriOrderLine	points to a 4-byte unsigned integer into which StoreISAMRecord stores the unique record identifier of the new record. This value may be used in subsequent procedures to identify the record.
&StatusBlock	points to a 4-byte record of type ISAMStatusBlock, as described in Example 1 above, into which the status codes from the operation are placed.

After successfully calling StoreISAMRecord, the record is in the data set, and all indices of the data set are updated to reflect the record's presence.

Applications

EXAMPLE 5: READING A SINGLE RECORD

A record can be read using a single procedure if either of the following is known:

- o Unique record identifier.
- o Value of a key field for which duplicates are not allowed.

If the unique record identifier (returned by StoreISAMRecord or any of the read procedures) of the record is known, then ReadISAMRecordByUri reads the record. This procedure reads the record without using the index structures of the data set, so the record can typically be retrieved in a single disk seek.

```
erc = ReadISAMRecordByUri(  
    ISAMHandle,  
    uriOrderLine,  
    &RorderLine, sizeof (RorderLine),  
    &StatusBlock);
```

where

erc	is a word into which the status code returned by OpenISAM is stored.
ISAMHandle	identifies the open data set, as described in Example 3 above.
uriOrderLine	contains the unique record identifier of the desired record.
&RorderLine	points to a variable of type orderLine (from Example 2) that will receive the data.
&StatusBlock	points to a 4-byte record of type ISAMStatusBlock, as described in Example 1 above, into which the status codes from the operation are placed.

If the value of a key field for which duplicates are not allowed is known, then ReadUniqueISAMRecord reads the record.

Applications

For an example of the use of the ReadUniqueISAMRecord procedure, add the field orderLineNo to orderLineType, shown in the following declaration:

```
struct orderLine {  
    char itemNo[6];  
    char dueDate[10];  
    char custNo[5];  
    char orderNo[8];  
    int short quantity;  
    char orderLineNo[10];  
    /* New Field */  
    /* Additional fields used by application */  
};
```

and specify a new index (OrderLineNo), for which duplicates are not allowed, to CreateISAM (index 3). Table 3-2 shows the values for this field.

Table 3-2. Fields of Order Line Number

Field	Index 3 Order Line Number
KeyOffset	31
KeySize	10
Type	1
fAscending	TRUE (not 0)
fNullIndexed	TRUE (not 0)
fDuplicates	FALSE (0)

If a record in the data set has the value "KL556-8890" in the field orderLineNo, the following statement reads that record into the variable orderLineType (of type OrderLineType immediately above):

```
erc = RdUniqueISAMRecord(  
    ISAMHandle,  
    IndexOrderLineNo,  
    &OrderLineNo,  
    strlen(OrderLineNo),  
    &RorderLine, sizeof (RorderLine),  
    &uriOrderLine,  
    &StatusBlock);
```

Applications

where

erc	is a word into which the status code returned by ReadUniqueISAMRecord is stored.
ISAMHandle	identifies the open data set, as described in Example 3 above.
IndexOrderLineNo	is 3 to identify the OrderLineNo index.
&OrderLineNo	points to an array of 10 bytes containing the ASCII string "KL556-8890".
&RorderLine	points to a variable of struct orderLine (from Example 2) that will receive the data.
&uriOrderLine	points to a 4-byte structure into which the unique record identifier of the record returns.
&StatusBlock	points to a 4-byte record of type ISAMStatusBlock, as described in Example 1 above, into which the status codes from the operation are placed.

EXAMPLE 6: READING MULTIPLE RECORDS

An application frequently needs to process all the records for which a key value either is known, lies in some specified range, or has a particular prefix. The multiple record procedures of ISAM provide this capability.

To access multiple records in a data set, specify the index and desired key, key prefix, or range of keys using one of the following two procedures:

- o SetLimitsISAMIteration
- o SetupPrefixIteration

Then read the records either one at a time, with ReadNextISAMRecord, or in groups, with GetISAMRecords. GetISAMRecords can read either records (and their unique record identifiers) or unique record identifiers without the associated records. You can use any of the single record procedures to process the records read between calls of ReadNextISAMRecord and GetISAMRecords. (ISAM does not support nested iterations on the same ISAM handle. For nested iteration, the data set must be opened twice using different ISAM handles.)

Applications

This example uses the data set of Example 2 above. The user of the application is an order-filling clerk who obtains the items ordered and places them with the order. For an item in stock, the application iterates through the order lines, displaying the quantity of the item ordered. The application deletes the order line while the clerk fills the order.

Of course, in a real application, the order lines would not be deleted. Rather, they would have a field to indicate whether or not they had been filled. They are deleted here to simplify the example.

The following skeleton program could serve as the framework for this part of the application. (An application must handle unexpected errors, and the program completely ignores normal status codes.)

```
#define ErcOK 0
#define IndexItemNo 0

short ISAMHandle; /* ISAM handle for open ISAM data set */
OrderLine RorderLine; /* see example 2 */
int uriOrderLine;
char ItemNo[6]; /* key field for the item no. */
struct ISAMStatusBlock StatusBlock;
short erc;

/* ItemNo contains the item number for which orders are to be
   filled. Iterate through the order lines for this item. Stop
   when the status code "No more records" is returned. Then lock
   the data set, set up the ISAM iteration, and get the first
   unfilled record. Transactions are used to avoid contention
   for the data from other programs. */

erc = SetUpLimitsISAMIteration( ISAMHandle, IndexItemNo,
                               3,&ItemNo, sizeof( ItemNo ), 0,0
                               &StatusBlock );
erc = BeginTransaction (&StatusBlock);
erc = RdHoldNextISAMRecord( ISAMHandle, &RorderLine, sizeof(
                             RorderLine ), &uriOrderLine,
                             &StatusBlock );

while( erc == ErcOk ) {

    /* the data set is locked and orderLine contains the next
       order line to be filled. Display the fields of interest
       to the clerk. Delete the order line while it is being
       filled */

    erc = DelISAMRecord( ISAMHandle, uriOrderLine,
                        &StatusBlock );
```

Applications

```
/* End the transaction so that other users can access the
   data set while the clerk is filling the order */

erc = CommitTransaction( &StatusBlock );

/* Wait for the clerk to fill the order */

erc = BeginTransaction ( &StatusBlock );

/* Get the next record and loop */

erc = RdHoldNextISAMRecord( ISAMHandle, &RorderLine,
    sizeof(RorderLine ),
    &uriOrderLine,
    &StatusBlock );
}; /* end while */

/* Unlock the data set to allow access by other workstations */

erc = CommitTransaction( &ISAMStatusBlock );

}
```

EXAMPLE 7: USING COMPOUND INDICES FOR EFFICIENT RETRIEVAL

Extend Example 6 above to process only records due prior to a specified date, using the dueDate field of the order lines records. Process the records in order by date due.

An inefficient solution would have the application check the dueDate field of each order line read. The application would process only those order lines for which the dueDate was prior to the specified date. This solution would not process the records in order by date due.

A compound index can be constructed for the order line data set that allows the application to process only the desired records, and to process them in the correct order. Recall that the records of the order line data set are of type orderLine:

```
struct orderLine {
    char itemNo[6];
    char dueDate[10];
    char custNo[5];
    char orderNo[8];
    int short quantity;
    char orderLineNo[10];
    /* Additional fields */
};
```

Specify a new index (ItemNoDueDat, the composition of item number and due date) to CreateISAM (index 4). Table 3-3 shows the values for this field.

Table 3-3. Fields of Item Number Due Date

<u>Field</u>	<u>Index 4</u> <u>Item Number Due Date</u>
KeyOffset	0
KeySize	16
Type	1
fAscending	TRUE (1)
fNullIndexed	TRUE (1)
fDuplicates	TRUE (1)

A possible fragment of the B-Tree for this new index is

... Record18 Record19 Record20

where

<u>Record</u>	<u>Item No.</u>	<u>Due Date</u>	<u>Composite Key</u>
18	H22288	1981-10-23	H222881981-10-23
19	H22288	1981-10-24	H222881981-10-24
20	H22294	1981-07-04	H222941981-07-04

The records are ordered first by item number, then by due date.

The only changes to the program skeleton of Example 6 would involve setting up the iteration

```
erc = SetupLimitsISAMIteration(
    ISAMHandle,
    IndexItemNoDueDate, 7,
    &itemNoDueDateFirst,
    sizeof(itemNoDueDateFirst),
    &itemNoDueDateLim,
    sizeof(itemNoDueDateLim),
    &StatusBlock);
```

Applications

where

IndexItemNoDueDate is 4 to identify the composite key.

itemNoDueDateFirst are both of type
itemNoDueDateLim ItemNoDueDateType:

```
struct{  
  char itemNo[6];  
  char dueDate[10];  
}
```

The itemNo fields of both itemNoDueDateLim and itemNoDueDateFirst are set to the desired item number, so that the iteration is restricted to that item. The dueDate field of itemNoDueDateFirst is set to binary 0's so that the iteration starts with the first order line for the item. The dueDate field of itemNoDueDateLim is set to the desired due date, so the iteration stops when that due date is encountered.

SECTION 4: ISAM PROCEDURES

The ISAM procedures create, access, manipulate, and manage entire data sets or specified records. To use these C procedures from Interpreted BASIC, consult the file /usr/lib/basic/Basgen.config.

Table 4-1 shows the procedures by functional category. In this section, the ISAM procedures are described in detail in alphabetical order.

Table 4-1. ISAM Procedures by Function

<u>Data Set Management</u>	<u>Multiple Record Access Iteration</u>
CreateISAM	GetISAMRecords
DeleteISAM	GetHoldISAMRecords
RenameISAM	RdNextISAMRecord
SetISAMProtection	RdHoldNextISAMRecord
	SetUpLimitsISAMIteration
<u>Data Set Access</u>	SetUpPrefixIteration
CloseISAM	<u>Record Locking</u>
OpenISAM	HoldISAMDataSet
	ISAMHoldRecord
<u>Record Management</u>	ReleaseDataSet
DelISAMRecord	ReleaseISAMRecord
DeleteByKeyISAMRecord	
ModifyISAMRecord	<u>Transactions</u>
ModifyByKeyISAMRecord	BeginTransaction
StoreISAMRecord	CommitTransaction
	QueryTransactionParams
<u>Single Record Access</u>	SetTransactionParams
ReadISAMRecordByUri	
ReadHoldISAMRecordByUri	<u>ISAM Service Access</u>
RdUniqueISAMRecord	VerifyMultiuserISAM
RdHoldUniqueISAMRecord	

Procedures

HEADER FILES

Two C format header files are included with ISAM. These are `isam.h`, which contains the structure and mode definitions used with these procedures, and `iserc.h`, which contains the various error codes that these procedures can return. (See Appendix A for a complete list of errors.) Both header files are in `/usr/include` on systems with ISAM installed.

THE LIBRARY

The ISAM library in `/usr/lib` (`libisam.a`) contains the procedures described in this section. To compile using this library, add the construction `-lisam` at the end of the command line. For example,

```
cc -o isam_app isam_app.c -lisam
```

FILE NAMES

File names used in calls to `OpenISAM` and `CreateISAM` are interpreted relative to the ISAM administrator's home directory. In other words, if a user creates a file called `file.isam`, it will be created in the ISAM administrator's directory. To access data sets in other directories, the user must supply the full pathname. In addition, ISAM must be able to access the directory. To create a file in a directory, the ISAM administrator must be able to write into that directory. After the file is created, the ISAM administrator needs only execute permission in the directory. All data sets created by ISAM are owned by the ISAM administrator and can be read and written only using calls to the server and the special utilities described in Section 5, *System Administration*.

NOTE ON PARAMETER NAMES

Many of the parameter names in this manual include prefixes that, by convention, denote parameter information. The list of prefixes is as follows:

`c` = count. For example, `cIndexes` is the number of indices for the data set.

`f` = flag (`TRUE`=not 0 or `FALSE`=0). For example, if `fAscending` is set to 1, the field is sorted in ascending order.

Procedures

p = logical memory address (pointer). For example, **pKeyDescriptor** specifies the address of a **KeyDescriptor**.

s = size in bytes (unsigned number). For example, **sRecord** specifies the number of bytes in a record.

STATUS BLOCK

All ISAM procedures include the **pStatusBlockRet** parameter, which points to the memory address of a status block containing errors. The 4-byte status block, shown in Table 4-2, contains two status codes: **ercISAM** and **ercDetail**.

Table 4-2. Status Block Format (**pStatusBlockRet** Parameter)

Offset	Size (bytes)	Field	Description
0	2	ercISAM	Status code. (See Appendix A.)
2	2	ercDetail	Detail UNIX status code.

ercISAM is either 0 (OK) or one of the ISAM status codes listed in Appendix A. If **ercISAM** is nonzero, **ercDetail** may give additional information about the error; refer to intro(2) in the AT&T UNIX PC UNIX System V Manual. For example, if the user tries to open a nonexistent file, **ercISAM** contains **CANT_OPEN** (Can't open data set) and **ercDetail** contains 2 (No such file or directory). The structure **StatusBlock** is included in the **isam.h** file. **ercISAM** is the same as the value returned by ISAM procedures.

Note

If an ISAM procedure returns an **ercISAM** other than 0, any other values set by the procedure are meaningless.

Procedures

NAME

BeginTransaction -- begin a transaction

SYNOPSIS

```
int BeginTransaction( pStatusBlockRet )  
struct tStatusBlock *pStatusBlockRet;
```

DESCRIPTION

The **BeginTransaction** procedure marks the start of a transaction for the application. The transaction ends when the application uses a **CommitTransaction** procedure.

BeginTransaction cannot be called from within a transaction.

An interpretation of the arguments is provided below.

pStatusBlockRet

is the memory address of the status block to which the status codes from the procedure return. (For detailed information, see the "Status Block" subsection at the beginning of this section.)

SEE ALSO

CommitTransaction

NAME CloseISAM -- close an open data set

SYNOPSIS

```
int CloseISAM( ISAMHandle, pStatusBlockRet )
unsigned short ISAMHandle;
struct tStatusBlock *pStatusBlockRet;
```

DESCRIPTION

The CloseISAM procedure closes and releases all resources associated with an open data set. An ISAM handle for the closed data set is not valid after the CloseISAM procedure is called.

An interpretation of the arguments is provided below.

ISAMHandle is the ISAM handle that identifies the open data set.

pStatusBlockRet is the memory address of the status block to which the status codes from the procedure return. (For detailed information, see the "Status Block" subsection at the beginning of this section.)

SEE ALSO
OpenISAM

Procedures

NAME

CommitTransaction -- complete an active ISAM transaction

SYNOPSIS

```
int CommitTransaction( pStatusBlockRet )
struct tStatusBlock *pStatusBlockRet;
```

DESCRIPTION

The **CommitTransaction** procedure signifies the successful completion of a transaction and unlocks all records and data sets locked by the application. When the **CommitTransaction** call returns, the ISAM server guarantees that all the changes caused by the transaction have been made a permanent part of the data sets used in the transaction.

Use **CommitTransaction** only in a transaction.

An interpretation of the arguments is provided below.

pStatusBlockRet	is the memory address of the status block to which the status codes from the procedure return. (For detailed information, see the "Status Block" subsection at the beginning of this section.)
------------------------	--

SEE ALSO

BeginTransaction

NAME

CreateISAM -- create a new ISAM data set

SYNOPSIS

```
int CreateISAM (pbDSFile, cbDSFile, pbPWFileCreate,
               cbPWFileCreate, pbIXFile, cbIXFile,
               pbPWIXCreate, cbPWIXCreate, pISAMDesc,
               sISAMDesc, pStatusBlockRet )
short  cbDSFile, cbPWFileCreate, cbIXFile, cbPWIXCreate,
        sISAMDesc;
char    *pbDSFile, *pbPWFileCreate, *pbIXFile,
        *pbPWIXCreate, *pISAMDesc;
struct tStatusBlock *pStatusBlockRet;
```

DESCRIPTION

The CreateISAM procedure defines the index structure of a data set and creates a new empty data set with the specified structure.

An interpretation of the arguments is provided below.

pbDSFile	Describe a file name for the new
cbDSFile	data set.
pbPWFileCreate	describe a password used when
cbPWFileCreate	creating the new data set.
pbIXFile	Reserved (not currently used).
cbIXFile	Value is always 0.
pbPWIXCreate	Reserved (not currently used).
cbPWIXCreate	Value is always 0.
pISAMDesc	describe a memory area containing
sISAMDesc	an ISAM Description Block. Table
	2-2 shows the structure of the ISAM
	Description Block.
pStatusBlockRet	is the memory address of the status
	block to which the status codes
	from the procedure return. (For
	detailed information, see the
	"Status Block" subsection at the
	beginning of this section.)

Procedures

NAME

DeleteByKeyISAMRecord -- delete a record by a key

SYNOPSIS

```
int DeleteByKeyISAMRecord( ISAMHandle, iIndex, pKey, sKey,  
                           pUriRecordRet, pStatusBlockRet )  
    unsigned short ISAMHandle, iIndex;  
    short sKey;  
    char *pKey;  
    unsigned *pUriRecordRet;  
    struct tStatusBlock *pStatusBlockRet;
```

DESCRIPTION

The DeleteByKeyISAMRecord procedure, like the DelISAMRecord procedure, deletes a record from a data set. The data set must be open for modification. DeleteByKeyISAMRecord, however, uses a unique key to identify the record instead of a unique record identifier.

The disk space occupied by the record is made available for a subsequent StoreISAMRecord procedure. ISAM automatically deletes all keys for one record from the indices of the data set, and destroys all data in the record.

If the data set is open in a transaction mode, DeleteByKeyISAMRecord can be called only from within a transaction. The record to be deleted does not have to be specially locked, since DeleteByKeyISAMRecord locks the record before deleting it.

An interpretation of the arguments is provided below.

ISAMHandle	is the ISAM handle that identifies the open data set.
iIndex	identifies the index used. (The indices are numbered from 0 in the order specified in CreateISAM.)
pKey sKey	describe the memory area containing the key that identifies the record to be deleted. sKey must be the correct length, in bytes, of the key for index iIndex.
pUriRecordRet	is the memory address of the 4-byte structure to which the unique record identifier for the record to be deleted returns.

Procedures

pStatusBlockRet

is the memory address of the status block to which the status codes from the procedure return. (For detailed information, see the "Status Block" subsection at the beginning of this section.)

WARNINGS

The unique record ID returned is not a valid identifier for future ISAM calls, since the record has been deleted.

SEE ALSO

DelISAMRecord

Procedures

NAME

DeleteISAM -- delete an open data set

SYNOPSIS

```
int DeleteISAM( ISAMHandle, pStatusBlockRet )
unsigned short ISAMHandle;
struct tStatusBlock *pStatusBlockRet;
```

DESCRIPTION

The DeleteISAM procedure closes and deletes a data set, thereby destroying all information in the data set. An ISAM handle for the data set is not valid after DeleteISAM is called.

ISAM must open the data set in administrator mode or batch modify mode.

An interpretation of the arguments is provided below.

ISAMHandle	is the ISAM handle that identifies the open data set.
pStatusBlockRet	is the memory address of the status block to which the status codes from the procedure return. (For detailed information, see the "Status Block" subsection at the beginning of this section.)

NAME

DelISAMRecord -- delete a record by its URI

SYNOPSIS

```
int DelISAMRecord( ISAMHandle, uriRecord, pStatusBlockRet )
unsigned short   ISAMHandle;
unsigned uriRecord;
struct tStatusBlock *pStatusBlockRet;
```

DESCRIPTION

The DelISAMRecord procedure deletes a record from a data set. ISAM must open the data set for modification or in administrator mode. The disk space occupied by the record is made available for a subsequent StoreISAMRecord procedure. ISAM automatically deletes all keys for the record from the indices in the data set. ISAM destroys all data in the record.

If the data set is open in a transaction mode, DelISAMRecord can be called only from within a transaction.

An interpretation of the arguments is provided below.

ISAMHandle is the ISAM handle that identifies the open data set.

uriRecord is the unique record identifier of the record that is to be deleted. A previous read or get procedure usually returns uriRecord.

pStatusBlockRet is the memory address of the status block to which the status codes from the procedure return. (For detailed information, see the "Status Block" subsection at the beginning of this section.)

SEE ALSO

DeleteByKeyISAMRecord

Procedures

NAME

GetISAMRecords -- read several records and/or their URIs in
key order
GetHoldISAMRecords -- read several records and/or
their URIs in key order

SYNOPSIS

```
int GetISAMRecords( ISAMHandle, fReadRecords, pBuffer,  
                   sBuffer, pCRecordsReadRet,  
                   pStatusBlockRet )  
unsigned short ISAMHandle;  
short sBuffer, *pCRecordsReadRet;  
char fReadRecords, *pBuffer;  
struct tStatusBlock *pStatusBlockRet;  
  
int GetHoldISAMRecords(ISAMHandle, fReadRecords, pBuffer,  
                       sBuffer, pCRecordsReadRet,  
                       pStatusBlockRet )  
unsigned short ISAMHandle;  
short sBuffer, *pCRecordsReadRet;  
char fReadRecords, *pBuffer;  
struct tStatusBlock *pStatusBlockRet;
```

DESCRIPTION

The GetISAMRecords procedure reads several records and/or their unique record identifiers in key order from a data set with a single call to ISAM. Records return in key order for the current iteration. If there are no more records for the iteration, the status code NO_MORE_RECS returns.

The GetHoldISAMRecords procedure is identical to GetISAMRecords, except that GetHoldISAMRecords can only be called in a transaction when the data set is open in the transaction mode.

The GetISAMRecords and GetHoldISAMRecords procedures return as many records (or unique record identifiers) in sequence as possible, subject to the following constraints:

- o The buffer capacity will not be exceeded.
- o The range specified for the current iteration will not be exceeded.
- o Records locked by other applications will not be read.

Procedures

For example, a particular GetISAMRecords procedure reads 4-byte unique record identifiers and 40-byte records into a 500-byte buffer. The buffer can contain up to 11 records and unique record identifiers ($440 + 44$). If another application locks the seventh record in the sequence, then the GetISAMRecords procedure returns only the first six records and unique record identifiers.

If GetISAMRecords is called to read only unique record identifiers, it returns unique record identifiers for locked records. Applications can thus skip over records locked by other applications.

An interpretation of the arguments is provided below.

ISAMHandle	is the ISAM handle that identifies the open data set.
fReadRecords	specifies whether records as well as unique record identifiers are returned. fReadRecords is FALSE if only unique record identifiers return, and TRUE if records return also.
pBuffer sBuffer	describe the memory area to which the unique record identifiers and records are read.

If only unique record identifiers return (fReadRecords is FALSE), then the record identifiers are packed into the buffer without padding. If both unique record identifiers and records return (fReadRecords is TRUE), then the unique record identifiers and records are packed together into the buffer without padding. Each record and its record identifier are packed as a pair with the record identifier preceding the record.

Table 4-3 shows the buffer structure after three 46-byte records are read.

Procedures

Table 4-3. Buffer Structure for
GetISAMRecords and GetHoldISAMRecords When
Records Are Read (46-Byte Records)

<u>Offset</u>	<u>Description</u>	<u>Size (bytes)</u>
0	Record identifier of Record 1	4
4	Record 1	46
50	Record identifier of Record 2	4
54	Record 2	46
100	Record identifier of Record 3	4
104	Record 3	46

pcRecordsReadRet

is the memory address of the short word to which the number of unique record identifiers and/or records read returns.

pStatusBlockRet

is the memory address of the status block to which the status codes from the procedure return. (For detailed information, see the "Status Block" subsection at the beginning of this section.)

SEE ALSO

SetUpLimitsISAMIteration, SetUpPrefixIteration

NAME

HoldISAMDataSet -- lock an ISAM data set

SYNOPSIS

```
int HoldISAMDataSet( ISAMHandle, fLockRecords,
                    pStatusBlockRet )
unsigned short ISAMHandle;
char fLockRecords;
struct tStatusBlock *pStatusBlockRet;
```

DESCRIPTION

The HoldISAMDataSet procedure locks an open data set.

If the data set is open in a transaction mode, HoldISAMDataSet can be called only from within a transaction.

An interpretation of the arguments is provided below.

ISAMHandle is the ISAM handle that identifies the open data set.

fLockRecords specifies whether a subsequent ReleaseDataSet procedure unlocks the data set's locked records as well as the data set itself. fLockRecords is either TRUE or FALSE.

If fLockRecords is TRUE, then if a subsequent ReleaseDataSet procedure unlocks the data set, locked records in the data set remain locked.

If fLockRecords is FALSE, then if the ReleaseDataSet procedure unlocks the data set, all records in the data set will be unlocked.

For example, DataSet X has five records and the following events occur:

1. The ISAMHoldRecord procedure locks Record 1.
2. The HoldISAMDataSet procedure locks Data Set X.
3. The RdHoldNextISAMRecord procedure locks Record 2.
4. The ReleaseDataSet procedure unlocks Data Set X.

Procedures

If `fLockRecords` is TRUE in step 2, then Record 1 and Record 2 remain locked after step 4. If `fLockRecords` is FALSE in step 2, then Record 1 and Record 2 are unlocked after step 4.

`pStatusBlockRet` is the memory address of the status block to which the status codes from the procedure return. (For detailed information, see the "Status Block" subsection at the beginning of this section.)

WARNINGS

This procedure should only be used when absolutely necessary. Overuse of this procedure on concurrently accessed data sets will cause a significant rise in transaction failure.

SEE ALSO

`ReleaseDataSet`, `ISAMHoldRecord`

NAME

iserror -- output ISAM error to the standard error stream

SYNOPSIS

```
int iserror(msg, status)
char *msg;
struct ISAMStatusBlock *status;
```

DESCRIPTION

The iserror procedure prints the user supplied message followed by a colon, a space, and a description of the error in the status block.

RETURN VALUE

iserror always returns one (1).

EXAMPLE

```
if(CloseISAM(handle, & StatusBlock) !=0)
{
    exit(iserror("Trying to close file", &StatusBlock));
}
might print the following:
```

Trying to close file: Bad Handle.

Procedures

NAME

ISAMHoldRecord -- lock a record by its URI

SYNOPSIS

```
int ISAMHoldRecord( ISAMHandle, uriRecord, pStatusBlockRet )
unsigned short  ISAMHandle;
unsigned uriRecord;
struct tStatusBlock *pStatusBlockRet;
```

DESCRIPTION

The ISAMHoldRecord procedure locks a record identified by its unique record identifier. The record identifier is not checked for validity by ISAMHoldRecord.

If the data set is open in a transaction mode, ISAMHoldRecord can be called only from within a transaction.

An interpretation of the arguments is provided below.

ISAMHandle	is the ISAM handle that identifies the open data set.
uriRecord	is the unique record identifier of the record that is to be locked. A previous read or get procedure usually returns uriRecord.
pStatusBlockRet	is the memory address of the status block to which the status codes from the procedure return. (For detailed information, see the "Status Block" subsection at the beginning of this section.)

SEE ALSO

HoldISAMDataSet

NAME

ModifyByKeyISAMRecord -- modify an existing record by a unique key and update all indices accordingly

SYNOPSIS

```
int ModifyByKeyISAMRecord( ISAMHandle, iIndex, pRecord,
                           sRecord, pUriRecordRet,
                           pStatusBlockRet )
unsigned short ISAMHandle, iIndex;
short sRecord;
char *pRecord;
unsigned *pUriRecordRet;
struct tStatusBlock *pStatusBlockRet;
```

DESCRIPTION

The **ModifyByKeyISAMRecord** procedure, like the **ModifyISAMRecord** procedure, modifies an existing record in a data set and updates all indices accordingly. **ModifyByKeyISAMRecord**, however, uses a unique key to identify the record instead of a unique record identifier.

If any key changes, then the record is removed from the index under the old value of the key and reindexed under the new value. If the new value of the key duplicates an existing key for the same index in another record, but duplicates are not allowed for that index, then the record is not modified, the index is not changed, and status code **DUP_KEY** is returned.

The parameters for this procedure include the index and the new record contents, but not a separate key for the record. The key that identifies the record is taken from the record itself.

For example, to change the salary of employee 150 in the Employee data set from \$34,000 to \$37,500, **iIndex** is specified as 0. (**empNo** is index 0, a unique key index.) ISAM extracts the **empNo** field, uses the **empNo** index to determine which record is to be modified, and modifies the record.

The data set must be open for modification.

If the data set is open in a transaction mode, **ModifyByKeyISAMRecord** can be called only from within a transaction. The record to be modified needs to be locked before being modified.

Procedures

An interpretation of the arguments is provided below.

ISAMHandle	is the ISAM handle that identifies the open data set.
iIndex	identifies the index used. (The indices are numbered from 0 in the order specified in CreateISAM.)
pRecord sRecord	describe the memory area containing the record that is to be modified. sRecord must be equal to the record size for the data set.
pUriRecordRet	is the memory address of the 4-byte structure to which the unique record identifier of the modified record returns.
pStatusBlockRet	is the memory address of the status block to which the status codes from the procedure return. (For detailed information, see the "Status Block" subsection at the beginning of this section.)

NOTES

This procedure cannot change the key being used to identify the record.

SEE ALSO

ModifyISAMRecord

Procedures

NAME

`ModifyISAMRecord` -- modify an existing record by its URI and update all indices accordingly

SYNOPSIS

```
int ModifyISAMRecord( ISAMHandle, uriRecord, pRecord,
                      sRecord, pStatusBlockRet)
unsigned short ISAMHandle;
unsigned uriRecord;
char *pRecord;
short sRecord;
struct tStatusBlock *pStatusBlockRet;
```

DESCRIPTION

The `ModifyISAMRecord` procedure modifies an existing record in a data set and updates all indices accordingly. The record's unique record identifier identifies the record.

If any key changes, then the procedure removes the old value from the index and reindexes the record under the new value. If the new value of the key duplicates an existing key for the same index in another record and duplicates are not allowed for that index, then the procedure does not modify the record, the index is not changed, and status code `DUP_KEY` is returned.

The data set must be open for modification.

If the data set is open in a transaction mode, `ModifyISAMRecord` can be called only from within a transaction.

An interpretation of the arguments is provided below.

<code>ISAMHandle</code>	is the ISAM handle that identifies the open data set.
<code>uriRecord</code>	is the unique record identifier of the record that is to be modified. A previous read or get procedure usually returns the <code>uriRecord</code> .
<code>pRecord</code> <code>sRecord</code>	describe the memory area containing the record to be written. <code>sRecord</code> must be equal to the record size for the data set.

Procedures

pStatusBlockRet

is the memory address of the status block to which the status codes from the procedure return.

(For detailed information, see the "Status Block" subsection at the beginning of this section.)

NAME

OpenISAM -- open an existing data set

SYNOPSIS

```
int OpenISAM( pISAMHandleRet, pbDataSetName, cbDataSetName,
              pbPassWord, cbPassWord, mode, sRecord,
              pStatusBlockRet )
unsigned short *pISAMHandleRet, mode;
short cbDataSetName, cbPassWord, sRecord;
char *pbDataSetName, *pbPassWord;
struct tStatusBlock *pStatusBlockRet;
```

DESCRIPTION

The OpenISAM procedure opens an existing data set and returns an ISAM handle for it. The ISAM handle is used to refer to the data set in subsequent procedures.

An interpretation of the arguments is provided below.

pISAMHandleRet	is the memory address of the short word to which the ISAM handle of the opened data set is to be returned.
pbDataSetName cbDataSetName	describe the memory area containing the character string representing the name of the data set.
pbPassword cbPassword	describe either the modify or read password for the data set that is opened. If the mode is batch modify or transaction modify, then the modify password must be supplied. If the mode is batch read or transaction read, then either password may be supplied.

Procedures

mode

specifies the mode in which the data set is to be opened:

<u>mode (ASCII)</u>	<u>dec</u>	<u>hex</u>
administrator (MD_ADMIN)	24932	6164x
batch modify (MD_BMOD)	25197	626dx
transaction read (MD_TREAD)	29810	7472x
transaction modify (MD_TMOD)	29805	746dx
batch read (MD_BREAD)	25202	6272x

In these ASCII constants, the first character is the high-order byte, and the second character is the low-order byte.

For further information about modes, refer to the subsection called "Data Set Sharing" in Section 2.

sRecord

is the fixed-length record size for the data set. This is checked against the actual record size of the data set.

pStatusBlockRet

is the memory address of the status block to which the status codes from the procedure return. (For detailed information, see the "Status Block" subsection at the beginning of this section.)

WARNINGS

The user must be the ISAM administrator to open a data set in administrator mode.

NAME

QueryTransactionParams -- access parameters associated with transactions for the application

SYNOPSIS

```
int QueryTransactionParams( pParamsRet, sParamsMax,
                           pSPParamsRet, pStatusBlockRet )
char *pParamsRet;
short sParamsMax, *pSPParamsRet;
struct tStatusBlock *pStatusBlockRet;
```

DESCRIPTION

The **QueryTransactionParams** procedure retrieves the application's current transaction parameters. (See the subsection called "Time-Outs" in Section 2.)

An interpretation of the arguments is provided below.

pParamsRet sParamsMax	describe the memory area in which the transaction parameters will be stored. Table 2-7 shows the format of the transaction parameters block.
--------------------------	--

pSPParamsRet	is the memory address of a short word to which the number of bytes retrieved will be stored. The value stored in this short word will not exceed sParamsMax.
--------------	--

pStatusBlockRet	is the memory address of the status block to which the status codes from the procedure return. (For detailed information, see the "Status Block" subsection at the beginning of this section.)
-----------------	--

SEE ALSO

SetTransactionParams

Procedures

NAME

RdNextISAMRecord -- read the next record in key order
RdHoldNextISAMRecord -- read and lock the next record in key order

SYNOPSIS

```
int RdNextISAMRecord( ISAMHandle, pRecordRet, sRecord,
                     pUriRecordRet, pStatusBlockRet )
unsigned short ISAMHandle;
char *pRecordRet;
short sRecord;
unsigned *pUriRecordRet;
struct tStatusBlock *pStatusBlockRet;

int RdHoldNextISAMRecord( ISAMHandle, pRecordRet, sRecord,
                         pUriRecordRet, pStatusBlockRet )
unsigned short ISAMHandle;
char *pRecordRet;
short sRecord;
unsigned *pUriRecordRet;
struct tStatusBlock *pStatusBlockRet;
```

DESCRIPTION

The RdNextISAMRecord procedure reads the next record in key order from a data set, within a range previously set up by SetUpLimitsISAMIteration or SetupPrefixIteration. The unique record identifier of the record is returned.

The RdHoldNextISAMRecord procedure is identical to RdNextISAMRecord, except RdHoldNextISAMRecord also locks the record when it is read.

If the data set is open in a transaction mode, RdHoldNextISAMRecord can be called only from within a transaction.

An interpretation of the arguments is provided below.

ISAMHandle	is the ISAM handle that identifies the open data set.
pRecordRet sRecord	describe the memory area to which the record is returned. sRecord must be equal to the record size for the data set.
pUriRecordRet	is the memory address of the 4-byte structure to which the unique record identifier of the record returns.

Procedures

pStatusBlockRet

is the memory address of the status block to which the status codes from the procedure return.

(For detailed information, see the "Status Block" subsection at the beginning of this section.)

SEE ALSO

RdUniqueISAMRecord
ReadISAMRecordByUri

Procedures

NAME

RdUniqueISAMRecord -- read a record by a unique key
RdHoldUniqueISAMRecord -- read and lock a record by a unique key

SYNOPSIS

```
int RdUniqueISAMRecord( ISAMHandle, iIndex, pKey, sKey,
                        pRecordRet, sRecord, pUriRecordRet,
                        pStatusBlockRet )
unsigned short ISAMHandle, iIndex;
short sRecord, sKey;
unsigned *pUriRecordRet;
char *pKey, *pRecordRet;
struct tStatusBlock *pStatusBlockRet;

int RdHoldUniqueISAMRecord( ISAMHandle, iIndex, pKey, sKey,
                           pRecordRet, sRecord,
                           pUriRecordRet, pStatusBlockRet )
unsigned short ISAMHandle, iIndex;
short sRecord, sKey;
unsigned *pUriRecordRet;
char *pKey, *pRecordRet;
struct tStatusBlock *pStatusBlockRet;
```

DESCRIPTION

The RdUniqueISAMRecord procedure reads a record uniquely identified by a given key from a data set. The index does not allow duplicates, so the key appears at most in one record of the data set.

The RdHoldUniqueISAMRecord procedure is identical to RdUniqueISAMRecord, except that RdHoldUniqueISAMRecord also locks the record.

If the data set is opened in a transaction mode, RdHoldUniqueISAMRecord can be called only from within a transaction.

An interpretation of the arguments is provided below.

ISAMHandle	is the ISAM handle that identifies the open data set.
iIndex	identifies the index used. (The indices are numbered from 0 in the order specified in CreateISAM.)

Procedures

pKey sKey	describe the memory area containing the unique key that identifies the record to be read. sKey must be the correct length, in bytes, of keys for the index, iIndex.
pRecordRet sRecord	describe the memory area into which the record is returned. sRecord must be equal to the record size for the data set.
pUriRecordRet	is the memory address of the 4-byte structure to which the unique record identifier of the record returns.
pStatusBlockRet	is the memory address of the status block to which the status codes from the procedure return. (For detailed information, see the "Status Block" subsection at the beginning of this section.)

SEE ALSO

RdNextISAMRecord
ReadISAMRecordByUri

Procedures

NAME

ReadISAMRecordByUri -- read a record by its URI
ReadHoldISAMRecordByUri -- read and lock a record by its URI

SYNOPSIS

```
int ReadISAMRecordByUri( ISAMHandle, uriRecord, pRecordRet,  
                        sRecord, pStatusBlockRet )  
  
unsigned short ISAMHandle;  
unsigned uriRecord;  
char *pRecordRet;  
short sRecord;  
struct tStatusBlock *pStatusBlockRet;  
  
int ReadHoldISAMRecordByUri( ISAMHandle, uriRecord,  
                             pRecordRet, sRecord,  
                             pStatusBlockRet )  
  
unsigned short ISAMHandle;  
unsigned uriRecord;  
char *pRecordRet;  
short sRecord;  
struct tStatusBlock *pStatusBlockRet;
```

DESCRIPTION

The ReadISAMRecordByUri procedure reads a record identified by its unique record identifier.

The ReadHoldISAMRecordByUri procedure is identical to ReadISAMRecordByUri, except ReadHoldISAMRecordByUri also locks the record when it is read.

If the data set is open in a transaction mode, ReadHoldISAMRecordByUri can be called only from within a transaction.

An interpretation of the arguments is provided below.

ISAMHandle	is the ISAM handle that identifies the open data set.
uriRecord	is the unique record identifier of the record to be read. A previous read or get procedure usually returns the uriRecord.
pRecordRet sRecord	describe the memory area to which the record is returned. sRecord must be equal to the record size for the data set.

Procedures

pStatusBlockRet

is the memory address of the status block to which the status codes from the procedure return. (For detailed information, see the "Status Block" subsection at the beginning of this section.)

SEE ALSO

ReadNextISAMRecord
RdUniqueISAMRecord

Procedures

NAME

ReleaseDataSet -- unlock a locked data set without ending the current transaction

SYNOPSIS

```
int ReleaseDataSet( ISAMHandle, pStatusBlockRet )
unsigned short ISAMHandle;
struct tStatusBlock *pStatusBlockRet;
```

DESCRIPTION

The **ReleaseDataSet** procedure releases a locked data set without ending the current transaction.

If the data set is open in a transaction mode, **ReleaseDataSet** can be called only from within a transaction.

An interpretation of the arguments is provided below.

ISAMHandle is the ISAM handle that identifies the open data set.

pStatusBlockRet is the memory address of the status block to which the status codes from the procedure return. (For detailed information, see the "Status Block" subsection at the beginning of this section.)

WARNINGS

This procedure should be used with care, since other applications can hold and even modify the released data set before the transaction is ended.

NAME

ReleaseISAMRecord -- unlock a locked record without ending the current transaction

SYNOPSIS

```
int ReleaseISAMRecord( ISAMHandle, uriRecord,
                       pStatusBlockRet )
unsigned short ISAMHandle;
unsigned uriRecord;
struct tStatusBlock *pStatusBlockRet;
```

DESCRIPTION

The ReleaseISAMRecord procedure releases a locked record without ending the current transaction.

If the data set is open in a transaction mode, ReleaseISAMRecord can be called only from within a transaction.

ISAMHandle	is the ISAM handle that identifies the open data set.
uriRecord	is the unique record identifier for the record that is to be released. A previous read or get procedure usually returns the uriRecord.
pStatusBlockRet	is the memory address of the status block to which the status codes from the procedure return. (For detailed information, see the "Status Block" subsection at the beginning of this section.)

WARNINGS

This procedure should be used with care, since other applications can hold and even modify the released record or data set before the transaction is ended.

Procedures

NAME

RenameISAM -- change the name of an existing data set

SYNOPSIS

```
int RenameISAM( ISAMHandle, pbDataStoreFile,
                cbDataStoreFile, pbPWDataStoreFileRename,
                cbPWDataStoreFileRename, pbIndexFile,
                cbIndexFile, pbPWIndexFileRename,
                cbPWIndexFileRename, pStatusBlockRet )
unsigned short ISAMHandle;
short cbDataStoreFile, cbPWDataStoreFileRename,
      cbIndexFile, cbWIndexFileRename;
char *pbDataStoreFile, *pbPWDataStoreFileRename,
      *pbIndexFile, *pbPWIndexFileRename;
struct tStatusBlock *pStatusBlockRet;
```

DESCRIPTION

The RenameISAM procedure changes the name of an existing data set.

The data set must be open in administrator mode.

An interpretation of the arguments is provided below.

ISAMHandle	is the ISAM handle that identifies the open data set.
pbDataStoreFile cbDataStoreFile	describe the file name that specifies the new name for the data set.
pbPWDataStoreFileRename cbPWDataStoreFileRename	Reserved (not currently used). Value is always 0.
pbIndexFile cbIndexFile	Reserved (not currently used). Value is always 0.
pbPWIndexFileRename cbPWIndexFileRename	Reserved (not currently used). Value is always 0.
pStatusBlockRet	is the memory address of the status block to which the status codes from the procedure return. (For detailed information, see the "Status Block" subsection at the beginning of this section.)

NOTE

RenameISAM can only rename files within a single UNIX file system. Otherwise, use the UNIX move utility (mv).

NAME

SetISAMProtection -- change the password for an existing data set

SYNOPSIS

```
int SetISAMProtection( ISAMHandle, pbPWOpnRead, cbPWOpnRead,
                      pbPWOpnModify, cbPWOpnModify,
                      pStatusBlockRet)
unsigned short ISAMHandle;
short cbPWOpnRead, cbPWOpnModify;
char *pbPWOpnRead, *pbPWOpnModify;
struct tStatusBlock *pStatusBlockRet;
```

DESCRIPTION

The SetISAMProtection procedure changes the passwords that allow an application to gain access to an open data set.

The data set must be open in administrator mode.

An interpretation of the arguments is provided below.

ISAMHandle	is the ISAM handle that identifies the open data set.
pbPWOpnRead cbPWOpnRead	describe the password that opens the data set for reading in either batch read or transaction read modes. The password cannot be longer than 8 bytes.
pbPWOpnModify cbPWOpnModify	describe the password that opens the data set for modification in either batch modify or transaction modify modes. This password can also be used to open the data set for reading in batch read or transaction read modes. The password cannot be longer than 8 bytes.
pStatusBlockRet	is the memory address of the status block to which the status codes from the procedure return. (For detailed information, see the "Status Block" subsection at the beginning of this section.)

WARNINGS

The passwords allowed by UNIX PC ISAM are only 8 bytes long and must be taken from the ASCII character set.

Procedures

NAME

SetTransactionParams -- set parameters associated with transactions for the program

SYNOPSIS

```
int SetTransactionParams( pParams, sParams,  
                          pStatusBlockRet )  
  
char    *pParams;  
short   sParams;  
struct tStatusBlock *pStatusBlockRet;
```

DESCRIPTION

The **SetTransactionParams** procedure sets the application's current transaction parameters. (See the subsection called "Time-Outs" in Section 2.)

An interpretation of the arguments is provided below.

pParams	describe the memory area containing the transaction parameters to be set. The transaction parameters block has the format shown in Table 2-7. If sParams is less than the length of the Transaction Parameters Block, then (1) the supplied block must not include a partial field at the end, and (2) only parameters included in the block are changed.
sParams	

pStatusBlockRet	is the memory address of the status block to which the status codes from the procedure return. (For detailed information, see the "Status Block" subsection at the beginning of this section.)
------------------------	--

SEE ALSO

QueryTransactionParams

NAME

SetUpLimitsISAMIteration -- initialize a sequence of read operations for records with a specific key within a given range

SYNOPSIS

```
int SetUpLimitsISAMIteration( ISAMHandle, iIndex, matchKind,
                             pKey1, sKey1, pKey2, sKey2,
                             pStatusBlockRet )
unsigned short ISAMHandle, iIndex, matchKind;
short sKey1, sKey2;
char *pKey1, *pKey2;
struct tStatusBlock *pStatusBlockRet;
```

DESCRIPTION

The **SetUpLimitsISAMIteration** procedure initializes a sequence of read procedures for records that have keys for a specified index within a given range. Subsequent calls to **GetISAMRecords**, **GetHoldISAMRecords**, **RdNextISAMRecord**, and **RdHoldNextISAMRecord** procedures read each record that has a key value within the range. Records are read in key value order.

If the index defined does not include null values (binary 0s), then records with null keys are not read.

An interpretation of the arguments is provided below.

ISAMHandle	is the ISAM handle that identifies the open data set.
iIndex	identifies the index used. (The indices are numbered from 0 in the order specified in CreateISAM .)
matchKind	is a short word specifying which records will be retrieved. The names (defined in isam.h) are shown in Table 4-4.

Procedures

Table 4-4. MatchKind Words for Record Retrieval

Name	Value	Description
MK_ALL	0	All records, regardless of their key values. key1 and key2 are both ignored; sKey1 and sKey2 should be 0.
MK_LT	1	All records containing a key less than key1. key2 is ignored, and sKey2 should be 0.
MK_LE	2	All records containing a key less than or equal to key1. key2 is ignored, and sKey2 should be 0.
MK_EQ	3	All records containing a key equal to key1. key2 is ignored, and sKey2 should be 0.
MK_GE	4	All records containing a key greater than or equal to key1. key2 is ignored, and sKey2 should be 0.
MK_GT	5	All records containing a key greater than key1. key2 is ignored, and sKey2 should be 0.
MK_GTLT	6	All records containing a key greater than key1 but less than key2.
MK_GELT	7	All records containing a key greater than or equal to key1 but less than key2.
MK_GELE	8	All records containing a key greater than or equal to key1 but less than or equal to key2.
MK_GTLE	9	All records containing a key greater than key1 but less than or equal to key2.

Procedures

pKey1
sKey1 describe the memory area containing a key record. The key affects the set of records returned by subsequent read procedures. (See matchKind above.)

pKey2
sKey2 describe the memory area containing a key record. The key affects the set of records returned by subsequent read procedures. (See matchKind above.)

pStatusBlockRet is the memory address of the status block to which the status codes from the procedure return. (For detailed information, see the "Status Block" subsection at the beginning of this section.)

Procedures

NAME

SetUpPrefixIteration -- initialize a sequence of read operations for records with a specific byte or character string key having a given prefix

SYNOPSIS

```
int SetUpPrefixIteration( ISAMHandle, iIndex, pKey, sKey,
                          pStatusBlockRet )
unsigned short ISAMHandle, iIndex;
short sKey;
char *pKey;
struct tStatusBlock *pStatusBlockRet;
```

DESCRIPTION

The **SetUpPrefixIteration** procedure initializes a sequence of read procedures for records with keys for a specified index having a given prefix. Subsequent calls to **GetISAMRecords**, **GetHoldISAMRecords**, **RdNextISAMRecord**, and **RdHoldNextISAMRecord** procedures read each record for which the given key is a prefix of the key stored in the record.

The index must contain either byte string or character string keys.

If the index defined does not include null values (binary zeros), then records with null keys are not read.

An interpretation of the arguments is provided below.

ISAMHandle	is the ISAM handle that identifies the open data set.
iIndex	identifies the index used. (The indices are numbered from 0 in the order specified in CreateISAM .)
pKey sKey	describe the memory area containing the key. sKey must be no larger than the length of the keys for index, iIndex .
pStatusBlockRet	is the memory address of the status block to which the status codes from the procedure return. (For detailed information, see the "Status Block" subsection at the beginning of this section.)

NAME

StoreISAMRecord -- create a new record with specified data
in a data set

SYNOPSIS

```
int StoreISAMRecord( ISAMHandle, pRecord, sRecord,
                    pUriRecordRet, pStatusBlockRet )
unsigned short ISAMHandle;
char *pRecord;
short sRecord;
unsigned *pUriRecordRet;
struct tStatusBlock *pStatusBlockRet
```

DESCRIPTION

The StoreISAMRecord procedure creates a new record in a data set. If necessary, the length of the data set is increased to accommodate the new record.

The indices are updated as required to reflect the presence of the new record. If the value of the new key duplicates an existing key in the same index, and duplicates are not allowed for that index, then the record is not stored and the status code DUP_KEY is returned.

The data set must be open for modification.

If the data set is open in transaction mode, StoreISAMRecord can be called only from within a transaction.

An interpretation of the arguments is provided below.

ISAMHandle	is the ISAM handle that identifies the open data set.
pRecord sRecord	describe the memory area containing the record to be written. sRecord must be equal to the record size for the data set.
pUriRecordRet	is the memory address of the 4-byte structure to which the unique record identifier of the record to be stored returns.
pStatusBlockRet	is the memory address of the ISAM Status Block to which the status codes from the procedure return. (For detailed information, see the "Status Block" subsection at the beginning of this section.)

Procedures

NAME

VerifyMultiUserISAM -- establish availability of ISAM

SYNOPSIS

```
int VerifyMultiUserISAM( pbNode, cbNode, pStatusBlockRet )
char    *pbNode;
short   cbNode;
struct tStatusBlock *pStatusBlockRet;
```

DESCRIPTION

The VerifyMultiUserISAM procedure sends a request to ISAM to verify that ISAM is available. If ISAM is installed, a status code of 0 (OK) returns. Any other status code indicates that multiuser ISAM is not available.

An interpretation of the arguments is provided below.

pbNode	Reserved (not currently used).
cbNode	
pStatusBlockRet	is the memory address of the status block to which the status codes from the procedure return. (For detailed information, see the "Status Block" subsection at the beginning of this section.)

SECTION 5: SYSTEM ADMINISTRATION

This section provides the background information of particular interest to the ISAM administrator and discusses the utilities provided for maintaining ISAM.

ADMINISTRATOR'S HOME DIRECTORY

In order to maintain a UNIX system running UNIX PC ISAM, there must be a special login set up called `isam`. A user logged into the system as `isam` is referred to as the ISAM administrator. The ISAM administrator's home directory is the default directory for all ISAM files created without an explicit pathname. The `isam` home directory also contains `ErrorLog`, the log of hard errors detected by ISAM. The administrator should check this file frequently to make sure that hard errors are not happening. This is especially important if users report that they are receiving `IO_ERROR` or `REC_FORM` returns from their ISAM calls. The ISAM administrator also maintains the configuration information that the server uses to determine the various parameters which it needs at run time.

THE SERVER

The UNIX PC ISAM system is based around the server. The server is a separate UNIX process, which receives requests from the various user processes that need ISAM services. These requests are sent from within the library functions contained in `libisam.a` and cannot be sent by user programs in any other manner. The server is typically started by the standard UNIX startup script contained in `/etc/rc`. (The installation procedure is described in detail below.) The server is actually the program `/usr/lib/isam/isam`. The directory `usr/lib/isam` also contains all the ISAM utilities.

The server always runs under the ISAM administrator and all files it creates are owned by the ISAM administrator. ISAM files are not readable or writable by any user other than the administrator through the ISAM server. No user can break the security of an

System Administration

ISAM data set without knowing its password. Users are allowed to create and move files in their own directories if they give the administrator write permission in those directories. They may also, of course, remove files using the UNIX remove utility (rm) if they have write permission in the directory where the files are stored. Since only the ISAM administrator has read/write permission for ISAM data sets in the home directory, other users cannot manipulate files in the ISAM administrator's directory. To start the ISAM server from a UNIX terminal, login as isam and run the server as /usr/lib/isam/isam. The server starts and the terminal displays the shell prompt. Many installations will want to start the ISAM server as part of the normal boot procedure. To include the ISAM server in the normal boot procedure, append the following two lines to the /etc/rc file (refer to the rc documentation in the AT&T UNIX PC System V User's Manual):

```
su isam -c "/usr/lib/isam/isam"
(message -i "ISAM server started. \n") &
```

The ISAM server is responsible for the integrity of all open data sets, and should not be killed while ISAM application programs are running.

SERVER CONFIGURATION

The ISAM server has several configurable parameters which are set each time the server is started. These parameters are specified in the configuration file. This file is an ASCII formatted text file called /usr/lib/isam/IsamConfig. The administrator can maintain this file using any standard UNIX text editor. Any of the UNIX utilities can be used to display the configuration file. For example, to use the cat utility to display the configuration file, type:

```
cat IsamConfig
```

after the shell prompt on your terminal screen, and press the Return key. An example configuration file looks like

```
# ISAM Configuration file for distribution.
# @(#)IsamConfig 1.1
```

BlockSize	1	# installation block size in kilobytes.
Buffers	64	# number of server buffers.
OpenFiles	32	# maximum number of user open files.
CommonFiles	10	# maximum UNIX open files.
Retained	0	# number of files to keep open.
Clients	10	# maximum number of clients.
WriteThru	1	# 1 = write through cache, 0 = otherwise.

System Administration

The format of the file is straightforward--each line contains a keyword indicating the parameter described followed by an arbitrary number of spaces or tabs followed by the value assigned to the parameter.

To change a parameter value, edit the file and change the value you want to change. To insert comments in the file, use the # character. All characters between the # and the end of the line are ignored when the file is read. If there is an error in the file when the server is started, it stops immediately and displays an appropriate message. Blank lines and comments are ignored.

The following subsections describe each of the parameters in detail.

Installation Block Size

Keyword = BlockSize, Default = 1

The server assumes that all the files that it handles have a common block size. This size indicates the size of the pieces of the file over which ISAM will do its consistency checking. It is also the size of the buffers in the server, so ISAM reads and writes this size at once. The server rejects any ISAM data set that a user tries to open which has a block size different from the installation block size specified to the server at startup. To change the block size of a data set, the administrator must use the IsamReorg utility (see below).

This parameter is in multiples of 1 kilobyte and must be a power of 2 between 1 and 32. The parameter should be 1 on all file systems that do not use PILF.

Number of Server Buffers

Keyword = Buffers, Default = 64

This parameter determines the size of the server's internal cache. The total size of the cache equals the installation block size specified above times the number of server buffers times 1k. Thus, the default cache size is 64k. The larger the number of buffers, the less often ISAM actually reads from disk.

System Administration

Maximum Number of User Handles

Keyword = OpenFiles, Default = 32

This is the maximum number of handles that all the clients can have at the same time. The number of handles is greater than the number of UNIX open data sets since users can share UNIX file descriptors if they have the same data sets open. The proper value for this parameter is dependent on the load pattern at a particular installation.

Maximum Number of UNIX Open Data Sets

Keyword = CommonFiles, Default = 19

This is the maximum number of UNIX files that the server can maintain open at once.

Number of Open Files Retained

Keyword = Retained, Default = 0

For most application systems, this parameter should be 0. However, some systems may open and close the same files with great frequency. In this case, set the open file retention value to some small number greater than 0. The exact value is dependent on the particular application system. For example, if a system frequently opens and closes the same 2 files, Retained should be set at 2.

Maximum Number of Server Clients

Keyword = Clients, Default = 64

This parameter indicates the maximum number of concurrent clients that the ISAM server can serve. It is advisable to use the default value unless a system is likely to have more than 64 processes using the ISAM server.

Write-Through Cache Option

Keyword = WriteThru, Default = 1

This parameter describes the discipline that ISAM uses to write data from its internal buffers to disk. The value of this parameter should be set to 0 (zero) or 1. When WriteThru is 1, ISAM immediately writes all the blocks affected by a given ISAM operation. If WriteThru is 0, ISAM buffers the data for longer periods, resulting in less actual disk usage, and, therefore, higher throughput.

When the write-through cache option is not used, ISAM ensures under any circumstances that all data is written to disk within about a minute. The CloseISAM function guarantees that when it returns, all the outstanding data for that file has been written. Any user accessing a data set through ISAM gets the most up-to-date data even if it has not yet been written to disk.

— Note —

If WriteThru is disabled, up to one minute's worth of data base changes can be lost in the event of a power failure.

System Administration

ISAM UTILITY PROGRAMS

The UNIX PC ISAM system provides several utility programs for ease of maintenance. The utilities are shown in Table 5-1.

Table 5-1. ISAM Utilities

Utility	Description	Who Can Access
IsamCreate	A program which allows the creation of new data sets and the definition of indices.	Any ISAM User
IsamProtect	A program which allows the ISAM administrator to password protect ISAM files for security.	ISAM Administrator or "qualified" ISAM user*
IsamReorg	A program that performs a variety of maintenance functions on existing ISAM data sets so the administrator can salvage or transport data sets.	ISAM Administrator or "qualified" ISAM user*
IsamStop	A program which allows the ISAM administrator or the UNIX super user to shut down an active ISAM server.	ISAM Administrator or UNIX super user

*A qualified ISAM user is anyone who owns the directory which contains the only UNIX link to the ISAM data set and who can supply the modification password for the data set.

Detailed information about the utilities, including the commands that invoke them, is presented in the following subsections. To exit from any of the programs, press the Delete key or the Rubout key.

IsamCreate

The IsamCreate utility creates an ISAM data set to meet your specifications. To run IsamCreate, type:

IsamCreate

after the shell prompt on your terminal screen and press the Return key. To determine the basic information about how the file will look, the program then asks the following sequence of questions:

Data Set Name?
Record Size?
Number of Indices?

(See the subsection in Section 2 titled "ISAM Description Block" for a detailed description of each parameter.) After each of the above questions appears, enter the required value and press the Return key. For each index specified, IsamCreate asks the following set of questions:

Offset of start of key from beginning of record?
Size of Key?
Key Type?
Sort keys in ascending key sequence (Y or N)?
Include null keys in index (Y or N)?
Duplicate keys allowed in this index (Y or N)?

(See Table 2-3, "ISAM Index Specification Block," for a detailed description of each parameter.) In response to the Key Type question, enter one of the following values:

binary
bytes
character
ShortReal
LongReal
LongIEEE
ShortIEEE
integer

An error message appears if you enter an invalid key type. Be sure to specify consistent values for record size (specified in the first set of questions) and key offset and key size.

After all questions have been answered, the IsamCreate utility requests the server to create the described data set. Any errors are reported at this time.

System Administration

Any user may run IsamCreate to build data sets. If a relative pathname is used to describe the data set being built, the ISAM server builds the file relative to the ISAM administrator's home directory. However, if an absolute pathname is used, the ISAM administrator must have the appropriate permissions to create the file in that path.

The input for IsamCreate may come from a file with standard input redirection. When input comes from a file, the file may contain blank lines and comments that begin with the pound sign (#).

IsamProtect

The IsamProtect utility allows qualified users (Table 5-1 defines a qualified user) to assign and change passwords on ISAM data sets. The ISAM administrator may change the passwords on any ISAM data set, and a qualified user may change the passwords on a data set if the data set has only one UNIX link to it.

To run IsamProtect, type:

IsamProtect datasetname

after the shell prompt on your terminal screen and press the Return key. IsamProtect displays the following sequence of prompts:

Enter password for modification:

Enter new modification password:

Re-enter new modification password:

Enter new read password:

Re-enter new read password:

Note that you have to enter your new password twice. The system asks for verification because your passwords are not echoed as you type them.

At its creation time, a data set does not have a password. Thus, the first time that you specify a password for a data set, just press the Return key after the first prompt. Similarly, if you do not wish to specify either a modification or read password, just press the Return key.

IsamReorg

IsamReorg is a multifunction utility which

- o Rebuilds damaged data sets.
- o Increases space efficiency of data sets.
- o Sorts data sets for more efficient access.
- o Changes the index specifications and loads records from files.

Any qualified user (Table 5-1 defines a qualified user) can run IsamReorg. To run the IsamReorg utility, enter the following command:

```
IsamReorg [options] datasetname
```

after the shell prompt on your terminal screen, and press the Return key. datasetname is the name of the data set, and options specifies a group of options chosen from the following set:

- sn sort the data set on key n, where n is a valid index number for this data set. The sort sequence and direction are based on the index specification block for that index in the data set itself. If the -s option is used more than once in a run of IsamReorg, the first occurrence on the command line is the primary sort key, the second is the secondary sort key, and so on. The use of sorting on a regular basis can significantly reduce access time on data sets, which are typically accessed sequentially based on a single key.
- t print information on the number of records in the file, the block size, and sorting statistics.
- bn set the output block size to n, where n is the number of kilobytes in a block. This option converts files with one blocking factor to another blocking factor for use with the block size option in the IsamConfig file. New blocking factors are required when moving files between installations where the ISAM servers use different blocking factors (see ISAM configuration above). This option is also required if an installation decides to change its blocking factor.

System Administration

- fname load fixed length data records from file name. The size of the records is determined from the data set. The UNIX file is taken to be a set of fixed length records equal in size to those in the data set. If there is a duplicate key error while inserting a record into the file, a warning is printed and that record is ignored.
- o output an index specification to standard output. The -o option causes IsamReorg to output a sequence of lines acceptable as input to IsamCreate. This includes the data set name, record size, number of indices, offset, length, and type of each key field, as well as the direction of sorting and whether or not it includes null keys or allows duplicates. If the output from IsamReorg is redirected into a file, the result can be edited with a UNIX text editor, and then used as standard input to IsamReorg using the -i option (see below) to create new indices or change existing ones. It can also be used as input to IsamCreate. The -o option does not cause the file to be reorganized.
- i input an index specification from standard input. (See the -o option above.) The name and record size should not be changed. If the user attempts to change either, a warning will be printed.
- p do reorganize in place. Normally, IsamReorg creates a temporary work file in /usr/tmp.
- dname dump the records in the ISAM data set into a UNIX file.

When IsamReorg runs, it moves all the data records to the beginning of the file, rebuilds the indices, and recreates the free chains. In addition, IsamReorg performs any specified options.

IsamStop

By issuing the IsamStop command, the ISAM administrator can stop the ISAM server without shutting down the UNIX system. To run IsamStop, type:

IsamStop

after the shell prompt on your terminal screen, and press the Return key. The super user can also use this command.

APPENDIX A: STATUS CODES

Each ISAM procedure returns a status code that reports the status of the procedure to the application. The status code is either 0 (OK) or one of the status codes described below. The names are defined in /usr/include/iserc.h. Some of the status codes have UNIX detail status codes associated with them. (Refer to the AT&T UNIX PC UNIX System V User's Manual for definitions.)

Name	Decimal	Hex	Description
NO_INDEX	3100	0C1C	No such index. An operation that includes a key was invoked, but the specified index does not exist.
INV_PREFIX	3101	0C1D	Prefix not valid. SetUpISAMIterationPrefix was invoked for an index that is neither a byte nor character string.
KEY_LENGTH	3102	0C1E	Bad key length. The length of a key is inconsistent with the type of the index.

Status Codes

<u>Name</u>	<u>Decimal</u>	<u>Hex</u>	<u>Description</u>
HANDLE	3103	0C1F	Bad ISAM handle. The ISAM handle does not identify an open ISAM data set.
HDR_SIZE	3104	0C1E	Bad file header size. The ISAM data set cannot be opened by the OpenISAM operation due to an inconsistency in the header of one of the files of the data set.
BAD_HDR	3105	0C21	Bad ISAM Header.
	3106	0C22	Internal error.
	3108	0C24	Internal error.
	3109	0C25	Internal error.
	3110	0C26	Internal error.
NO_HANDLES	3111	0C27	No free ISAM handles. The ISAM data set cannot be opened because the maximum number of ISAM data sets that can be simultaneously opened by all users combined has been reached.
	3112	0C28	Internal error.
	3114	0C2A	Internal error.
	3116	0C2C	Internal error.
BAD_URI	3117	0C2D	Bad unique record identifier. An incorrect unique record identifier parameter was passed to ISAM.

Status Codes

Name	Decimal	Hex	Description
DUP_KEY	3118	0C2E	Duplicate key. An attempt to store or modify a record would duplicate the key stored in another record for the same index, but the index does not allow duplicates.
INDEX	3119	0C2F	Index file error. This status code is returned as the status code of an ISAM operation. The detail status code refers to a problem with the index file of the ISAM data set.
SECURITY	3120	0C30	Attempted privacy breach. An attempt was made to modify a data set that is open in batch read or transaction read mode.
BAD_REQUEST	3121	0C31	Bad ISAM request. Either the request block is invalid, or the parameters of an ISAM operation are inconsistent or have invalid values.
DATA_STORE	3122	0C32	Data store error.
KEY_INCONSISTENT	3123	0C33	Index to data error.
RECORD_SIZE	3124	0C34	Incorrect record size. The specified record size is incorrect for the ISAM data set.
DUP_ALLOWED	3125	0C35	Duplicates allowed. An operation specified a unique key parameter (that is, duplicates are not allowed), but the index allows duplicates.

Status Codes

<u>Name</u>	<u>Decimal</u>	<u>Hex</u>	<u>Description</u>
NO_RECORD	3126	0C36	No such record. A unique key was used to identify a record, but no record is stored in the ISAM data set with that key.
NO_MORE_RECS	3127	0C37	No more records. A ReadNextISAMRecord(Hold) or GetISAMRecords(Hold) operation has read all the records within the range specified for the current iteration. No record is read when this status code is generated.
BAD_KEY	3128	0C38	Bad key. Either (1) a key field is longer than 64 bytes or is defined to occupy bytes past the end of the record, or (2) a supplied key is invalid; for example, a DECIMAL key with a digit that is not between 0 and 9.
BAD_INDEX	3129	0C39	Bad index. The specified key field does not exist; that is, the iIndex parameter of an ISAM operation is greater than or equal to the number of indices in the ISAM data set.
BAD_MODE	3130	0C3A	Bad ISAM mode. The mode parameter of the OpenISAM operation is invalid.
CANT_OPEN	3131	0C3B	Cannot open ISAM. This is returned as the status code of an OpenISAM operation. The detail status code gives the UNIX error number returned.

Status Codes			
<u>Name</u>	<u>Decimal</u>	<u>Hex</u>	<u>Description</u>
BAD_PASSWORD	3132	0C3C	Bad ISAM password. Either the password does not give the access desired by the OpenISAM operation, or the password is larger than the 8 bytes accepted by the SetISAMProtection operation.
WRONG_SIZE	3133	0C3D	Wrong record size. The OpenISAM operation detects the wrong size record.
MODE_INCOMPAT	3134	0C3E	Incompatible ISAM mode. An attempt was made to open a data set when the data set is already open in an incompatible mode. Detail status code is the mode in which the file is open.
	3135	0C3F	Internal error.
ADMIN_ERR	3136	0C40	Not administrator. An operation for which the data set must be open in administrator mode is attempted with the data set open in some other mode.
CANT_CREATE	3137	0C41	Cannot create ISAM. This is returned as the status code of the CreateISAM operation. Detail status code gives the UNIX error number.
	3139	0C43	Internal error.
SMALL_RECORD	3140	0C44	Small ISAM record. An attempt was made to create an ISAM data set with records shorter than four bytes.

Status Codes

<u>Name</u>	<u>Decimal</u>	<u>Hex</u>	<u>Description</u>
NO_TRANSACT	3141	0C45	Not in transaction. An ISAM operation that can be invoked only when the application is in a transaction was invoked while the application was not in a transaction.
FILE_LOCK	3142	0C46	Data set not available. An attempt to read or hold a record or to hold a data set failed because the data set was held by another user.
REC_LOCK	3143	0C47	Record not available. An attempt to read or hold a record failed because the record was held by another user.
NO_LOCK	3144	0C48	Record not locked. An operation for which the record (or its data set) must be held was invoked when neither the record nor its data set was held.
MANY_LOCKS	3145	0C49	Too many records locked. An attempt was made to hold a record when the maximum allowable number of records were already held.
IN_TRANSACT	3146	0C4A	In transaction. BeginTransaction was invoked during a transaction.
	3147	0C4B	Internal error.
PURGED	3148	0C4C	Transaction purged.
	3149	0C4D	Internal error.
	3150	0C4E	Internal error.

Status Codes

<u>Name</u>	<u>Decimal</u>	<u>Hex</u>	<u>Description</u>
NO_FILE_LOCK	3151	0C4F	Data set not locked. An operation for which the data set must be held was called when the data set was not held.
NO_HEAP	3152	0C50	ISAM heap full. The ISAM operation failed because there is not enough room in the ISAM server's heap to store data structures required by the operation.
	3153	0C51	Internal error.
BAD_MATCH	3155	0C53	Bad match kind. A SetUpISAMIterationLimits operation contained an invalid matchKind argument.
	3156	0C54	Internal error.
BARRIER	3157	0C55	Transaction barrier after modification.
KEY_LOCK	3158	0C56	Key locked -- detail status for 3143.
REC_HELD	3159	0C57	Record held -- detail status for 3143.
ROLLBACK	3160	0C58	Transaction rolled back.
JOURN_ERR	3161	0C59	Journal file error.
JOURN_OPEN	3162	0C5A	Journal open error.
JOURN_CLOSE	3163	0C5B	Journal close error.
JOURN_WRITE	3164	0C5C	Journal write error.
JOURN_ILL	3165	0C5D	Journal record length error.
KEY_TYPE	3200	0C5E	Bad key type.
KEY_SIZE	3201	0C5F	Bad key size.
KEY_ERROR	3202	0C60	Bad key.

Status Codes

<u>Name</u>	<u>Decimal</u>	<u>Hex</u>	<u>Description</u>
ISAM_FILE	3300	0C61	Not an ISAM file.
HDR_CHECKSUM	3301	0CE5	Bad header checksum on file.
REC_EXIST	3302	0CE6	Record does not exist.
	3166- 3199	0C5E- 0C7F	Internal errors.
REC_FORM	3303	0CE7	Malformed record.
			One of the data set's blocks is malformed. Detail status codes are:
		1	Unexpected block number in block header.
		2	Unexpected block size in block header.
		3	Version numbers in block header and trailer don't match.
FILE_TYPE	3305	0CE9	Bad file type.
BUFFER_TYPE	3306	0CEA	Bad buffer size.
SPACE_ERR	3940	0F64	Cannot allocate IPC space.
			The procedure called was unable to allocate space to set up an interprocess communication (IPC) message to send to the ISAM server.
			Detail status code is the UNIX error number.
IPC_ERR	3941	F65	IPC error.
			Unable to communicate with the ISAM server. May mean that the server is not installed.
			Detail status code is the UNIX error number.
MANY_CLIENTS	3942	F66	Too many clients.

Status Codes			
Name	Decimal	Hex	Description
IO_ERROR	3943	F67	I/O error. Error doing an input/output operation to a data set. The detail code gives the UNIX error received. I/O errors should be reported to the ISAM administrator.
BAD_CHAIN	3944	F68	Corrupted free chain. The chain of free records has been corrupted. Run IsamReorg.
	3945- 3999	0F69- 0F9F	Internal errors.

Status Codes

APPENDIX B: PROGRAMMING EXAMPLES

This appendix contains three examples of UNIX PC ISAM applications. The first example creates an ISAM data set, the second stores records in the data set, and the third reads the stored records.

PROGRAM 1: CREATING AN ISAM DATA SET

```
#define ISAMIndCnt=1 /* Set to same value as IsamDesc.Indices */
#include <isam.h>
#include <stdio.h>

main()
{
    struct ISAMStatusBlock StatusBlock;
    struct ISAMDesc IsamDesc;
    unsigned short erc;

    IsamDesc.RecordSize = 40;
    IsamDesc.Indices = 1;
    IsamDesc.IndSpec[0].KeyOffset = 0;
    IsamDesc.IndSpec[0].KeySize = 8;
    IsamDesc.IndSpec[0].Type = 2; /* character */
    IsamDesc.IndSpec[0].fAscending = 1;
    IsamDesc.IndSpec[0].fNullIndexed = 0;
    IsamDesc.IndSpec[0].fDuplicatesAllowed = 0;

    erc= CreateISAM("testfile.isam", 13,
                   0,0,0,0,0,0,
                   &IsamDesc, sizeof(IsamDesc),
                   &StatusBlock);

    if (erc != 0) {
        printf("errors are %d, %d\n",
              StatusBlock.ercDetail);
    } else printf("No errors in CreateISAM\n");
    exit(0);
}
```

Programming Examples

PROGRAM 2: STORING RECORDS

```
1      ' Demonstrate building a data set from given random keys.
2      ' Generate keys in a random order and loops to write each
3      ' record based on that key
4
5      ' Data Set was created using the ISAMCreate facility
6      ' (see 5-7). ISAM functions accessed via Basgen.config
7      ' (see Basic Interpreter Programmer's Guide, section 7).
10     DEFINT A-Z
150    MODE% = 25197          batch modify mode (no locking needed)
200    SRECORD% = 40
260    INPUT "Enter Data set name ",FILENAME$
270    CBDATASETNAME% = LEN( FILENAME$ )
280    RECORD$ = SPACE$(40) ' will hold the record that is read
330    PRINT "Opening Isam File"
390    ERC% = OpenISAM( PTR(HANDLE%), PTR(FILENAME$), CBDATASETNAME%,
0,0, MODE%, SRECORD%, PTR(SBLOCK%))
420    PRINT "errors are.. ", ERC%, (SBLOCK% AND 255),
(SBLOCK%/32768)
425    IF ERC% <> 0 THEN STOP
430
431    INPUT "Enter number of records you want",N
435    DIM S$(N)
436    X = 77
440    FOR I% = 1 TO N
445        X = (X+337) MOD N
450        RECORD$ = "x"+STR$(X*2)+"x"+STR$(X*2)+"x"+STR$(X*2)+"data
data data ..."
455        S$(I%) = RECORD$
460    NEXT I%
470    PRINT "Looping to store ";n;" ISAM records"
480    FOR I% = 1 TO N
490        RECORD$ = S$(I%)
500        ERC% = StoreISAMRecord(HANDLE%, PTR(RECORD%), SRECORD%,
PTR(URI), PTR(SBLOCK%))
520        IF ERC% <> 0 THEN PRINT "ERC on Store:"; ERC% : GOTO 570
550    NEXT I%
560
570    PRINT "closing ISAM"
580    ERC% = CloseISAM( HANDLE%, PTR(SBLOCK%))
590    PRINT "CloseISAM errors.. ", ERC%, (SBLOCK% AND 255)
600    END
```

Programming Examples

PROGRAM 3: READING RECORDS

```

1  ' Does sequential read of an existing ISAM data set
2  '
20  DEFINIT A-Z
30  KEY$ = SPACE$(8)
40  SKEY = 8
50  RECORD$ = SPACE$(40)
60  MODE% = 25197 ' batch modify
70  SRECORD% = 40
80  INPUT "Enter data set name",FILENAME$ 90  CBDATASETNAME% = LEN(
FILENAME$ )
90  CBDATASETNAME% = LINE( FILENAME$ )
110  PRINT "Opening Isam File"
120  ERC% = OpenISAM( PTR(HANDLE%), PTR(FILENAME$), CBDATASETNAME%,
0,0, MODE%, SRECORD%, PTR(SBLOCK%))
130  PRINT "errors are.. ", ERC%, (SBLOCK% AND 255) : IF ERC% <> 0
GOTO 350
140  INPUT "Enter number of iterations",N
150  ERC% = SetUpPrefixIteration( HANDLE%, 0, PTR(STARTKEY%),
LEN(STARTKEY%), PTR(SBLOCK%))
160  IF ERC% <> 0 THEN PRINT "SetUp error:";ERC% : GOTO 350
240  FOR I% = 1 TO N
260      ERC% = RdNextISAMRecord( HANDLE%, PTR(RECORD$), SRECORD%,
PTR(URI), PTR(SBLOCK%))
290      IF ERC% <> 0 GOTO 350
330  NEXT I%
350  ERC% = CloseISAM( HANDLE%, PTR(SBLOCK%))
360  PRINT "CloseISAM errors.. ", ERC%, (SBLOCK% AND 255)
370  END

```

Programming Examples

Index

A

- absolute pathname 5-8
 - creation of file in 5-8
- access
 - data set 2-12, 4-1
 - direct 2-5, 2-6
 - efficient, sorting for 5-9
 - exclusive 1-1, 2-13, 2-14
 - existing 2-13
 - multiuser 2-14
 - random 1-1
 - routine 1-1
 - sequential 2-5, 2-6
 - shared 1-1, 2-14
 - simultaneous 2-14
- access modes
 - administrator 2-12, 4-24
 - batch 2-12, 4-24
 - transaction 2-12, 4-24
- accessing directories 4-2
- administrator
 - activities performed 2-12
 - mode (MD_ADMIN) 3-8, 4-24
 - opening data set in 4-24
- anomalies 2-23
- ascending sort order 2-2
- ASCII
 - lowercase characters 2-2, 2-3
 - modes in 4-24
 - uppercase characters 2-2, 2-3
- ASCII character set, passwords from 4-35
- ASCII formatted text file
 - maintenance of 5-2 to 5-3
- automatic compression 2-6
- avoiding inconsistency 4-20, 4-22, 4-32

Index

B

- B-tree structure 2-6, 3-15
 - advantages of 2-6
- BASIC 1-1
 - Interpreter 2-10
 - representations in 2-10
- batch mode
 - exclusive access 2-13, 2-14
 - read 2-13
- Batch modify 2-13, 3-8
 - (MD_BMOD) 4-24
- Batch read 3-8, 4-24
- BeginTransaction 2-16, 2-17, 4-1, 4-4
- binary
 - bytes 2-2
 - fixed-length string 2-2, 2-3
 - key type 2-2
 - zeros 4-37, 4-40
- Binary(0) 2-3
- block
 - size 5-3
- block splitting 2-6
- boot procedure 5-2
- buffer
 - capacity 4-12
 - server 5-3
 - structure 4-13
- Buffer Structure for GetISAMRecords and GetHoldISAMRecords When Buffer Structure for GetISAMRecords and GetHoldISAMRecords When Records Are Read (46-Byte Records) 4-14
- byte 2-2
 - low address 2-3, 2-4
 - string key 4-40
- Byte(1) 2-3

C

- C 1-1, 2-10
 - examples written in 3-1
 - format header files 4-2
- cache
 - default 5-3
 - internal 5-3
- calls to server 4-2
- cbDataSetName 4-23
- cbDataStoreFile 4-34

C (continued)

- cbDSFile 4-7
- cbIndexFile 4-34
- cbIXFile 4-7
- cbNode 4-42
- cbPassword 4-23
- cbPWDataStoreFileRename 4-34
- cbPWFileCreate 4-7
- cbPWIndexFileRename 4-34
- cbPWOpnModify 4-35
- cbPWOpnRead 4-35
- cbWIXCreate 4-7
- changes, guarantee of 4-6
- character
 - data 2-2
 - string(s) 2-2
 - string(s) contents of 2-3
 - string(s) key 2-3, 4-40
 - string(s) programming languages and 2-3, 2-4
 - string(s) representation in programming languages 2-3
 - string(s) sorting 2-3
- character data 1-1
- Character(2) 2-3
- cIndexes 4-2
- CloseISAM 2-17, 4-1, 4-5, 5-5
- COBOL
- CommitTransaction 2-16, 2-17, 4-1, 4-4, 4-6
- compatibility 1-1
- compile, using library to 4-2
- composite keys 2-1, 2-2
- compound indices 3-4, 3-14
- concurrent clients, serving 5-4
- configuration file 5-2
 - parameters 5-2 to 5-4
- conflicts and deadlocks 2-20
 - resolving 2-22
- consistency checking 5-3
- constraints, transaction-related 2-16
- conventions for parameter names 4-2
- create new ISAM data set 4-7
- CreateISAM 2-17, 3-2, 3-4 to 3-7, 4-1, 4-2, 4-7
- creating new record 4-41
- creation of data set 2-1

Index

D

data set

- accessing 2-12, 2-13
 - attributes 2-6
 - changing 2-14, 2-18
 - changing name 4-34
 - changing password 4-35
 - close 4-5
 - consistent state 2-18
 - controlling access 2-12
 - creating multiple 3-6
 - creating new 4-7
 - creating simple 3-1
 - creation 2-1
 - deleting 2-12, 4-10
 - description 2-7
 - empty 4-7
 - exclusive access 2-13, 2-14
 - inconsistent state 2-18
 - increasing length 4-41
 - increasing space efficiency 5-9
 - locking 1-1, 2-16, 2-18, 2-20, 4-15
 - management 4-1
 - modifying 1-1, 2-13, 2-23
 - opening existing 3-7, 4-23
 - problems 2-23
 - reading 2-13
 - rebuilding damaged 5-9
 - reconstructing 2-23
 - rejection of by server 5-3
 - releasing (unlocking) 2-20, 4-32
 - renaming 2-12
 - restraints controlling access 2-13
 - security 5-2
 - setting protection 2-12
 - sharing 2-12, 2-13, 2-14
 - sort for efficient access 5-9
 - storage 2-6
 - unavailable 2-22
 - UNIX 1-1
 - unlocking (releasing) 2-20, 4-32
- Data Set Management 4-1
- data store section 2-6
- deadlocks 2-20, 2-21
- avoiding 2-22
- decimal
- packed 2-2

D (continued)

- default
 - directory 5-1, 5-2, 5-8
 - timeout value 2-22
- DeleteByKeyISAMRecord 2-17, 4-1, 4-8, see also
DelISAMRecord
- DeleteISAM 2-17, 4-1, 4-10
- deleting 2-12
- DelISAMRecord 2-17, 4-1, 4-11
- descending sort order 2-2
- destruction of data 4-8, 4-10
- determining sort order 2-2
- direct access 2-5, 2-6
- directory
 - and ISAM utilities 5-1
 - creating files for 4-2
 - home 2-23, 5-1, 5-2, 5-8
- disk
 - allocation policies 2-7
 - freeing space 4-8
 - management 2-6
 - reducing usage 5-4
- dump records into UNIX file 5-10
- duplicate key values 2-1
 - allowance of 2-1, 2-5, 4-41
 - presence of when not allowed 4-19, 4-21, 4-28, 4-41
- duplicate keys 4-19, 4-21
- DUP_KEY 4-19, 4-21, 4-41

E

- empty data set 4-7
- ercDetail 3-3, 4-3
- ercISAM 3-3, 4-3
- error message 5-7
- ErrorLog 2-23, 5-1
- errors
 - description 4-17
 - hardware 2-23
 - in ISAMConfig file 5-3
 - list of A-1 to A-9
 - log of hard 5-1
 - logging 2-23
 - output to standard error stream 4-17
 - prevention 2-14
 - report 5-7
 - return codes 4-2
 - software 2-23

Index

E (continued)

/etc/rc 5-1
exclusive access 1-1, 2-13, 2-14
execute permission 4-2
existing access 2-13
exit from programs 5-6
explicit pathname 5-1
external event, deadlocks and 2-22

F

fAscending 2-9, 3-3, 3-7, 3-15, 4-2
fDuplicates 2-9, 3-3, 3-7, 3-15
fields 2-1, 2-2
file descriptors, UNIX 5-3
file locations 4-2
files 2-6, 5-1, 5-2, 5-9
Files, Errorlog 2-23
fixed-length records 1-1
fLockRecords 4-15
fNullIndexed 2-9, 3-7, 3-15
FORTRAN-77
 representations in 2-10
fReadRecords 4-13
free chains 5-10
free records 2-6
freeing space see deleting

G

GetHoldISAMRecords 2-17, 4-1, 4-12, 4-37, 4-40
GetISAMRecords 2-17, 3-12, 4-1, 4-12, 4-37, 4-40
GrowData 2-8
GrowIndex 2-8, 3-3

H

hardware errors 2-23
hardware failure 2-23
header 2-6
 files, C format 4-2
HoldISAMDataSet 2-17, 4-1, 4-15
home directory, administrator's 2-23, 5-1, 5-2, 5-8
 adding files in 4-2

I, J

- identical records 2-1
- identifying records 1-1
- iIndex 4-8, 4-20, 4-37, 4-40
- inconsistent data, avoidance of 4-32
- index
 - automatic updates 2-5, 4-19, 4-21, 4-41
 - building structure 2-5
 - defining for data set 4-7
 - self-reorganization 2-6
 - sorting with composite keys 2-1 to 2-2
 - sorting with simple keys 2-1
 - specifications 5-9, 5-10
- index section 2-6
- IndexSpec 2-8, 3-2, 3-3, 3-7
- Indices 2-8, 3-3
 - implementation 2-6
 - multiple 3-6
 - rebuild 5-10
 - using compound 3-1, 3-14
- InitDataSize 2-8
- InitIndexSize 2-8, 3-3
- insrc.h 4-2
- installation block size 5-3
- integer 2-2
 - signed 2-4
 - unsigned 2-2
- Integer (7) 2-3
- integer key, length in bytes 2-4
- internal consistency checking 2-23
- invalid key type 5-7
- IO_ERROR 5-1
- ISAM
 - data sets 1-1
 - establishing availability of 4-42
 - handle 2-11
 - library 4-2
 - maintenance, utilities for 5-1
 - procedures 2-11, 2-12, 4-1 to 4-43
 - reference in subsequent procedures 4-23
 - server 1-1, 5-1 to 5-3, 5-10
 - service access 4-1
 - utilities 1-1
 - validity 2-11
- ISAM administration 1-1

Index

I, J (continued)

ISAM administrator
 assigning, changing passwords 5-8
 data security 2-23
 data sets belonging to 4-2
 file permissions 4-2
 home directory 2-23, 4-2, 5-1, 5-2, 5-8
 mode 2-12
 permission 5-2, 5-8
 server and 5-1, 5-2
ISAM Description Block 2-7, 2-8, 3-2
ISAM Index Specification Block 2-9
ISAM Key Types and Programming Language
 Representations 2-10
ISAM Procedures by Function 4-1
ISAM Service Access 4-1
ISAM Status Block 3-2 to 3-5
ISAM Utility Programs 5-6
isam.h 3-3, 3-8, 4-2, 4-3, 4-37
isam.h name 2-3
IsamCreate 1-1, 2-23, 3-6, 5-6, 5-7, 5-10
ISAMDesc 3-2, 3-5, 3-7
ISAMHandle 2-11, 3-7, 3-9, 4-5, see also specific
 procedures
ISAMHoldRecord 2-17, 4-1, 4-18
IsamProtect 1-1, 2-23, 5-6, 5-8
IsamReorg 1-1, 2-23, 5-3, 5-6, 5-9
ISAMStatusBlock 3-8
IsamStop 1-1, 2-23, 5-6, 5-10
iserror 4-17

K

key component, type and size 2-9
key description 2-7
key fields 1-1, 2-5
key types 1-1, 2-2 to 2-4, 5-7
 UNIX PC ISAM 2-3
key values 2-1, 2-5
KeyDescriptor 4-3
KeyOffset 2-9, 3-3, 3-7, 3-15
keys 1-1, 2-1, 2-2, 2-4, 2-6, 2-9, 4-37, 4-40
KeySize 2-9, 3-3, 3-7, 3-15

L

languages, programming 1-1, 2-2 to 2-4
libisam.a 5-1
library 1-1, 4-2
 functions 5-1
list of errors A-1 to A-9
loading from UNIX files 5-10
locking
 open data set 4-15
 procedures 2-16, 2-18 to 2-20, 2-20
log of hard errors 5-1
login 5-2
 set up 5-1
long real keys 2-4, 2-10
Long/Short IEEE 2-4
LongIEEE (8) 2-3
LongReal (4) 2-3
low-address byte 2-2
 significance 2-3, 2-4

M

maintaining data sets 2-23
maintenance, ISAM utilities 1-1, 5-1
matchKind 4-37, 4-38
MatchKind Words for Record Retrieval 4-38
Modes 4-24
modification 2-14, 2-16, 2-18
modify password 2-23
ModifyByKeyISAMRecord 2-17, 4-1, 4-19
modifying 2-12
 data sets 2-23
 records 2-16, 2-18, 4-19
 using URI 4-21
ModifyISAMRecord 2-17, 4-1, 4-21
Motorola 68000 1-1
multiple fields 2-1
multiple record(s)
 access iteration 4-1
 reading 3-12
multiple updates, simultaneous 2-14
multiuser ISAM 4-42

Index

N

- natural order 2-1
- NO_MORE_RECS 4-12
- null keys 4-37, 4-40
- null value keys 2-1
- null values 4-37, 4-40
- numeric data 1-1
 - key types 2-2
 - programming languages 2-4

O

- offset 2-1, 2-8, 2-9
- open data sets
 - locking 4-15
 - number of 5-4
- open file retention 5-4
- opening data sets 2-13
- OpenISAM 2-11, 2-17, 3-7, 4-1, 4-2, 4-23
- order of
 - creation 2-1,
 - records 2-1
- order of significance 2-2
- order, natural 2-1
- overflow chains 2-6
- ownership of files 5-1

P

- packed decimal 2-2
- packing 4-13
- pad, padding 2-9, 4-13
- parameter name conventions 4-2, 4-25, 4-36, 5-1
 - setting 4-36
- parameters for keys 2-1
- Pascal 2-10
- passwords 4-23, 4-35, 5-2, 5-6, 5-8
- pathname 4-2, 5-1, 5-8
- pbDataSetName 4-23
- pbDataStoreFile 4-34
- pbDSFile 4-7
- pbIndexFile 4-34
- pbIXFile 4-7
- pbNode 4-42
- pbPassword 4-23
- pbPWDataStoreFileRename 4-34
- pbPWFileCreate 4-7

P (continued)

pbPWIndexFileRename 4-34
pbPWIXCreate 4-7
pbPWOpnModify 4-35
pbPWOpnRead 4-35
pBuffer 4-13
pCRecordsReadRet 4-14
PFILE 5-3
pISAMDesc 4-7
pISAMHandleRet 4-23
pKey 4-8, 4-29, 4-40
pKey1 4-39
pKeyDescriptor 4-3
position in record 2-1
pParams 4-36
pParamsRet 4-25
pRecord 4-20, 4-21, 4-41
pRecord Ret 4-26, 4-29, 4-30
prefix definitions 4-2 to 4-3
Procedures 2-11 to 2-13, 4-1 to 4-43
 BeginTransaction 2-17, 4-1, 4-4
 CloseISAM 2-17, 4-1, 4-5, 5-5
 CommitTransaction 2-17, 4-1, 4-4, 4-6
 CreateISAM 2-17, 4-1, 4-2, 4-7
 DeleteByKeyISAMRecord 2-17, 4-1, 4-8
 DeleteISAM 2-17, 4-1, 4-10
 DelISAMRecord 2-17, 4-1, 4-11
 GetHoldISAMRecords 2-17, 4-1, 4-12, 4-37, 4-40
 GetISAMRecords 2-17, 4-1, 4-12, 4-37, 4-40
 HoldISAMDataSet 2-17, 4-1, 4-15
 ISAMHoldRecord 2-17, 4-1, 4-18
 Iserror 4-17
 ModifyByKeyISAMRecord 2-17, 4-1, 4-19
 ModifyISAMRecord 2-17, 4-1, 4-21
 OpenISAM 2-11, 2-17, 4-1, 4-2, 4-23
 QueryTransactionParams 2-17, 2-22, 4-1, 4-25
 RdHoldNextISAMRecord 2-17, 4-1, 4-26, 4-37, 4-40
 RdHoldUniqueISAMRecord 2-17, 4-1, 4-28
 RdNextISAMRecord 2-17, 4-1, 4-26, 4-37, 4-40
 RdUniqueISAMRecord 2-17, 4-1, 4-28
 ReadHoldISAMRecordByUri 2-17, 4-1, 4-30
 ReadISAMRecordByUri 2-17, 4-1, 4-30
 ReleaseDataSet 2-17, 4-1, 4-32
 ReleaseISAMRecord 2-17, 4-1, 4-33
 RenameISAM 2-17, 4-1, 4-34
 SetISAMProtection 2-17, 4-1, 4-35
 SetTransactionParams 2-17, 2-22, 4-1, 4-36
 SetUpLimitsISAMIteration 2-17, 4-1, 4-26, 4-37
 SetUpPrefixIteration 2-17, 4-1, 4-26, 4-40

Index

P (continued)

Procedures (continued)

- StoreISAMRecord 2-17, 4-1, 4-41
- VerifyMultiuserISAM 2-17, 4-1, 4-42
- Programming Examples B-1 to B-3
- programming languages 1-1
- pSParamsRet 4-25
- pStatusBlockRet parameter 4-3, see also specific procedures
- pUriRecordRet 4-8, 4-20, 4-26, 4-29, 4-41

Q

- qualified user 5-6, 5-9
 - assigning, changing passwords for 5-8
- QueryTransactionParams 2-17, 2-22, 4-1, 4-25
- queue, queue request 2-20, 2-22

R

- random access 1-1
- range 4-12, 4-26
- RdHoldNextISAMRecord 2-17, 4-1, 4-26, 4-37, 4-40
- RdHoldUniqueISAMRecord 2-17, 4-1, 4-28
- RdNextISAMRecord 2-17, 4-1, 4-26, 4-37, 4-40
- RdUniqueISAMRecord 2-17, 4-1, 4-28
- read
 - initialize sequence of 4-37, 4-40
 - operations 2-12
 - password 2-23
 - permission 5-2
 - procedures 2-12
- read-only access 2-13
- ReadHoldISAMRecordByUri 2-17, 4-1, 4-30
- ReadHoldUniqueISAMRecord 2-17
- reading 2-11, 2-12
 - multiple records 3-12
 - records 2-5, 2-11
 - single records 3-10
- ReadISAMRecordByUri 2-17, 3-10, 4-1, 4-30
- ReadNextISAMRecord 3-12
- ReadUniqueISAMRecord 3-10 to 3-12
- real keys 2-4
- rebuild indices 5-10
 - locking 1-1

R (continued)**Record**

- Locking 4-1
- Management 4-1

- record key 2-1

records

- constraints on returning 4-12, 4-13
- creating 4-41
- deleting 2-6, 2-12, 4-9
- deleting by its URI 4-11
- deleting by key 4-8
- deleting data set 4-10
- fixed length 1-1
- fixed-length 4-24
- identical 2-1
- identifier, validity 4-18
- identifying 1-1, 2-12
- length 2-1
- loading from file 5-9
- location 2-6
- locking 2-16, 2-18, 2-20, 4-1
- locking by URI 4-18
- management 4-1
- modification 2-5, 2-12
- modify by unique key 4-19
- modify by URI 4-21
- number of keys 2-1
- read by unique key 4-28
- read by URI 4-30
- reading in key order 2-5
- reading key order, several in 4-12
- reading multiple 3-12
- reading single 2-5, 3-10
- releasing records 4-12, 4-33
- retrieval, efficient 3-14
- retrieving 2-5
- retrieving records 4-37, 4-38
- size 2-1, 2-7, 2-8
- storage 2-5
- storage new 3-9
- storage new (example) 3-1
- storing new 2-11
- unavailability 2-22
- unlocking 4-33

- RecordSize 2-8

- REC_FORM 5-1

- relative pathname 5-8

- ReleaseDataSet 2-17, 4-1, 4-32

- ReleaseISAMRecord 2-17, 4-1, 4-33

Index

R (continued)

RenameISAM 2-17, 4-1, 4-34
reorganize indices 5-10
request to open data set, denial 2-13, 2-14
Reserved 2-8
Restrictions on Multiple Access to a Data Set 2-13
retrieval 2-11, 2-12, 3-14
routines 1-1
Rubout Key 5-6
run time 5-1

S

sBuffer 4-13
Sector Node 3-3
security 5-1, 5-2
sequential access 2-1, 2-5, 2-6
server 5-1 to 5-3, 5-10
service requests 5-1
SetISAMProtection 2-17, 4-1, 4-35
SetLimitsISAMIteration 3-12
setting protection 2-12
SetTransactionParams 2-17, 2-22, 4-1, 4-36
SetUpLimitsISAMIteration 2-17, 4-1, 4-26, 4-37
SetUpPrefixIteration 2-17, 3-12, 4-1, 4-26, 4-40
shared access 1-1, 2-14
ShortIEEE (9) 2-3
ShortReal (5) 2-3
signed integer 2-4
simple keys, sorting index 2-1
simultaneous access 2-14
Single Record Access 4-1
single records, reading 3-10
sISAMDesc 4-7
size defaults 2-8
sKey 4-8, 4-28, 4-40
sKey1 4-39
sKey2 4-39
software errors or failure 2-23
sort order 2-2, 2-4
sParams 4-36
sParamsMax 4-25
sRecord 3-8, 4-20, 4-21, 4-24, 4-26, 4-29, 4-30, 4-41
standard error stream 4-17
standard input redirection 5-8
startup scripts, UNIX 5-1

S (continued)

Status Block Format (pStatusBlockRet Parameter) 4-3
status block, errors 4-3
Status Codes 4-3, A-1 to A-9
StoreISAMRecord 2-17, 3-9, 4-1, 4-41
storing 2-11
 new records 2-11, 3-9
 new records (example) 3-1
super user 5-6, 5-10

T

text editor, UNIX 5-2
time-out(s) 2-22, 4-25
 value 2-22
Transaction 1-1, 2-14, 2-15, 2-16, 4-16
 Begin 4-4
 parameters retrieving current 4-25
 setting current parameters 4-36
 successful completion 4-6
transaction mode 2-13
Transaction modify 2-13, 3-8, 4-24
Transaction Parameters Block Format 2-22
Transaction read 2-13, 3-8, 4-24
transaction-related constraints 2-16
 controlling operations 2-22
Type 2-9, 3-3, 3-7, 3-15
TY_BINARY 2-3
TY_BYTES 2-3
TY_CHAR 2-3
TY_INTEGER 2-3
TY_LIEEE 2-3
TY_LREAL 2-3
TY_SIEEE 2-3

U

unavailability of ISAM 4-42
uninterpreted fixed-length string 2-2
unique key(s) 2-5
 identifying record 4-8
 locking record 4-28
 modifying 4-19
 reading record 4-28
Unique Record Identifier (URI) 2-11

Index

U (continued)

UNIX

- administrator mode 2-12
- data sets 1-1
- file descriptors 5-3
- move utility (mv) 4-34
- remove utility (rm) 5-2
- startup script /etc/rc 5-1
- text editor 5-2
- utilities 2-23, 5-1, 5-6
 - programming languages 1-1
- unlocking
 - data sets 4-6
 - records 4-6
- updating indexes 4-19, 4-41
- uriRecord 4-11, 4-18, 4-21, 4-30
- URIs 2-12
- user 2-1
 - handles 5-3
 - qualified 5-6, 5-9
 - super 5-6, 5-10
- /user/lib (libisam.a) 4-2
- utilities, UNIX PC ISAM 1-1, 2-23, 4-2, 5-1, 5-6
 - ISAMReorg 5-9

V

- Verifying ISAM availability 4-42
- VerifyMultiUserISAM 2-17, 4-1, 4-42

W, X, Y, Z

Warnings

- DeleteByKeyISAMRecord 4-9
- HoldISAMDataSet 4-16
- ModifyByKeyISAMRecord 4-20
- ModifyISAMRecord 4-22
- OpenISAM 4-24
- ReleaseDataSet 4-32
- ReleaseISAMRecord 4-33
- SetISAMProtection 4-35
- write permission 5-2
- WriteThru 5-5
- writing to disk 5-4
- wTicksWait 2-22