Burroughs

Reference Manual

B 20 Systems

Indexed Sequential Access Method (ISAM)

(Relative to Release Level 4.0)

Priced Item
Printed in U.S.A.
July 1984

1168465

Burroughs

Reference Manual

B 20 Systems Indexed Sequential Access Method (ISAM)

(Relative to Release Level 4.0)

Priced Item
Printed in U.S.A.
July 1984

1168465

Burroughs cannot accept any financial or other responsibilities that may be the result of your use of this information or software material, including direct, indirect, special or consequential damages. There are no warranties extended or granted by this document or software material.

You should be very careful to ensure that the use of this software material and/or information complies with the laws, rules, and regulations of the jurisdictions with respect to which it is used.

The information contained herein is subject to change without notice. Revisions may be issued to advise of such changes and/or additions.

Correspondence regarding this publication should be forwarded using the Remarks form at the back of the manual, or may be addressed directly to Corporate Documentation-West, Burroughs Corporation, 1300 John Reed Court, City of Industry, California 91745, U.S.A.

LIST OF EFFECTIVE PAGES

Page	Issue
Title	Original
ii	Original
iii	Original
iv	Blank
v thru ix	Original
x	Blank
xi	Original
xii	Blank
1-1 thru 1-2	Original
2-1 thru 2-17	Original
2-18	Blank
3-1 thru 3-79	Original
3-80	Blank
4-1 thru 4-17	Original
4-18	Blank
5-1 thru 5-13	Original
5-14	Blank
6-1 thru 6-16	Original
A-1 thru A-15	Original
A-16	Blank
B-1 thru B-4	Original
C-1 thru C-2	Original
D-1 thru D-3	Original
D-4	Blank
E-1 thru E-4	Original
1 thru 5	Original
6	Blank

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont)

TABLE OF CONTENTS (Cont)

TABLE OF CONTENTS (Cont)

APPENDIXES

LIST OF ILLUSTRATIONS

LIST OF TABLES

INTRODUCTION

This manual provides descriptive and operational information for the Indexed Sequential Access Method (ISAM) data management facility that B 20 microcomputer systems use. ISAM provides efficient and flexible random access to data identified by multiple keys. The information is presented as follows:

Section 1: Overview
Section 2: Concepts
Section 3: ISAM Operations: Procedures and Services
Section 4: ISAM Utilities
Section 5: ISAM Reorganization
Section 6: ISAM Server Installation
Appendix A: Status Codes
Appendix B: Upward Compatibility Support
Appendix C: Software Requirements
Appendix D: Estimating Index Files Sizes
Appendix E: Glossary

An index follows the appendixes.

The following manuals are referenced for additional information:

B 20 Systems Operating System (BTOS) Reference Manual

B 20 Systems System Programmer's Guide, Part 1

B 20 Systems Linker/Librarian Reference Manual

B 20 Systems Sort/Merge Reference Manual

SECTION 1
OVERVIEW

The Burroughs Indexed Sequential Access Method (ISAM) is a software product that provides efficient, flexible random access to fixed-length records that are identified by keys contained in the records. You can use Assembler, BASIC, COBOL, FORTRAN, and PASCAL to write application systems that access ISAM.

ISAM is supported on a standalone workstation, a cluster workstation, and a master workstation.

FEATURES

ISAM includes the following features:

- random access to data identified by multiple keys
- up to 100 keys per ISAM data set
- key types to support many different representations of character and numeric data
- shared or exclusive access to ISAM data sets, with transactions and record- or data set-level locking
- an ISAM server installed in memory at the master workstation of a cluster or at a standalone workstation, or loaded as a task of a BTOS application system
- utilities to perform maintenance and modifications to data sets and to provide status reports for data sets

USING THIS MANUAL

This manual is intended for system designers and applications programmers writing ISAM applications to be used under BTOS on Burroughs B 20 workstations.

Section 2 provides a high-level view of the features and functions of ISAM. Section 3 describes the ISAM operations by functional categories and also presents all the operations in detail alphabetically. Sections 4 and 5 document the ISAM utilities for data set maintenance. Section 6 explains how to install and configure the ISAM server.

Additional information such as status codes and software and hardware requirements is included in the appendixes.

EXAMPLES IN THIS MANUAL

This manual uses Personnel data sets for examples and illustrations. This approach offers data elements that are familiar to most users and provides continuity as the different functional levels of ISAM are presented.

The Personnel information is contained in three data sets: an Employee data set, a Department data set, and a Dependent data set. The fields included in each data set are:

Employee data set, with one record per employee

- department number
- employee number
- employee name
- salary

Department data set, with one record per employee

- department number
- department name
- employee number of department manager

Dependent data set, with one record per dependent

- employee number
- number of dependents
- dependent name
- dependent date of birth

SECTION 2
CONCEPTS

ISAM supports access to fixed-length data records contained in ISAM data sets. Each ISAM data set holds one type of data record. When you create an ISAM data set, you specify the record length and key fields. You access the records of a data set through fixed-length keys. You can define multiple key fields to support different access patterns.

FILE TYPES

Each ISAM data set is stored as two physical files: a data store file and an index file. You can place these two files on different physical volumes.

The data store file holds the data records. Because all the records in a data set have the same length, disk space management is simple and efficient. Whenever you delete a record, the system marks it as deleted and adds it to a list of free records to be reused later when you create a new record. The data store file is a Direct Access Method (DAM) file.

The name of a data set is the name of its data store file. You must supply a file specification for the data store file to access or create a data set.

The index file holds the indexes for all of a data set's keys. Indexes 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:

- direct support of both direct (by key) and sequential access
- 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

- very fast sequential access: two or three reads to locate and read the first record and usually a single read to read each subsequent record

- self-reorganization: automatic expansion to accommodate new keys

- automatic compression when keys are removed to improve disk utilization and efficiency of access

The separation of data and indexes is structured to allow the use of Record Sequential Access Method (RSAM) and DAM for read-only access to ISAM data set files. (For further information on RSAM, see the *B 20 Systems Operating System (BTOS) Reference Manual.* ISAM maintains logical dependencies between the data store file and index file of a data set, so neither file can be opened for modification by any access method other than ISAM. ISAM includes a consistency check to detect whether an invalid access has occurred.

The use of separate data store and index files also enables the File Maintainer utility to recover data from the data store file after hardware or software failures.

KEY TYPES

A record can have up to 100 keys. Each key is described by its position in the record, that is: the offset from the first byte of the record; the key length; and the key type. The ISAM key types support the various character and numeric representations used by the Burroughs programming languages and processors. Byte string and character string key types support character data. Numeric key types support integer, binary, packed decimal, display, and several forms of real numbers.

To increase flexibility, you can specify the following parameters for each key when you create an ISAM data set:

- whether duplicates are allowed

- whether the index is to be kept in ascending or descending order

- whether indexing of null value keys is to be suppressed to reduce the size of the index

One index is used for record retrieval for each key. Records can be retrieved in key-order sequence by any key field, starting with any key value. The index is automatically updated when records are stored or modified.

UNIQUE RECORD IDENTIFIER

A 4-byte unsigned integer uniquely identifies each record in a data set. This integer is called the unique record identifier (URI). Store and Read operations return the unique record identifiers of the records that have been stored or read. Modify and Delete operations use the unique record identifier to identify the record to be processed.

Unique record identifiers are valid only while the data set is open. You cannot save them to identify records once you have closed and reopened the data set.

ISAM OPERATIONS

ISAM supports four main types of operations: storing, reading, modifying, and deleting.

Storing

When an application system stores a new record, ISAM inserts the record into the data set and then automatically indexes the record according to the values in all the record's key fields.

Reading

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

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

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

The retrieval result can be either the specified records or a sequence of 4-byte unique record identifiers. If unique record identifiers are retrieved, then the application program can later obtain the corresponding records by using a special form of the Read operation that does not reaccess the index.

Modifying

When an application system modifies an existing record, ISAM automatically removes it from each index for which the key field is being changed, and then indexes it under the new key.

Deleting

When an application program deletes an existing record, ISAM removes the record from the data set and from each index.

DATA SET ACCESS MODES

You can use one of three modes to open a data set in ISAM. You use administrator mode when you are doing data set-level activities such as deleting, renaming, and setting protection. You use batch mode to open a data set in applications that require exclusive use of the data set. Opening a data set in transaction mode allows other applications to access and modify the data set concurrently.

Data sets opened in batch or transaction mode are opened for either read or modify. Whether read or modify is used affects the extent to which the application systems can share the data set.

A data set opened in batch read mode can be shared by other users who open it for read-only access. A request to open the data set in administrator, batch modify, or transaction modify mode is denied if any user opens the data set in batch read mode. A request to open the data set in batch read mode cannot be executed if the data set has already been opened in administrator, batch modify, or transaction modify mode.

A data set opened in batch modify mode is opened exclusively. No other user can open it. A request to open the data set in batch modify mode cannot be executed if another user has already opened it in any mode.

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

A data set opened 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 be executed if another user has the data set open in administrator, batch read, or batch modify mode.

A data set opened in administrator mode is opened exclusively. A request to open the data set in administrator mode cannot be executed if another user has already opened it in any mode.

These rules are summarized in table 2-1.

Table 2-1. Data Set Access Modes

Initial Mode	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

TRANSACTIONS

An application system can open a data set in batch mode to read or modify records. In batch mode, an application system has exclusive access to the data set. Since other application systems do not have access to the data set at the same time, it is not possible for multiple updates to be made simultaneously. Modification of the records and data sets does not affect any other application system. In transaction mode, when data sets are open for shared access, however, many application systems can access the same records and data sets. Any modification of a record or data set by one application system affects the other application systems. Errors can arise if simultaneous access to a data set is not coordinated. In ISAM, transactions are the coordination mechanism.

In transaction mode, application systems 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 2-1. Changes to a data set must be made within a transaction. Only after a transaction is completed can other application systems access the data that has been changed.

The beginning of a transaction is the Begin-Transaction operation; the end of a transaction is either a CommitTransaction or a RollBack-Transaction operation.

While some operations can be performed whether or not the application system is in a transaction, any operation that locks or modifies a record (or locks a data set) must be performed while the application system is in a transaction. See table 2-2.

Locking a Record or Data Set

During a transaction, an application system can make a number of changes to a data set. When an application system is in the midst of modifying data, the data set is not in a consistent state. Other application systems must be prevented from accessing the changed data. Application systems prevent other applications from accessing inconsistent data by using transaction modify mode and locking records and data sets.

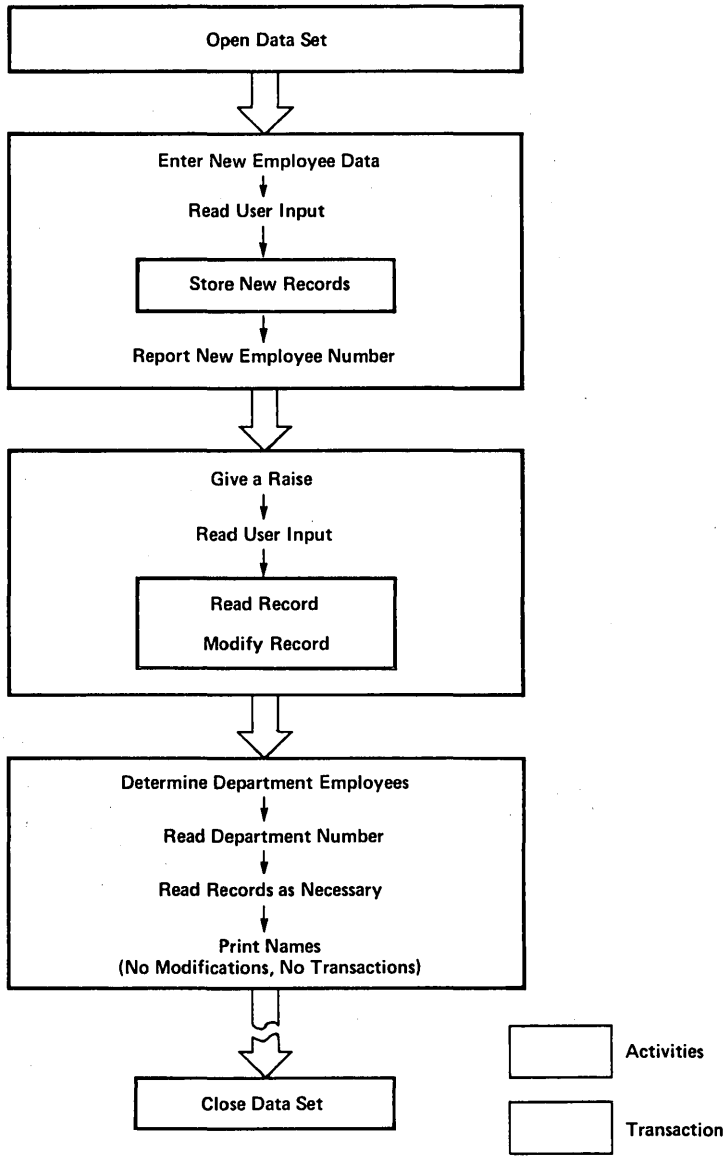

Figure 2-1. Series of Transactions

Table 2-2. Operations and Transactions

Operations Allowed Only During a Transaction	Operations Allowed At Any Time
CommitTransaction	Close ISAM
DeleteISAMRecord	GetISAMRecords
DeleteISAMRecordByKey	ISAMRequest
GetISAMRecordsHold	NormalizeISAMStatus
HoldISAMDataSet	OpenISAM
HoldISAMRecord	QueryTransactionParams
ModifyISAMRecord	ReadISAMRecordByUri
ModifyISAMRecordByKey	ReadNextISAMRecord
ReadISAMRecordByUriHold	ReadUniqueISAMRecord
ReadNextISAMRecordHold	RollBackTransaction
ReadUniqueISAMRecordHold	SetTransactionParams
ReleaseISAMDataSet	SetUpISAMIterationLimits
ReleaseISAMRecord	SetUpISAMIterationPrefix
StoreISAMRecord	

NOTE

There are no transaction-related constraints for the following operations: CreateISAM, DeleteISAM, LoadSingle-UserISAM, RenameISAM, SetISAMProtection, and Verify-MultiuserISAM.

BeginTransaction is not allowed during a transaction.

To prevent concurrent modification by multiple application systems, ISAM allows an application system to lock a record or data set, thereby giving that application system 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. In either case, after both transactions are complete, the balance in their account remains the same as it was before the transactions.

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

	John Kelly's Transaction	Jill Kelly's Transaction
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.

	John Kelly's Transaction	Jill Kelly's Transaction
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 the manual banking system, the pair of transactions worked because the teller held the passbook (account record) during an entire transaction. With only a single passbook, the parallel transactions could not occur.

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

John Kelly's withdrawal is accomplished by:

1. starting the transaction
2. reading the account balance and locking the account record
3. subtracting $500 from the balance
4. writing the new balance to the account
5. ending the transaction, thereby releasing the account record

Jill Kelly's deposit is accomplished by:

1. starting the transaction
2. reading the account balance and locking the account record
3. adding $500 to the balance
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 system locks a record, only that application system has access to the record until it is released or unlocked. If an application system locks a data set, only that application system has access to the data set and all of its records until it is unlocked.

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

Conflicts and Deadlocks

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

A special case of conflict occurs when one application system has locked a record or data set and attempts to lock another one that is already locked by a second application system. A deadlock occurs if this second application system happens to be waiting for the record or data set locked by the first application system. Both application systems are queued for the record or data set locked by the other and can thus wait forever. This basic form of deadlock can become quite complicated when a number of application systems and requests are involved, as shown in figure 2-2.

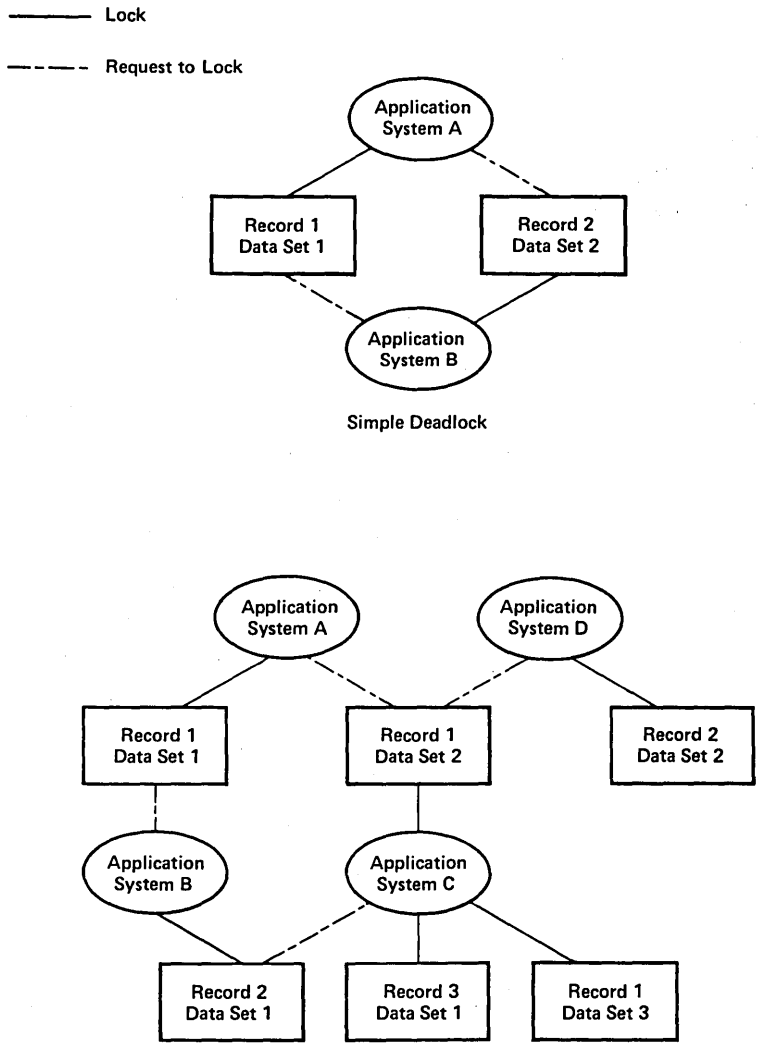

Figure 2-2. Examples of Deadlock

A similar problem occurs when an application system 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 system is waiting for the locked record or data set. The second application system may have to wait a long time.

To solve both problems, ISAM uses timeouts.

Timeouts

Deadlock is avoided in ISAM by using timeouts when an application system requests a record or data set that another user has already locked. The maximum time a request is queued is specified in a timeout value in the Transaction Parameters Block. (See table 3-8 in section 3.) The timeout value, wTicksWait, specifies the maximum time a request to lock a record or data set can be queued. If the time specified in wTicksWait is exceeded, ISAM reports that the record or data set is not available.

ISAM INSTALLATION

You can use ISAM on a standalone workstation or in cluster configurations. ISAM supports both single-user and multiuser access. Single-user ISAM is loaded as a task by the application system, and multiuser ISAM is installed as a system service. Multiuser ISAM can be installed in either a single-partition or multipartition BTOS.

Single-Partition BTOS

In a single-partition BTOS, ISAM is permanently installed in memory. Once you install ISAM, you cannot remove the ISAM server, nor can you reallocate its memory unless you rebootstrap BTOS.

Multipartition BTOS

In a multipartition BTOS, ISAM is installed in a secondary application partition. (See the B 20 *Systems Operating System (BTOS) Reference Manual* for further information about secondary application partitions.)

ISAM Configuration File

An ISAM configuration file specifies the sizes of the ISAM server's memory areas, based on the number of users. Default values are provided for this file. Additionally, there is a utility that modifies the file to improve performance or reduce the amount of memory.

ISAM UTILITIES

ISAM provides various utility commands that you can invoke from the Executive to maintain and modify ISAM data sets. The ISAM utilities are:

- ISAM Copy
- ISAM Create
- ISAM Delete
- ISAM Rename
- ISAM Reorganize
- ISAM Set Protection
- ISAM Status

DATA SECURITY

Data security is provided at each ISAM data set level through two associated passwords: a read password and a modify password. In addition, BTOS protects the files of an ISAM data set from unauthorized access.

DATA INTEGRITY

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

- Error logging. Errors discovered in the file structures that ISAM maintains are logged in the Log File [Sys]<Sys>Log.Sys. The PLog utility lists the contents of this Log File.

- 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.

- Write-through cache. ISAM maintains a set of I/O buffers that are used to bring segments of disk records into memory as needed. Whenever a record is stored or modified, ISAM writes the changed data in the buffers back to disk. ISAM always updates the disk before completing an operation that changes a data set. This policy makes the data less susceptible to damage from hardware or software failures, unless the failure occurs in the middle of a Modify, Store, or Delete operation. All files are completely updated; no partial modifications are held in memory without updating the disk.

If hardware or software failures damage an ISAM data set, you can recover undamaged records by using the Maintain File utility. You can then use the ISAM Reorganize utility to reconstruct the data set.

SECTION 3
ISAM OPERATIONS: PROCEDURES AND SERVICES

OVERVIEW

You use the ISAM procedures and services in application systems to create, access, manipulate, and manage data in data sets. You can write ISAM application systems in any of the Burroughs programming languages.

Table 3-1 categorizes procedures and services by function. The first part of this section contains brief descriptions of each functional category, including general information that applies to the use of the operations within the category. The operations are then presented in alphabetic order with a brief description of the operation, the procedural interface, and the request block parameters, where applicable.

Most of the ISAM services can be accessed either by a procedural interface or by the ISAMRequest and Wait operations. You use the ISAMRequest operation to send requests to ISAM because multiuser ISAM uses the Request operation while single-user ISAM uses the Send operation. ISAMRequest handles this difference.

Using the procedural interface is easier because most of the necessary housekeeping is performed automatically. Using the ISAMRequest and Wait operations is more powerful because of the potential for a greater degree of overlap between computation and I/O operations. Refer to Asynchronous Requests in this section for more information.

Previous releases of ISAM contain operations that are no longer standard in Release 4.0. Programs written to use earlier versions of ISAM that call these superseded operations will still run with ISAM 4.0 and need not be changed. Appendix B lists the services that are no longer part of the standard ISAM set of operations.

Table 3-1. ISAM Operations by Function

Data Set Management

CreateISAM
DeleteISAM
RenameISAM
SetISAMProtection

Data Set Access

CloseISAM
OpenISAM

Record Management

DeleteISAMRecord
DeleteISAMRecordByKey
ModifyISAMRecord
ModifyISAMRecordByKey
StoreISAMRecord

Single Record Access

ReadISAMRecordByUri
ReadISAMRecordByUriHold
ReadUniqueISAMRecord
ReadUniqueISAMRecordHold

Multiple Record Access (Iteration)

GetISAMRecords
GetISAMRecordsHold
ReadNextISAMRecord
ReadNextISAMRecordHold
SetUpISAMIterationLimits
SetUpISAMIterationPrefix

Record Locking

HoldISAMDataSet
HoldISAMRecord
ReleaseISAMDataSet
ReleaseISAMRecord

Transactions

BeginTransaction
CommitTransaction
QueryTransactionParams
RollBackTransaction
SetTransactionParams

ISAM Service Access

LoadSingleUserISAM
VerifyMultiuserISAM

Asynchronous Requests

ISAMRequest
NormalizeISAMStatus

STATUS BLOCK

All ISAM operations include the pStatusBlockRet parameter, which points to the memory address of a status block used to report errors to the application system. The 4-byte status block, shown in table 3-2, contains two status codes: erc and ercDetail.

Table 3-2. Status Block Format (pStatusBlockRet Parameter)

Offset	Size (bytes)	Field	Description
0	2	erc	Status code (see appendix A)
2	2	ercDetail	Detail status code (see appendix A)

erc is either 0 ("OK") or one of the ISAM status codes listed in appendix A. If erc is nonzero, ercDetail gives additional information about the error. For example, if a device error causes a sector of the index file to be unreadable, erc contains 3119 (**Index file error**) and ercDetail contains 301 (**I/O error**).

DATA SET MANAGEMENT

Following are discussions of operations, indexes and keys, and key types.

Operations

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

DeleteISAM — deletes the files of an open data set, thereby destroying the data set.

RenameISAM — changes the name of an existing data set; that is, the name of the data store and index files.

SetISAMProtection — changes the passwords used to gain access to an existing data set.

Indexes and Keys

An index is a structure designed to help you efficiently locate a particular record of a data set. An index is defined for a key field of a data set.

Index keys are based on the key field values for the records contained in a data set. The index is sorted by key values in ascending or descending order. Figure 3-1 shows an index for a data set.

An application system uses an index to read records in key order or to read a single record directly by a unique key. Using the Personnel data sets, 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 of these examples read the records by using an employee number index. To access records directly, as in the second example, the index must be composed of unique keys. Unique keys mean that only one record exists for each key value. Duplicate keys are not allowed.

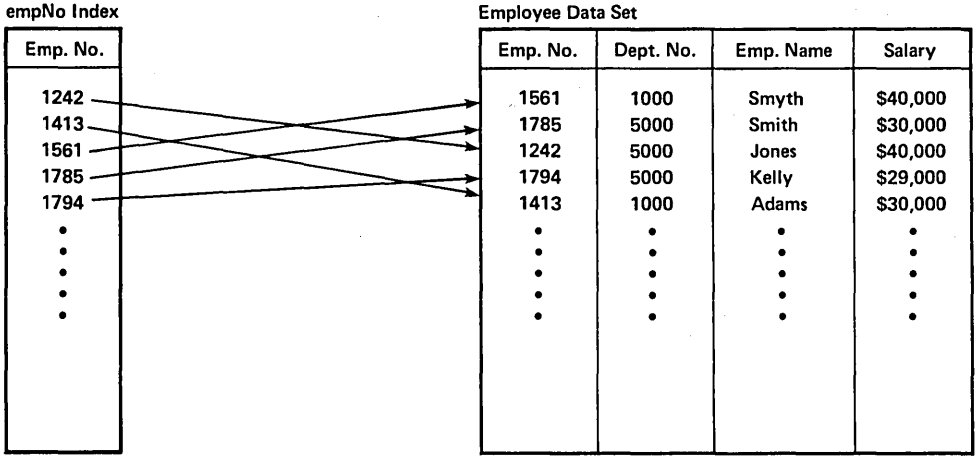

Figure 3-1. Index for a Data Set

Index keys can be simple or composite. A simple key is based on a single field. An index composed of simple keys is sorted 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. An index composed of composite keys is sorted by the first field. Groups of records with duplicate values for the first field are sorted again by the second field, and so forth. If the entire key is duplicated, records with duplicate values are accessed in random order. If a composite key is unique, such duplication is not allowed.

In a composite key:

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

For example, for a data set with the following structure:

Byte	Length	Type	Name
0	4	Byte string	deptNo
4	5	Byte string	empNo

A composite key (deptNo,empNo) can be defined, but (empNo,deptNo) cannot. The order of the fields does not permit (empNo,deptNo) keys.

A composite key is defined as a single key field. Using the example above, the composite key (deptNo,empNo) would be defined as a 9-byte byte string key located at offset 0.

Key Types

ISAM supports many different types of keys. This wide variety of key types enables most of the different data representations that can be specified in each of the Burroughs programming languages to be used as keys. You can use 12 different key types to specify an index. Each of the 12 types has a notation for non-COBOL applications (types 0 to 11) and a corresponding notation for COBOL applications (types 20 to 31).

A brief description of each key type follows. For more information on the relationships between key types and programming language representations refer to table 3-3.

Binary

A binary key is an unsigned 1- to 8-byte integer. The high-address byte of a binary key is the most significant for determining sort order on the workstation processor. For COBOL COMP fields, 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, and a distinction is made between upper- and lowercase ASCII characters. Byte strings have the same representation in all programming languages, including COBOL.

Character String

A character string key is a fixed-length string of 1 to 64 binary bytes. Like a byte string, a character string is sorted with the low-address byte as the most significant. Unlike a byte string, however, character string keys are sorted with no distinction between upper- and lowercase ASCII characters. Character strings have the same representation in all programming languages, including COBOL.

Decimal (Odd/)/Decimal (Even)

A decimal key contains two decimal digits in each byte, except for the last (high-address) byte where the rightmost four bits are reserved for a sign. (This format is the same as COBOL COMP-3.) Decimal (even) is used for values that have even numbers of digits, and decimal (odd) is for values with odd numbers of digits. The number of digits before the number is packed determines whether the (even) or (odd) decimal type is used. A decimal key can contain one to 18 decimal digits.

Decimal fields have the same representation in all programming languages, including COBOL.

Display

A display key is used in COBOL applications for USAGE is DISPLAY numeric fields. All of the COBOL sign options are supported. Display keys can be one to 19 bytes long, containing one to 18 decimal digits.

Integer

An integer key is a signed 1- to 8-byte integer. The high-address byte of an integer key is the most significant for determining sort order in a workstation application. For COBOL COMP fields, however, the low-address byte is the most significant.

Long/Short/Extended IEEE

Long IEEE, short IEEE, and extended IEEE keys are used for real numbers in Pascal or FORTRAN applications on workstations. A long IEEE key is 8 bytes long, a short IEEE is 4 bytes, and an extended IEEE is 10 bytes.

Long/Short Real

Long real and short real keys are used in BASIC applications on workstations. A long real key is an 8-byte real number, and a short real key is a 4-byte real number.

Byte and character strings are represented in the same manner in all of the programming languages, including COBOL. Decimal and display key types are also the same. For all of the other numeric types, however, the representations are different. The workstation representation uses the low-address byte as the least significant.

The general rule, therefore, is on workstations, for BASIC, FORTRAN, and PASCAL representations, use key types 0 to 11.

ISAM Description Block

When you create a data set with the CreateISAM operation, an ISAM Description Block supplies a description of the data set that includes the record size, key description, and sector allocation policies. The structure of this block is shown in table 3-4.

Table 3-3. ISAM Key Types and Programming Language Representations

Language and Key Type	Index Spec. cbIndexField	wType
BASIC Interpreter		
Integer (%)	2	7
ShortReal (!)	4	5
LongReal (#)	8	4
BASIC Compiler		
Integer (%)	2	7
ShortReal (!)	4	5
LongReal (#)	8	4
COBOL		
USAGE is DISPLAY (n-byte) (numeric types)	n	31
USAGE is COMP (n-byte) (signed)	n	27
USAGE is COMP (n-byte) (unsigned)	n	20
USAGE is COMP-3 (n-digit) (n even)	(n+2)/2	26
USAGE is COMP-3 (n-digit)	(n+2)/2	23

Table 3-3. ISAM Key Types and Programming Language Representations (Cont)

Language and Key Type	Index Spec. cbIndexField	wType
FORTRAN (Microsoft)		
INTEGER*2	2	7
INTEGER*4	4	7
REAL*4	4	9
REAL*8	8	8
DOUBLE PRECISION	8	8
FORTRAN-86		
(same as FORTRAN above)		
TEMPREAL	10	10
Pascal (Microsoft)		
Byte	1	0
Integer	2	7
Real	4	9
SInt	1	7
Word	2	0

Table 3-4. ISAM Description Block

Offset	Field	Size (bytes)	Description
0	lfaInitSizeIndexFile	4	This is the initial size of the index file of the data set. It must be a multiple of 512. The size defaults to 15360 (30 sectors) if lfaInitSizeIndexFile is 0.
4	qbGrowIndexFile	4	This is the number of bytes by which the index file grows when the existing memory is exhausted. The value must be a multiple of 512. The index file grows by a default of 15360 (30 sectors) if qbGrowIndexFile is 0.
8	cSectorsNode	2	This is the size of the B-tree nodes, in sectors. cSectorsNode must not exceed the index buffer size specified in the configuration file used to install ISAM. It defaults to that buffer size.
10	lfaInitSizeDataStoreFile	4	This is the initial size of the data store file of the data set. It must be a multiple of 512. The size defaults to 15360 (30 sectors) if lfaInitSizeDataStoreFile is 0.

Table 3-4. ISAM Description Block (Cont)

Offset	Field	Size (bytes)	Description
14	qbGrowDataStoreFile	4	This is the number of bytes by which the data store file grows when the existing file space is exhausted. The value must be a multiple of 512. The data store file grows by a default of 15360 (30 sectors) if qbGrowDataStore-File is 0.
18	sRecord	2	This is the size in bytes of the records in the data set. Records must be at least four bytes long.
20	cIndexes	2	This is the number of indexes for the data set.
22	rgIndexSpec	Indexes*20	This is the set of Index Specification Blocks for the data set. There is one ISAM Index Specification Block per index. The structure of an ISAM Index Specification Block is shown in table 3-5.

Table 3-5. ISAM Index Specification Block

Offset	Field	Size (bytes)	Description
0	rbIndexField	2	This is the offset of the key component in each record of the data set.
2	cbIndexField	2	This is the size of the key component in bytes.
4	wType	2	This is one of the values 0 to 11 (20 to 30 for COBOL) used to represent a key type as shown in table 3-6.
6	fAscending	2	This is TRUE (0FFh) if keys for this index are in ascending order, or FALSE (0h) if the keys are in descending order.
8	fNullIsIndexed	2	This is TRUE (0FFh) if null values (binary 0's) are indexed, or FALSE (0h) if null values are not indexed.

Table 3-5. ISAM Index Specification Block (Cont)

Offset	Field	Size (bytes)	Description
10	fDuplicatesAllowed	2	This is TRUE (0FFh) if duplicate values are valid for this index. It is FALSE (0h) 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 fDuplicatesAllowed 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 wType and cbIndexField together specify the type and size of the key component in each ISAM Index Specification Block. See table 3-6.

Table 3-6. Type of Key Component

Type	Name of Type	Description
0	Binary	cbIndexField contains the length of the key in bytes. 1 to 8 are valid values.
1	Byte	cbIndexField contains the length of the key in bytes. 1 to 64 are valid values.
2	Character	cbIndexField contains the length of the key in bytes. 1 to 64 are valid values.
3	Decimal (Odd)	cbIndexField contains (d + 2)/2, where d is the number of decimal digits in the key (d $\leq$ 18).
4	LongReal	cbIndexField must contain 8.
5	ShortReal	cbIndexField must contain 4.
6	Decimal (Even)	See Decimal (Odd) above for the value of cbIndexField. This type is used for keys with an even number of decimal digits.
7	Integer	cbIndexField contains the length of the key in bytes. 1 to 8 are valid values.
8	LongIEEE	cbIndexField must contain 8.
9	ShortIEEE	cbIndexField must contain 4.
10	ExtendedIEEE	cbIndexField must contain 10.
11	Display	cbIndexField contains the length of the key in bytes. 1 to 19 are valid values.

COBOL applications use the values 20 to 31 for the corresponding key types listed in this table.

DATA SET ACCESS

Information on data set access operations and ISAM handle follows.

Operations

CloseISAM — closes an open data set.

OpenISAM — opens an existing data set.

ISAM Handle

An ISAM handle is a word used to identify an open data set. The OpenISAM operation returns the ISAM handle. In subsequent ISAM operations, you use it as the parameter ISAMhandle. An ISAM handle is valid until the data set is closed or deleted.

RECORD MANAGEMENT AND ACCESS

Information on record management and access operations follows.

Record Management Operations

DeleteISAMRecord
: removes a record (identified by its unique record identifier) from a data set.

DeleteISAMRecordByKey
: removes a record (identified by a unique key) from a data set.

ModifyISAMRecord
: modifies an existing record (identified by its unique record identifier) in the data set and updates all indexes accordingly.

ModifyISAMRecordByKey
: modifies an existing record (identified by a unique key) in the data set and updates all indexes accordingly.

StoreISAMRecord
: creates a new record with specified data in a data set.

Single Record Access Operations

ReadISAMRecordByUri
: reads a record identified by its unique record identifier.

ReadISAMRecordByUriHold
: reads and locks a record identified by its unique record identifier.

ReadUniqueISAMRecord
: reads a record identified by a unique key.

ReadUniqueISAMRecordHold
: reads and locks a record identified by a unique key.

Multiple Record Access (Iteration) Operations

GetISAMRecords
: reads several records or unique record identifiers in key order.

GetISAMRecordsHold
: reads and locks several records or unique record identifiers in key order.

ReadNextISAMRecord
: reads the next record in key order.

ReadNextRecordHold
: reads and locks the next record in key order.

SetUpISAMIterationLimits
: initializes a sequence of Read operations for records with a specific key within a given range.

SetUpISAMIterationPrefix
: initializes a sequence of Read operations for records with a specific byte or character string key having a given prefix.

LOCKING

Information on Locking operations follows.

Operations

HoldISAMDataSet
: locks an ISAM data set.

HoldISAMRecord
: locks a record identified by its unique record identifier.

ReleaseISAMDataSet
: unlocks a locked data set without ending the current transaction.

ReleaseISAMRecord
: unlocks a locked record without ending the current transaction.

TRANSACTIONS

Information on Transaction operations follows.

Operations

BeginTransaction
: marks the start of a transaction.

CommitTransaction
: signifies the successful completion of a transaction and unlocks all records and data sets locked by the application system.

QueryTransactionParams
: accesses parameters associated with transactions for the application system.

RollBackTransaction
: signifies the unsuccessful completion of a transaction and unlocks all records locked by the application system. This operation does not undo any changes made to ISAM data sets.

SetTransactionParams
: sets parameters associated with transactions for the application system.

Transaction-Related Constraints

The constraints associated with each ISAM operation used during transaction processing are shown in table 3-7. These constraints do not apply to operations on data sets open for batch access, that is, in administrator, batch modify, or batch read mode.

Transaction Parameters Block

The Transaction Parameters Block (see table 3-8) contains values that control the operation of transactions. The wTicksWait field specifies the maximum time an application system can wait to lock a record or data set. The other fields do not affect ISAM data set access. For further information about the use of wTicksWait, see section 2.

You can examine these fields with the Query-TransactionParams operation and change them with the SetTransactionParams operation.

Table 3-7. Transaction-Related Constraints

Must Be in a Transaction	Must Not Be in a Transaction	No Transaction Constraints
	BeginTransaction	
		CloseISAM
CommitTransaction		
		CreateISAM *
		DeleteISAM **
DeleteISAMRecord		
DeleteISAMRecordByKey		
		GetISAMRecords
GetISAMRecordsHold		
HoldISAMDataSet		
HoldISAMRecord		
		ISAMRequest
ModifyISAMRecord		
ModifyISAMRecordByKey		
		NormalizeISAMStatus
		OpenISAM
		QueryTransactionParams
		ReadISAMRecordByUri
ReadISAMRecordByUriHold		
		ReadNextISAMRecord
ReadNextISAMRecordHold		
		ReadUniqueISAMRecord
ReadUniqueISAMRecordHold		
ReleaseISAMDataSet		
ReleaseISAMRecord		
		RenameISAM **
		RollBackTransaction
		SetISAMProtection **
		SetTransactionParams
		SetUpISAMIteration-Limits
		SetUpISAMIteration-Prefix
StoreISAMRecord		

* Not applicable. A data set is not open when you create it.

**Not applicable. A data set must be open in administrator mode to use this operation

Table 3-8. Transaction Parameters Block Format

Offset	Length	Field	Description
0	2	wTicksWait	A word specifying the maximum number of 0.1-second ticks a request to lock a record or data set is queued. The default value is 100.
5	1	Reserved	Padding returned as 0 (null).
6	*	Reserved	The asterisk indicates that it is for expansion and not required at present.

ISAM SERVICE ACCESS

Following is information on ISAM service access operations and using Single-user ISAM.

Operations

LoadSingleUserISAM
: loads ISAM as a task and initializes communication with ISAM.

VerifyMultiuserISAM
: establishes whether or not multiuser access to ISAM is available on the standalone or master workstation.

Using Single-User ISAM

An application system initializes single-user ISAM by calling the LoadSingleUserISAM operation. The single-user ISAM server does not inform BTOS that ISAM is serving requests. Instead, the application system builds request blocks and sends the requests directly to ISAM using the BTOS Send operation rather than the BTOS Request operation.

ISAM sends the requests back using the BTOS Send operation instead of the BTOS Respond operation.

If an application system and data set are at a master or standalone workstation, the application system can use either multiuser or single-user ISAM to access the data set. If an application system and data set are located at a cluster workstation, the application system must use single-user ISAM to access the data set. (Multiuser ISAM cannot be installed on a cluster workstation.) If an application system at a cluster workstation wants to access a data set at the master workstation, multiuser ISAM, installed at the master workstation, must be used.

Memory Usage

Single-user ISAM requires the same amount of memory as multiuser ISAM installed for a single user. (See appendix C for more information.) When you invoke the LoadSingleUserISAM operation, this memory is allocated as short-lived memory from the pool of unallocated memory available to the application system.

Using Either Multiuser or Single-User ISAM

An application system can be written to use either multiuser ISAM, if that is installed, or single-user ISAM. By incorporating calls to both multiuser and single-user ISAM, an application system is not dependent on multiuser ISAM server installation.

The application system calls the VerifyMultiUser-ISAM operation to check whether multiuser ISAM is available. If it is, a status code of 0 ("OK") is returned, and the application system uses multiuser ISAM. If any other status code is returned, the application system calls the LoadSingleUserISAM operation to load single-user ISAM.

ASYNCHRONOUS REQUESTS

Information on operations and asynchronous requests versus procedural requests follows.

Operations

ISAMRequest issues an ISAM request and returns without waiting for the request to be completed.

NormalizeISAMStatus ensures the status block returned by an asynchronous ISAM operation is valid.

Asynchronous Requests Versus Procedural Interface

You can access ISAM system services indirectly, by a procedural interface, or directly, by the ISAMRequest, Wait, and NormalizeISAMStatus operations.

Using the procedural interface is easier because it automatically performs most of the necessary housekeeping and issues the ISAMRequest and Wait operations.

Using the ISAMRequest, Wait, and NormalizeISAMStatus operations enables applications to run more quickly and efficiently because there is a greater degree of overlap between processing by ISAM and computation by the application system.

Only one ISAM operation should be outstanding at a time because the order in which the operations are performed is undefined. Overlapping certain operations (for example, doing a CommitTransaction operation while a ModifyISAMRecord operation is pending) can cause problems.

When using asynchronous requests, the Wait operation must be followed by the NormalizeISAMStatus operation. Otherwise, the values in the status block may not be valid.

PROCEDURE DEFINITIONS

Information on procedures follows.

BeginTransaction

description

The BeginTransaction procedure marks the start of a transaction for the application system. The transaction ends when the application system uses either a CommitTransaction or RollBack-Transaction operation.

BeginTransaction cannot be called from within a transaction.

procedural interface

BeginTransaction (pStatusBlockRet): ErcType

where

pStatusBlockRet
: is the memory address of the status block into which the status codes from the operation are returned.

request block

BeginTransaction is an object module procedure.

CloseISAM

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 this procedure is called.

procedural interface

```
CloseISAM (ISAMHandle, pStatusBlockRet): ErcType
```

where

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

pStatusBlockRet — is the memory address of the status block into which the status codes from the operation are returned.

request block

CloseISAM is an object module procedure.

description

The CommitTransaction procedure signifies the successful completion of a transaction and unlocks all records and data sets locked by the application system.

You can use CommitTransaction only in a transaction.

procedural interface

```
CommitTransaction (pStatusBlockRet): ErcType
```

where

pStatusBlockRet
: is the memory address of the status block into which the status codes from the operation are returned.

request block

CommitTransaction is an object module procedure.

description

The CreateISAM service defines the index structure of a data set and creates a new empty data set with the specified structure. The parameters that are supplied enable the application program to locate the data store and index files on different volumes and to control the allocation of disk sectors for the two files independently.

procedural interface

```
CreateISAM (pbFileSpecDataStoreFile,
            cbFileSpecDataStoreFile,
            pbPasswordDataStoreFileCreate,
            cbPasswordDataStoreFileCreate,
            pbFileSpecIndexFile,
            cbFileSpecIndexFile,
            pbPasswordIndexFileCreate,
            cbPasswordIndexFileCreate, pISAMDesc,
            sISAMDesc, pStatusBlockRet): ErcType
```

where

pbFileSpecDataStoreFile
cbFileSpecDataStoreFile
: describe a file specification for the data store file of the new data set.

pbPasswordDataStoreFileCreate
cbPasswordDataStoreFileCreate
: describe a password used to create the data store file of the new data set.

pbFileSpecIndexFile
cbFileSpecIndexFile
: describe a file specification for the index file of the new data set. If cbFileSpecIndexFile is 0, then the file specification for the index is derived from the file specification for the data store file. ISAM copies the file specification defined by pbPasswordDataStoreFileCreate and cbPasswordDataStoreFileCreate, then replaces the suffix beginning with the period

character with the characters .Ind . For example, if the file specification for the data store file is

```
[vol]<dir>DataSet.Isam
```

then the file specification for the index file is

```
[vol]<dir>DataSet.Ind
```

pbPasswordIndexFileCreate
cbPasswordIndexFileCreate
: describe a password used to create the index file of the new data set. If the name of the index file is null, then these two parameters are ignored, and the password specified for the data store file is used to create the index file as well.

pISAMDesc
sISAMDesc
: describe a memory area containing an ISAM Description Block, which has the structure shown in table 3-4.

pStatusBlockRet
: is the memory address of the status block into which the status codes from the operation are returned.

request block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	2	0
2	nReqPbCb	1	5
3	nRespPbCb	1	1
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	79
12	pbFileSpecDataStore-File	4	
16	cbFileSpecDataStore-File	2	
18	pbPasswordDataStore-FileCreate	4	
22	cbPasswordDataStore-FileCreate	2	
24	pbFileSpecIndexFile	4	
28	cbFileSpecIndexFile	2	
30	pbPasswordIndexFile-Create	4	
34	cbPasswordIndexFile-Create	2	
36	pISAMDesc	4	
40	sISAMDesc	2	
42	pStatusBlockRet	4	
46	sStatusBlock	2	4

DeleteISAM

description

The DeleteISAM service deletes the files of a data set, thereby destroying all information in the data set.

The data set must be open in adminstrator mode.

procedural interface

DeleteISAM (ISAMHandle, pStatusBlockRet): ErcType

where

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

pStatusBlockRet — is the memory address of the status block into which the status codes from the operation are returned.

request block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	2	2
2	nReqPbCb	1	0
3	nRespPbCb	1	1
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	80
12	ISAMHandle	2	
14	pStatusBlockRet	4	
18	sStatusBlock	2	4

description

The DeleteISAMRecord service removes a record from a data set. The data set must be open for modification. The disk space occupied by the record is made available for a subsequent StoreISAMRecord operation. All keys for the record are automatically removed from the indexes of the data set. All data in the record is destroyed.

If the data set is open in a transaction mode, then DeleteISAMRecord can be called only from within a transaction and when the record to be deleted is locked.

procedural interface

```
DeleteISAMRecord (ISAMHandle, uriRecord,
                  pStatusBlockRet): ErcType
```

where

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

uriRecord is the unique record identifier of the record to be deleted. uriRecord is usually the value returned by a previous Read operation.

pStatusBlockRet
is the memory address of the status block into which the status codes from the operation are returned.

request block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	2	6
2	nReqPbCb	1	0
3	nRespPbCb	1	1
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	81
12	ISAMHandle	2	
14	uriRecord	4	
18	pStatusBlockRet	4	
22	sStatusBlock	2	4

DeleteISAMRecordByKey

description

The DeleteISAMRecordByKey service removes a record from a data set. The data set must be open for modification. DeleteISAMRecordByKey, 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 operation. All keys for the record are automatically removed from the indexes of the data set. All data in the record is destroyed.

If the data set is open in a transaction mode, DeleteISAMRecordByKey can be called only from within a transaction. The record to be deleted does not need to be locked. DeleteISAMRecord-ByKey locks the record before deleting it.

procedural interface

```
DeleteISAMRecordByKey (ISAMHandle, iIndex, pKey,
                       sKey, pUriRecordRet,
                       pStatusBlockRet): ErcType
```

where

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

iIndex — identifies the index used. (The indexes 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 where the unique record identifier for the record to be deleted is returned.

pStatusBlockRet
: is the memory address of the status block into which the status codes from the operation are returned.

request block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	2	4
2	nReqPbCb	1	1
3	nRespPbCb	1	2
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	199
12	ISAMHandle	2	
14	iIndex	2	
16	pKey	4	
20	sKey	2	
22	pUriRecordRet	4	
26	sUriRecord	2	4
28	pStatusBlockRet	4	
32	sStatusBlock	2	4

description

The GetISAMRecords service reads several records or unique record identifiers in key order from a data set, with a single call to ISAM. Records are returned in key order for the current iteration. If there are no more records for the iteration, the status code 3127 **(No more records)** is returned.

The GetISAMRecordsHold service is identical to GetISAMRecords, except GetISAMRecordsHold also locks the records read (or those that have unique record identifiers returned). If records are read, then each one read is locked. If unique record identifiers are returned, then the record identified by each unique record identifier is locked. Thus, if GetISAMRecordsHold is used, none of the accessed records can be accessed by other application systems until they are released, usually by a subsequent Commit-Transaction operation.

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

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

For example, a particular GetISAMRecords operation 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 the seventh record in the sequence is locked by another application system, then the GetISAMRecords operation 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. Application

systems can thus "skip over" records locked by other application systems.

If the data set is open in a transaction mode, the application system does not have to be in a transaction before calling GetISAMRecords, but it must be in a transaction before calling GetISAMRecordsHold.

procedural interface

```
GetISAMRecords
GetISAMRecordsHold (ISAMHandle, fReadRecords,
                    pBuffer, sBuffer,
                    pCRecordsReadRet,
                    pStatusBlockRet): ErcType
```

where

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 are returned, and TRUE if records are returned also.

pBuffer
sBuffer — describe the memory area into which the unique record identifiers and records are to be read.

If only unique record identifiers are returned (fReadRecords is FALSE), then the record identifiers are packed into the buffer without padding. If both unique record identifiers and records are returned (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.

For example, the buffer structure after three 46-byte records are read is shown in table 3-9.

pCRecordsReadRet
: is the memory address of the word where the number of unique record identifiers and/or records read is returned.

pStatusBlockRet
: is the memory address of the status block into which the status codes from the operation are returned.

Table 3-9. Buffer Structure for GetISAMRecords and GetISAMRecordsHold when Records are Read (46-Byte Records)

Offset	Description	Size (bytes)
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

request block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	2	4
2	nReqPbCb	1	0
3	nRespPbCb	1	3
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	*
12	ISAMHandle	2	
14	fReadRecords	1	
15	reserved	1	0
16	pBuffer	4	
20	sBuffer	2	
22	pCRecordsReadRet	4	
26	sCRecordsRead	2	2
28	pStatusBlockRet	4	
32	sStatusBlock	2	4

```
*GetISAMRecords = 82;
 GetISAMRecordsHold = 0.
```

HoldISAMDataSet

description

The HoldISAMDataSet service locks an open data set.

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

procedural interface

```
HoldISAMDataSet (ISAMHandle, fLockRecords,
                 pStatusBlockRet): ErcType
```

where

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

fLockRecords specifies whether a subsequent ReleaseISAMDataSet operation 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 the data set is unlocked by a subsequent ReleaseISAMDataSet operation, locked records in the data set will continue to be locked.

If fLockRecords is FALSE, then if the data set is unlocked by the ReleaseISAMDataSet operation, all records in the data set will be unlocked.

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

1. Record 1 is locked (for example, with the HoldISAMRecord operation).

2. Data set X is locked with the HoldISAMDataSet operation.

3. Record 2 is locked (for example, with the ReadNextISAMRecordHold operation).

4. Data set X is unlocked with the ReleaseISAMDataSet operation.

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 into which the status codes from the operation are returned.

request block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	2	4
2	nReqPbCb	1	0
3	nRespPbCb	1	1
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	205
12	ISAMHandle	2	
14	fLockRecords	1	
15	reserved	1	0
16	pStatusBlockRet	4	
20	sStatusBlock	2	4

HoldISAMRecord

description

The HoldISAMRecord service locks a record identified by its unique record identifier. The record identifier is not checked for validity by HoldISAMRecord.

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

procedural interface

```
HoldISAMRecord (ISAMHandle, uriRecord,
                pStatusBlockRet): ErcType
```

where

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

uriRecord — is the unique record identifier of the record to be locked. uriRecord is usually the value returned by a previous Read operation.

pStatusBlockRet — is the memory address of the status block into which the status codes from the operation are returned.

request block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	2	6
2	nReqPbCb	1	0
3	nRespPbCb	1	1
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	130
12	ISAMHandle	2	
14	uriRecord	4	
18	pStatusBlockRet	4	
22	sStatusBlock	2	4

ISAMRequest

description

The ISAMRequest procedure issues an ISAM request and returns without waiting for the request to be completed.

This operation is used to issue asynchronous ISAM requests in both multiuser and single-user ISAM. For multiuser ISAM, ISAMRequest uses the Request operation, but for single-user ISAM, ISAMRequest uses the Send operation. (See the *B 20 Systems Operating System (BTOS) Reference Manual* for more information.) ISAMRequest allows application systems to issue asynchronous requests without regard to whether multiuser or single-user ISAM is being used.

procedural interface

```
ISAMRequest (pRq): ErcType
```

where

pRq — is the memory address of the request block specifying the ISAM.

request block

ISAMRequest is an object module procedure.

LoadSingleUserISAM

description

The LoadSingleUserISAM procedure loads ISAM as a task and initializes communication with ISAM. The ISAM task is loaded into short-lived memory allocated from the pool of unallocated memory available to the application system.

procedural interface

```
LoadSingleUserISAM (pbFileSpecRunFile,
                    cbFileSpecRunFile,
                    pbPwRunFile, cbPwRunFile,
                    pbConfigFile, cbConfigFile,
                    pbPwConfigFile,
                    cbPwConfigFile,
                    pStatusBlockRet): ercType
```

where

pbFileSpecRunFile
cbFileSpecRunFile
: describe the memory area containing the name of the ISAM run file to be used. If cbFileSpecRunFile is 0, the run file name defaults to [Sys]<Sys>ISAMServer.Run.

pbPwRunFile
cbPwRunFile
: describe the memory area containing the password used to open the ISAM run file.

pbConfigFile
cbConfigFile
: describe the memory area containing the name of the ISAM configuration file to be used. If cbConfigFile is 0, the configuration file name defaults to [Sys]<Sys>ISAMConfig.-Sys.

pbPwConfigFile
cbPwConfigFile
: describe the memory area containing the password used to open the ISAM configuration file.

pStatusBlockRet
is the memory address of the status block into which the status codes from the operation are returned.

request block

LoadSingleUserISAM is an object module procedure.

description

The ModifyISAMRecord service modifies an existing record in a data set and updates all indexes accordingly. The record is identified by its unique record identifier.

If any key is changed, then the record entry for the old value is removed from the index and the record is reindexed 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 record is not modified, the index is not changed, and status code 3118 (**Duplicate key**) is returned.

The data set must be open for modification. The record remains locked after modification.

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

procedural interface

```
ModifyISAMRecord (ISAMHandle, uriRecord, pRecord,
                  sRecord, pStatusBlockRet):
                  ErcType
```

where

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

uriRecord — is the unique record identifier of the record to be modified. uriRecord is usually the value returned by a previous Read operation.

pRecord
sRecord — describe the memory area containing the record to be written. sRecord must be equal to the record size for the data set.

pStatusBlockRet
: is the memory address of the status block into which the status codes from the operation are returned.

request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	2	6
2	nReqPbCb	1	1
3	nRespPbCb	1	1
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	84
12	ISAMHandle	2	
14	uriRecord	4	
18	pRecord	4	
22	sRecord	2	
24	pStatusBlockRet	4	
28	sStatusBlock	2	4

description

The ModifyISAMRecordByKey service modifies an existing record in a data set and updates all indexes accordingly. ModifyISAMRecordByKey, however, uses a unique key to identify the record instead of a record identifier.

If any key is changed, 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 3118 (**Duplicate key**) is returned.

The parameters for this operation include the index and the new record contents, but not a separate key for the record. The key used to identify 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.

Note that this service cannot be used to change the key being used to identify the record.

The data set must be open for modification.

If the data set is open in a transaction mode, then ModifyISAMRecordByKey can be called only from within a transaction. The record to be modified need not be locked; ModifyISAMRecordByKey locks the record before modifying it.

procedural interface

```
ModifyISAMRecordByKey (ISAMHandle, iIndex,
                       pRecord, sRecord,
                       pUriRecordRet,
                       pStatusBlockRet): ErcType
```

where

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

iIndex — identifies the index used. (The indexes are numbered from 0 in the order specified in CreateISAM.)

pRecord
sRecord — describe the memory area containing the record 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 where the unique record identifier of the modified record is returned.

pStatusBlockRet — is the memory address of the status block into which the status codes from the operation are returned.

request block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	2	4
2	nReqPbCb	1	1
3	nRespPbCb	1	2
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	198
12	ISAMHandle	2	
14	iIndex	2	
16	pRecord	4	
20	sRecord	2	
22	pUriRecordRet	4	
26	sUriRecord	2	2
28	pStatusBlockRet	4	
32	sStatusBlock	2	4

NormalizeISAMStatus

description

The NormalizeISAMStatus procedure ensures that both the ercRet field of the request equals the erc field of the status block and the ercDetail field of the status block is valid.

NormalizeISAMStatus must be called after asynchronous ISAM requests (issued by ISAMRequest) are completed. If it is not called, the status block that the ISAM operation returns has undefined contents.

The status code that NormalizeISAMStatus returns diagnoses problems encountered while normalizing the status block. In general, the status code returned will differ from the ercRet field in the request block. For example, if an asynchronous Read operation returns status code 3127 (**no more records**) and a detail status of 0 ("OK"), then NormalizeISAMStatus ensures that the ercRet field in the request block matches the status block erc field and returns 0 ("OK").

procedural interface

```
NormalizeISAMStatus (pRq): ErcType
```

where

pRq — is the memory address of the request block.

request block

NormalizeISAMStatus is an object module procedure.

OpenISAM

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 operations.

procedural interface

```
OpenISAM (pISAMHandleRet, pbDataSetName,
          cbDataSetName, pbPassword, cbPassword,
          mode, sRecord, pStatusBlockRet):
          ErcType
```

where

pISAMHandleRet
: is the memory address of the word into which the ISAM handle of the opened data set is 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 opened data set, or a file system password that gives modify access to the data set files. If the mode is batch modify or transaction modify, then you must supply the modify password. If the mode is batch read or transaction read, then you can supply either password. If the mode is administrator, then you must supply a file system password.

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

Value (ASCII)	Mode
ad	administrator
bm	batch modify
br	batch read
tm	transaction modify
tr	transaction read

In these ASCII constants, the first character is the high-order byte, and the second character is the low-order byte. This is the reverse of the byte order of strings in the Burroughs programming languages.

For further information about modes, refer to section 2.

sRecord — is the fixed-length record size for the data set.

pStatusBlockRet — is the memory address of the status block into which the status codes from the operation are returned.

request block

OpenISAM is an object module procedure.

QueryTransactionParams

description

The QueryTransactionParams procedure retrieves the application system's current transaction parameters. (See Transaction Parameters Block in this section for more information.)

procedural interface

```
QueryTransactionParams (pParamsRet, sParamsMax,
                        pSParamsRet,
                        pStatusBlockRet): ErcType
```

where

pParamsRet
sParamsMax — describe the memory area into which the transaction parameters are stored. The Transaction Parameters Block has the format shown in table 3-8.

pSParamsRet — is the memory address of a word into which the number of bytes retrieved are stored. The value stored in this word cannot exceed sParamsMax.

pStatusBlockRet — is the memory address of the status block into which the status codes from the operation are returned.

request block

QueryTransactionParams is an object module procedure.

ReadISAMRecordByUri
ReadISAMRecordByUriHold

description

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

The ReadISAMRecordByUriHold service is identical to ReadISAMRecordByUri, except ReadISAMRecordByUriHold also locks the record when it is read.

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

procedural interface

```
ReadISAMRecordByUri
ReadISAMRecordByUriHold (ISAMHandle, uriRecord,
                         pRecordRet, sRecord,
                         pStatusBlockRet):
                         ErcType
```

where

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

uriRecord — is the unique record identifier of the record to be read. uriRecord is usually the value that a previous Read operation returns.

pRecordRet
sRecord — describe the memory area into which the record is read. sRecord must be equal to the record size for the data set.

pStatusBlockRet — is the memory address of the status block into which the status codes from the operation are returned.

request block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	2	6
2	nReqPbCb	1	0
3	nRespPbCb	1	2
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	*
12	ISAMHandle	2	
14	uriRecord	4	
18	pRecordRet	4	
22	sRecord	2	
24	pStatusBlockRet	4	
28	sStatusBlock	2	4

```
*ReadISAMRecordByUri = 86;
 ReadISAMRecordByUriHold = 131.
```

ReadNextISAMRecord
ReadNextISAMRecordHold

description

The ReadNextISAMRecord service reads the next record in key order from a data set. The unique record identifier of the record is returned.

The ReadNextISAMRecordHold service is identical to ReadNextISAMRecord, except ReadNextISAM-RecordHold also locks the record when it is read.

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

procedural interface

```
ReadNextISAMRecord
ReadNextISAMRecordHold (ISAMHandle, pRecordRet,
                        sRecord, pUriRecordRet,
                        pStatusBlockRet): ErcType
```

where

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

pRecordRet
sRecord — describe the memory area into which the record is read. sRecord must be equal to the record size for the data set.

pUriRecordRet — is the memory address of the 4-byte structure into which the unique record identifier of the record is returned.

pStatusBlockRet — is the memory address of the status block into which the status codes from the operation are returned.

request block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	2	2
2	nReqPbCb	1	0
3	nRespPbCb	1	3
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	*
12	ISAMHandle	2	
14	pRecordRet	4	
18	sRecord	2	
20	pUriRecordRet	4	
24	sUriRecord	2	4
26	pStatusBlockRet	4	
30	sStatusBlock	2	4

```
*ReadNextISAMRecord = 87;
 ReadNextISAMRecordHold = 132.
```

description

The ReadUniqueISAMRecord service reads a record uniquely identified by a given key from a data set. Duplicates are not allowed for the index used, so that the key appears at most in one record of the data set.

The ReadUniqueISAMRecordHold service is identical to ReadUniqueISAMRecord, except ReadUniqueISAMRecordHold also locks the record.

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

procedural interface

```
ReadUniqueISAMRecord
ReadUniqueISAMRecordHold  (ISAMHandle, iIndex,
                           pKey, sKey,
                           pRecordRet, sRecord,
                           pUriRecordRet,
                           pStatusBlockRet):
                           ErcType
```

where

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

iIndex — identifies the index used. (The indexes are numbered from 0 in the order specified in CreateISAM.)

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 read. sRecord must be equal to the record size for the data set.

pUriRecordRet
: is the memory address of the 4-byte structure into which the unique record identifier of the record is returned.

pStatusBlockRet
: is the memory address of the status block into which the status codes from the operation are returned.

request block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	2	4
2	nReqPbCb	1	1
3	nRespPbCb	1	3
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	*
12	ISAMHandle	2	
14	iIndex	2	
16	pKey	4	
20	sKey	2	
22	pRecordRet	4	
26	sRecord	2	
28	pUriRecordRet	4	
32	sUriRecord	2	4
34	pStatusBlockRet	4	
38	sStatusBlock	2	4

*ReadUniqueISAMRecord = 88;
ReadUniqueISAMRecordHold = 133.

ReleaseISAMDataSet

description

The ReleaseISAMDataSet service releases a locked data set without ending the current transaction.

NOTE

Use this operation with care, since other application systems can hold and even modify the released data set before the transaction is ended.

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

procedural interface

```
ReleaseISAMDataSet (ISAMHandle, pStatusBlockRet):
                                    ErcType
```

where

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

pStatusBlockRet — is the memory address of the status block into which the status codes from the operation are returned.

request block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	2	2
2	nReqPbCb	1	0
3	nRespPbCb	1	1
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	206
12	ISAMHandle	2	
14	pStatusBlockRet	4	
18	sStatusBlock	2	4

ReleaseISAMRecord

description

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

NOTE

Use this operation with care, since other application systems can hold and even modify the released record before the transaction is ended.

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

procedural interface

```
ReleaseISAMRecord (ISAMHandle, uriRecord,
                   pStatusBlockRet): ErcType
```

where

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

uriRecord — is the unique record identifier for the record to be released. uriRecord is usually the value returned by a previous Read operation.

pStatusBlockRet — is the memory address of the status block into which the status codes from the operation are returned.

request block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	2	6
2	nReqPbCb	1	0
3	nRespPbCb	1	1
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	134
12	ISAMHandle	2	
14	uriRecord	4	
18	pStatusBlockRet	4	
22	sStatusBlock	2	4

RenameISAM

description

The RenameISAM service changes the name of an existing data set by changing the name of both the data store and index file for the data set. It is invalid to rename only one of the two files.

RenameISAM is implemented by using two invocations of the BTOS RenameFile operation; one to change the name of the data store file and one to change the name of the index file of the data set.

There are certain RenameFile operations, such as renaming a file from one volume to another, or renaming a file using an incorrect password, that are invalid. If a RenameISAM operation is attempted where one of the two required BTOS RenameFile operations is invalid, ISAM detects the error and renames the data set by using a valid name for both the data store and index file. One or both of the files, in this case, may retain the original name.

You must open the data set in administrator mode.

procedural interface

```
RenameISAM (ISAMHandle, pbFileSpecDataStoreFile,
            cbFileSpecDataStoreFile,
            pbPasswordDataStoreFileRename,
            cbPasswordDataStoreFileRename,
            pbFileSpecIndexFile,
            cbFileSpecIndexFile,
            pbPasswordIndexFileRename,
            cbPasswordIndexFileRename,
            pStatusBlockRet): ErcType
```

where

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

pbFileSpecDataStoreFile
cbFileSpecDataStoreFile
— describe the file specification for the new name of the data store file for the data set.

pbPasswordDataStoreFileRename
cbPasswordDataStoreFileRename
describe the password used when renaming the data store file for the data set.

pbFileSpecIndexFile
cbFileSpecIndexFile
describe a file specification for the new name of the index file for the data set. If cbFileSpecIndex-File is 0, then the file specification for the index is derived from the file specification for the data store file. ISAM copies the file specification defined by pbPasswordDataStoreFile and cbPass-wordDataStoreFile. Then ISAM replaces the suffix (beginning with the period character) with the characters .Ind . For example, if the file specification for the data store file is

```
[vol]<dir>DataSet.Isam
```

then the file specification for the index file is

```
[vol]<dir>DataSet.Ind
```

pbPasswordIndexFileRename
cbPasswordIndexFileRename
describe the password used when renaming the index file of the data set. If the name of the index file is null, then these two parameters are ignored and the password specified for the data store file is also used to create the index file.

pStatusBlockRet
is the memory address of the status block into which the status codes from the operation are stored.

request block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	2	2
2	nReqPbCb	1	4
3	nRespPbCb	1	1
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	89
12	ISAMHandle	2	
14	pbFileSpecDataStore-File	4	
18	cbFileSpecDataStore-File	2	
20	pbPasswordDataStore-FileRename	4	
24	cbPasswordDataStore-FileRename	2	
26	pbFileSpecIndexFile	4	
30	cbFileSpecIndexFile	2	
32	pbPasswordIndexFile-Rename	4	
36	cbPasswordIndexFile-Rename	2	
38	pStatusBlockRet	4	
42	sStatusBlock	2	4

description

The RollBackTransaction procedure signifies the unsuccessful end of a transaction and unlocks all records and data sets locked by the application system.

Although you can call RollBackTransaction, it is effective only when the calling application system is in a transaction.

procedural interface

```
RollBackTransaction (pStatusBlockRet): ErcType
```

where

pStatusBlockRet
: is the memory address of the status block into which the status codes from the operation are returned.

request block

RollBackTransaction is an object module procedure.

SetISAMProtection

description

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

SetISAMProtection does not change the file system passwords for the data set files. You use the Executive's Set Protection command to change these passwords. You must open the data set in administrator mode.

procedural interface

```
SetISAMProtection (ISAMHandle,
                   pbPasswordOpenRead,
                   cbPasswordOpenRead,
                   pbPasswordOpenModify,
                   cbPasswordOpenModify,
                   pStatusBlockRet): ErcType
```

where

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

pbPasswordOpenRead
cbPasswordOpenRead
describe the password that opens the data set for reading in either batch read or transaction read modes. The password cannot be longer than 12 bytes.

pbPasswordOpenModify
cbPasswordOpenModify
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 12 bytes.

pStatusBlockRet
is the memory address of the status block into which the status codes from the operation are returned.

request block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	2	2
2	nReqPbCb	1	2
3	nRespPbCb	1	1
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	90
12	ISAMHandle	2	
14	pbPasswordOpenRead	4	
18	cbPasswordOpenRead	2	
20	pbPasswordOpenModify	4	
24	cbPasswordOpenModify	2	
26	pStatusBlockRet	4	
30	sStatusBlock	2	4

description

The SetTransactionParams procedure sets the application system's current transaction parameters. (See Transaction Parameters Block in this section for more information.)

procedural interface

```
SetTransactionParams (pParams, sParams,
                      pStatusBlockRet): ErcType
```

where

pParams
sParams — describe the memory area containing the transaction parameters to be set. The Transaction Parameters Block has the format shown in table 3-8. (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.

pStatusBlockRet — is the memory address of the status block into which the status codes from the operation are returned.

request block

SetTransactionParams is an object module procedure.

SetUpISAMIterationLimits

description

The SetUpISAMIterationLimits service initializes a sequence of Read operations for records that have keys for a specified index within a given range. Subsequent calls to GetISAMRecords(Hold) and ReadNextISAMRecord(Hold) operations read each record that has a key value within the range. Records are read in key value order.

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

procedural interface

```
SetUpISAMIterationLimits (ISAMHandle, iIndex,
                          matchKind, pKey1,
                          sKey1, pKey2, sKey2
                          pStatusBlockRet):
                          ErcType
```

where

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

iIndex — identifies the index used. (The indexes are numbered from 0 in the order specified in CreateISAM.)

matchKind — is a word specifying which records will be retrieved according to the following:

- 0 — all records, regardless of their key values. key1 and key2 are both ignored; sKey1 and sKey2 should be 0.
- 1 — all records containing a key less than key1. key2 is ignored, and sKey2 should be 0.
- 2 — all records containing a key less than or equal to key1. key2 is ignored, and sKey2 should be 0.

3 all records containing a key equal to key1. key2 is ignored, and sKey2 should be 0.

4 all records containing a key greater than or equal to key1. key2 is ignored, and sKey2 should be 0.

5 all records containing a key greater than key1. key2 is ignored, and sKey2 should be 0.

6 all records containing a key greater than key1, but less than key2.

7 all records containing a key greater than or equal to key1, but less than key2.

8 all records containing a key greater than or equal to key1, but less than or equal to key2.

9 all records containing a key greater than key1, but less than or equal to key2.

pKey1
sKey1 describe the memory area containing a key record. The key affects the set of records that subsequent read operations return (see matchKind).

pKey2
sKey2 describe the memory area containing a key record. The key affects the set of records that subsequent read operations return (see matchKind).

pStatusBlockRet
is the memory address of the status block into which the status codes from the operation are returned.

request block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	2	6
2	nReqPbCb	1	2
3	nRespPbCb	1	1
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	210
12	ISAMHandle	2	
14	iIndex	2	
16	matchKind	1	
17	reserved	1	0
18	pKey1	4	
22	sKey1	2	
24	pKey2	4	
28	sKey2	2	
30	pStatusBlockRet	4	
34	sStatusBlock	2	4

SetUpISAMIterationPrefix

description

The SetUpISAMIterationPrefix service initializes a sequence of Read operations for records with keys for a specified index having a given prefix. Subsequent calls to GetISAMRecords(Hold) and ReadNextISAMRecord(Hold) operations 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 is not defined to include null values (binary 0's), then records with null keys are not read.

procedural interface

```
SetUpISAMIterationPrefix (ISAMHandle, iIndex,
                          pKey, sKey,
                          pStatusBlockRet):
                          ErcType
```

where

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

iIndex — identifies the index used. (The indexes 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 into which the status codes from the operation are returned.

request block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	2	4
2	nReqPbCb	1	1
3	nRespPbCb	1	1
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	92
12	ISAMHandle	2	
14	iIndex	2	
16	pKey	4	
20	sKey	2	
22	pStatusBlockRet	4	
26	sStatusBlock	2	4

description

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

The indexes 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 3118 (**duplicate key**) is returned.

The data set must be open for modification.

If the data set is open in transaction mode, then StoreISAMRecord can be called only from within a transaction. The created record is locked after StoreISAMRecord is called.

procedural interface

```
StoreISAMRecord (ISAMHandle, pRecord, sRecord,
                 pUriRecordRet, pStatusBlockRet):
                 ErcType
```

where

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 where the unique record identifier of the record to be stored is returned.

pStatusBlockRet — is the memory address of the ISAM status block in which the status codes from the operation are returned.

request block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	2	6
2	nReqPbCb	1	2
3	nRespPbCb	1	1
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	94
12	ISAMHandle	2	
14	pRecord	4	
18	sRecord	2	
20	pUriRecordRet	4	
24	sUriRecord	2	4
26	pStatusBlockRet	4	
30	sStatusBlock	2	4

VerifyMultiUserISAM

description

The VerifyMultiUserISAM procedure sends a request to ISAM at either the BNET (Burroughs Network) node where the application system is running or at another BNET node. If ISAM is installed, a status code of 0 ("OK") is returned. Any other status code indicates that multiuser ISAM is not available.

procedural interface

```
VerifyMultiUserISAM (pbNode, cbNode,
                     pStatusBlockRet): ercType
```

where

pbNode
cbNodedescribe the memory area containing the name of the BNET node. The default node is the master workstation for the cluster or the standalone workstation, where the application system is running.

pStatusBlockRet
is the memory address of the status block into which the status codes from the operation are returned.

request block

VerifyMultiUserISAM is an object module procedure.

SECTION 4
ISAM UTILITIES

INTRODUCTION

ISAM provides utilities for data set maintenance. You invoke the ISAM utilities from the Executive. They require exclusive control of the data set you want them to maintain. Table 4-1 gives a brief description of each utility.

This section describes the ISAM utilities used for data set maintenance. The ISAM Reorganize utility, which is used to recover lost data and to build a data set from any standard access method file, is described in section 5. ISAM Install, a utility used to install the multiuser ISAM server, is described in section 6.

Invoking the Utilities

To invoke any of the utilities, you type the name (or as many letters from the name as required to make it unique) in the command field and press RETURN. You then fill in the various fields in the utility form and press GO.

Optional Fields

On the form, optional fields are enclosed in brackets. The default values are included with the field name.

Fields Ending With a Question Mark

Fields ending with a question mark require a response of Yes or No. The default value is always No.

You use file system passwords to open a data set in administrator mode. All ISAM utilities use these passwords.

Whenever a password is required, the logged-in password is used unless you provide a password as part of a file specification. To type a password as part of a file specification, you type the file specification, a caret character (^), and then the password. The combined file specification and password must not contain any embedded spaces. To maintain security, the characters of the password are echoed by the Executive as number signs (#).

Default Index File Name

Several ISAM utilities include an optional field for an index file specification. If you do not enter a file specification, a default index file specification is derived from the data store file specification. ISAM copies the file specification for the data store and replaces the suffix after the period character with the suffix Ind . For example, if the file specification for the data store is

```
[vol]<dir>DataSet.Isam
```

then the file specification for the index file is

```
[vol]<dir>DataSet.Ind
```

If the data store file is

```
DataSet
```

then the index file is

```
DataSet.Ind
```

Table 4-1. ISAM Utilities

Utility	Description
ISAM Copy	Copies the files of a data set to produce a new data set. The passwords for the new data set remain the same as those used for the original data set.
ISAM Create	Creates an empty data set with the specified record size and index fields.
ISAM Delete	Deletes both the data store and index files of the data set and destroys all the data in the data set.
ISAM Rename	Renames the data store and index files to rename the data set. The passwords are unchanged.
ISAM Set Protection	Changes the passwords used to gain access to an existing data set.
ISAM Status	Displays information about a data set. The information can be printed or written to a disk file, in addition to being displayed.

ISAM COPY

The ISAM Copy utility creates a new data set by copying an existing one. Both the data store file and the index file of the existing data set are copied.

ISAM Copy Form

```
ISAM Copy
  ISAM data set from  _____________________________
  ISAM data set to    _____________________________
  [Index file to]     _____________________________
  [Overwrite ok?]     _____________________________
```

where

ISAM data set from
: is the file specification for the data store file of the data set to be copied. The password supplied must be the file system password; that is, it must be the volume, directory, or file password for both of the data set files.

ISAM data set to
: is the file specification for the data store file of the new data set. The password supplied must be the volume or directory password for the new data store file.

[Index file to]
: is the file specification for the index file of the new data set. The default file specification, which is used if this field is not specified, is derived from the data store file specification as described in Default Index File Name in this section. The password specified must be the volume or directory password for the new index file.

[Overwrite ok?]
: is Yes or No (the default).

  If you specify Yes, the copy is performed. If existing files have the same names, they are overwritten without issuing a confirmation message.

If you specify No, or leave the field blank, and either of the files for the new data set exists, a prompt is issued to confirm overwriting the existing file. Overwriting an existing file destroys the data in it. You press **GO** to confirm the overwrite, **CANCEL** or **FINISH** to deny it.

Example

```
ISAM Copy
  ISAM data set from Employee.ISAM
  ISAM file to EmployeeNew.ISAM
  [Index file to]
  [Overwrite ok?
```

This example copies a new data set from an existing one. Since you have not entered the index file specification, the default index specification, in this case EmployeeNew.Ind, is created. If a file called EmployeeNew.ISAM or EmployeeNew.Ind already exists, ISAM prompts for an overwrite confirmation by default.

ISAM CREATE

The ISAM Create utility creates an empty data set with the record size and index fields that you enter on the ISAM Create form.

ISAM Create Form

```
ISAM Create
  ISAM data set                                  _____
  [Index file]                                   _____
  Record size (e.g., 20 bytes)                   _____
  Index keys (e.g., Byte:10.8.ANU.w)             _____
  [B-Tree node size
    (default 2 sectors)]                         _____
  [Growth increment for data store
    file (default 30 sectors)]                   _____
  [Growth increment for index file
    (default 30 sectors)]                        _____
  [Initial size of data store file
    (default 30 sectors)]                        _____
  [Initial size of index file
    (default 30 sectors)]                        _____
  [Overwrite ok?]                                _____
```

where

ISAM data set is the file specification for the data store file of the data set to be created. The password supplied must be the volume or directory password for the new data store file.

[Index file] is the file specification for the index file of the new data set. [Index file] defaults as described in the Default Index File Name in this section. The password you supply must be the volume or directory password for the new index file.

Record size (e.g., 20 bytes)
is the size, in bytes, of the records in the new data set. Records must be at least 4 bytes long. Maximum record size is 65,528 bytes.

Index keys (e.g., Byte:10.8.ANU.W)]
is the parameter list that specifies the index fields of the

data set. Each parameter in the list is specified in the following format (no embedded spaces):

t:l.o.anu.w

or

t.o.anu.w

where

t is the type of the field.

Binary
Byte
Character
Decimal
Display
Integer
LongIEEE
ShortIEEE
ExtendedIEEE
LongReal
ShortReal

l is the length of the field in bytes for byte or character strings, the number of digits for decimal, or the number of bytes for binary, display, and integer numbers. Binary fields default to two bytes if you omit this entry. Long, short, and extended IEEE index fields and long and short real index fields do not use this entry.

o is the byte offset in the record for the index field. You specify it as a decimal number.

a is A if you want the key order for this key to be ascending, or D if you want the key order to be descending.

n is N if you want null values (binary 0's) indexed, and S if you want null values suppressed.

u is U if you want the key to be unique, and D if you want duplicate keys permitted.

w is W for non-COBOL application systems, and M for COBOL application systems. This field is optional; the default is W.

[B-Tree node size (default 2 sectors)]
is the number of sectors in the B-Tree nodes for the new data set. Maximum value is 12 sectors.

[Growth increment for data store file (default 30 sectors)]
[Growth increment for index file (default 30 sectors)]
[Initial size of data store file (default 30 sectors)]
[Initial size of index file (default 30 sectors)]
are values used to avoid wasting disk space and to avoid disk fragmentation, which is caused by excessive numbers of disk extents.

[Overwrite ok?]
is Yes or No (the default).

If you specify Yes, the data set is created.

If you specify No, or leave the field blank, and either of the files for the new data set exists, a prompt is issued to confirm overwriting the existing file. Overwriting an existing file destroys the data in it. You press **GO** to confirm the overwrite, or press **CANCEL** or **FINISH** to deny the overwrite.

Example

```
ISAM Create
ISAM data set                          Employee.ISAM
[Index file]                           _____________
Record size (e.g., 20 bytes)           43
Index keys (e.g.,
  Byte:10.8.ANU.W)                     @Employee.Keys
[B-Tree node size
  (default 2 sectors)]                 _____________
[Growth increment for data store
  file (default 30 sectors)]           _____________
[Growth increment for index file
  (default 30 sectors)]                _____________
[Initial size of data store file
  (default 30 sectors)]                _____________
[Initial size of index file
  (default 30 sectors)]                _____________
[Overwrite ok?]                        _____________
```

This example creates the Employee data set of the Personnel example. The data set name is Employee.ISAM, and the index file is Employee.Ind, by default. The record size is defined as 43 bytes. Employee.Keys is a text file containing the following index definitions:

```
BYTE:5.4.ANU.W
CHARACTER:30.9.AND.W
BYTE:9.0.ANU.W
```

The data store and index file sizes and growth increments use the default values. Both files are created with an initial size of 30 sectors. Each time either of the files is full, the length is extended by 30 sectors.

By default, a prompt is issued to confirm overwriting if either Employee.ISAM or Employee.Ind exists.

ISAM DELETE

The ISAM Delete utility deletes both the data store file and the index file of the data set. It destroys all the data in the data set.

ISAM Delete Form

```
ISAM Delete
  ISAM data set _____________________________
```

where

ISAM data set
: is the file specification for the data store file of the data set you want to delete. The password you supply must be the file system password; that is, it must be the volume, directory, or file password for both files of the data set.

Example

```
ISAM Delete
  ISAM data set  Employee.ISAM_____________________
```

This example deletes the Employee data set of the Personnel example. All records in Employee.ISAM, including the index file Employee.Ind, are deleted.

ISAM RENAME

The ISAM Rename utility changes the name of an existing data set by changing the name of both the data store file and the index file of the data set. It is invalid to rename only one of the two files.

You implement ISAM Rename by using two invocations of the BTOS RenameFile operation: one to change the name of the data store file and one to change the name of the index file of the data set.

There are certain RenameFile operations, such as renaming a file from one volume to another, or renaming a file using an incorrect password, that are invalid. If one of the two required BTOS RenameFile operations is invalid, ISAM Rename detects the error and renames the data set by using a valid name for both the data store and index files. One or both of the files, in this case, may retain the original name.

ISAM Rename Form

```
ISAM Rename
  ISAM data set from  _____________________________
  ISAM data set to    _____________________________
  [Index file to]     _____________________________
  [Overwrite ok?]     _____________________________
```

where

ISAM data set from
: is the file specification for the data store file of the data set you are renaming. The password you supply must be the file system password, that is, it must be the volume, directory, or file password for both files of the data set.

ISAM data set to
: is the file specification for the data store file of the renamed data set. The volume you specify in "ISAM data set to" must be the same volume specified in ISAM data set from." The password you supply must be the volume or directory password for the renamed data store file.

[Index file to]
is the file specification for the index file of the renamed data set. If you do not make an entry, the default index file specification is derived as described in Default Index File Name in this section. The volume specified in [Index file to] must be the same volume that the existing index file resides on. The password supplied must be the volume or directory password for the new index file.

[Overwrite ok?]
is Yes or No (the default).

If you specify Yes, the data set is renamed.

If you specify No, or leave the field blank, and either of the files for the new data set exists, a prompt is issued to confirm overwriting the existing file. Overwriting an existing file destroys the data in it. You press **GO** to confirm the overwrite. You press **CANCEL** or **FINISH** to deny it.

Example

```
ISAM Rename
  ISAM data set from  Employee.ISAM
  ISAM data set to    <NewDir>Employee.ISAM
  [Index file to]
  [Overwrite ok?]
```

This example renames the Employee data set of the Personnel example. The new data store file name has the same name as the old data set, but the directory on the logged-in volume has been changed. The index file specification is derived from the data set file specification by default. The new index file name is <NewDir>Employee.Ind.

ISAM SET PROTECTION

The ISAM Set Protection utility enters or modifies the passwords that permit access to a data set. ISAM Set Protection does not change the file system passwords for the files of the data set. You can change the file system passwords using the Executive Set Protection command. All ISAM utilities require file system passwords. You use passwords to open data sets in administrator mode also. See section 2 for more information on the use of administrator mode.

ISAM Set Protection Form

```
ISAM Set Protection
  ISAM data set                    _____________________
  [Password for modification]      _____________________
  [Password for reading]           _____________________
```

where

ISAM data set
: is the file specification for the data store file of the data set. This is the data set for which you are modifying the access passwords. The password you supply must be the file system password; that is, it must be the volume, directory, or file password for both files of the data set.

[Password for modification]
: is the new password for modify access to the data set. If you do not enter this password, any password permits modify access. Passwords have a maximum length of 12 characters.

[Password for reading]
: is the new password for read access to the data set. If you do not enter this password, any password permits read access. Passwords have a maximum length of 12 characters.

Example

```
ISAM Set Protection
  ISAM data set                 Employee.ISAM
  [Password for modification]   xxy
  [Password for reading]        xxz
```

This example protects the Employee.ISAM data set for read and modify access. Batch and transaction read modes require either xxy or xxz as the password, and batch and transaction modify modes require xxy as the password. Administrator mode still requires the file system password.

ISAM STATUS

The ISAM Status utility produces a status report for a data set on the video screen. The status report can also be printed or written to a disk file. The information displayed includes the file names, sizes, growth increments, the record and B-tree node sizes, and a description of each index, including the depth of each B-tree. If you entered YES in response to [Details?], the following additional information is included:

- the number of records (and deleted records) in the data set
- the number of B-tree nodes for each index (and the number of deleted nodes)
- the number of records indexed under each key
- the average percentage of node space currently used for each node of each index

Retrieving this additional information involves scanning the files of the data set, so ISAM Status takes significantly longer to execute if the details are requested.

Figure 4-1 shows both forms of the ISAM Status report.

ISAM Status Form

```
ISAM Status
  ISAM data set  _____________________________
  [Log file]     _____________________________
  [Details?]     _____________________________
```

where

ISAM data set
: is the file specification for the data store file of the data set for which you are generating the status report. The password you supply must be the file system password; that is, it must be the volume, directory, or file password for both files of the data set.

[Log file]
: is the file specification for the file you want the status report written to. If you do not specify

a log file, the report appears only on the video display.

[Details?] is Yes or No (the default).

Yes displays the additional details described previously.

If you specify No, or leave the field blank, the details described previously are not displayed.

Example

```
ISAM Status
  ISAM data set  Employee.ISAM
  [Log file]
  [Details?]     Yes
```

This example produces a status report with details as shown in the second part of figure 4-1.

```
Data store file                                    [Win]<Personnel>Employee.ISAM
Index file                                         [Win]<Personnel>Employee.Ind
Last accessed                                      Thu Jul 21, 1983 10:09 AM
Last modified                                      Thu Jul 21, 1983 10:09 AM
Record size (bytes)                                43
Node size (sectors)                                2
Size of Data store file (sectors)                  5
Size of Index file (sectors)                       10
Growth increment for Data store file (sectors)     5
Growth increment for Index file (sectors)          5
Number of records (valid and deleted)              7
Number of nodes (valid and deleted)                3

Index       Index                 BTree
Number   Specification            Depth
  0      Byte:5.4.ANU.W            1
  1      Character:30.9.AND.W      1
  2      Byte:9.0.ANU.W            1
```

```
Data store file                                    [Win]<Personnel>Employee.ISAM
Index file                                         [Win]<Personnel>Employee.Ind
Last accessed                                      Thu Jul 21, 1983 10:09 AM
Last modified                                      Thu Jul 21, 1983 10:09 AM
Record size (bytes)                                43
Node size (sectors)                                2
Size of Data store file (sectors)                  5
Size of Index file (sectors)                       10
Growth increment for Data store file (sectors)     5
Growth increment for Index file (sectors)          5
Number of records                                  7
Number of deleted records                          0
Number of nodes                                    3
Number of deleted nodes                            0

Index       Index                 BTree    Number      Number of       Average
Number   Specification            Depth  of nodes  records indexed  Fullness (%)
  0      Byte:5.4.ANU.W            1        1          7               7
  1      Character:30.9.AND.W      1        1          7               24
  2      Byte:9.0.ANU.W            1        1          7               9
```

Figure 4-1. Status Reports

SECTION 5
ISAM REORGANIZATION

INTRODUCTION

The ISAM Reorganize utility builds a data set from any Standard Access Method file of fixed-length records, including the data store file of a data set.

You can use this utility by itself to:

- initially load a data set from a Direct Access Method (DAM) file
- change the definition of the indexes of an existing data set or add new indexes
- change CreateISAM parameters, such as B-tree node sizes

You can use this utility with the Maintain File utility to:

- recover records from a data set damaged by a software or hardware failure
- reclaim space in a data set from which you deleted records
- merge several data sets or DAM files into a new data set

You can use this utility with the interactive Sort utility (described in the *B 20 Systems Sort/Merge Reference Manual*) to:

- sort the physical records to organize the data set in order by key values
- recover records, reclaim space, and merge data sets (as with the Maintain File utility)
- modify records by changing, adding, deleting, or rearranging fields

Sorting the records of a data set improves the performance of application systems that access the records of the data set in the sort order. No other difference is visible at the application system level.

ISAM Reorganize performs an in-place reorganization. The DAM file or the data store file of the data set becomes the data store file of the reorganized data set. The index file of the data set is deleted and then rebuilt.

At the end of this section, there are examples of how to use ISAM Reorganize alone or with Maintain File or Sort to perform the tasks described previously.

Invoking ISAM Reorganize

To invoke ISAM Reorganize, you type ISAM Reorganize (or as many letters from the name as are required to make it unique) in the command field and press **RETURN.** You then fill in the various fields in the utility form and press **GO.**

Optional Fields

On the form, optional fields are enclosed in brackets. The default values are included with the field name.

Fields Ending With a Question Mark

Fields ending with a question mark require a response of Yes or No. The default value is No.

Fields Requiring a Password

You use file system passwords to open a data set in administrator mode. All the ISAM utilities require these passwords.

Whenever a password is required, you use the logged-in password unless a password is provided as part of a file specification. To type a password as part of a file specification, you enter the file specification, a caret character (^), and then the password. The combined file specification and password must not contain any embedded spaces. To maintain security, the Executive echoes the characters of the password as number signs (#).

ISAM Reorganize Form

```
ISAM Reorganize
  ISAM data set or DAM file                                  _____
  [Index file]                                               _____
  [Work file 1]                                              _____
  [Work file 2]                                              _____
  [Use parameters from ISAM data set?]                       _____
  [Index keys (e.g., Byte:10.8.ANU.W)]                       _____
  [B-tree node size (2 sectors)]                             _____
  [Data store file growth increment
   (30 sectors)]                                             _____
  [Index file growth increment
    (30 sectors)]                                            _____
  [Minimum index file size
    (30 sectors)]                                            _____
  [Maximum initial B-tree node
    fullness (80%)]                                          _____
  [Overwrite ok?]                                            _____
```

where

ISAM data set or DAM file
: is the file specification for an existing data set, or for a DAM file from which a data set is to be built. If you specify a DAM file, the DAM file becomes the data store file of the new data set. ISAM Reorganize can determine by the content of the data file or an ISAM data set whether the file is a DAM file or an ISAM data set.

[Index file]
: is the file specification for the index file of the reorganized data set.

  If the file specified in ISAM data set or DAM file above is an existing data set, then the index file name defaults to the name of the existing index file for the data set.

  If [Index file] changes the name of the index file of an existing data set, then the old index file is deleted.

If ISAM data set or DAM file above specifies the name of a DAM file, then the index file name defaults to the name of the DAM file with the suffix .Ind .

[Work file 1]
[Work file 2]

are two work files, each approximately cRecords*(sKeyMax + 8) + 512 bytes long, where cRecords is the number of records in the data set and sKeyMax is the length of the longest key field in bytes.

The work files default to [Sys]<$nnn>ISAMWork1tmp and [Sys]<$nnn>ISAMWork2tmp, where nnn is the workstation number. If the directory [Sys]<$nnn> does not exist, then the work files default to ISAMWork1tmp and ISAMWork2tmp in the logged-in directory (using the logged-in default file prefix).

For optimal ISAM Reorganize performance, the work files should be on different Winchester disk drives.

[Use parameters from ISAM data set?]
is Yes or No (the default).

Yes indicates that an existing data set is to be reorganized. If you leave any of the fields in the form from [Index keys (e.g., Byte:10.8.-ANU.W)] through [Index file growth increment (30 sectors)] blank, then the values you specified when you created or last reorganized the data set are used instead of the listed defaults.

If you specify Yes for this field and the index file of the data set is deleted or unusable, then a status message appears on the video display and ISAM Reorganize terminates.

If you specify No for this field, then you must fill in [Index keys (e.g., Byte:10.8.ANU.W)]. If

you do not fill in any of the fields [B-tree node size (2 sectors)], [Data store file growth increment (30 sectors)], or [Index file growth increment (30 sectors)], then the listed default is used.

Index keys (e.g., Byte:10.8.ANU.W)]
is the parameter list that specifies the index fields of the data set. Each parameter in the list is specified in the following format (no embedded spaces):

```
t:l.o.anu.w
```

or

```
t.o.anu.w
```

where

t is the type of the field:

Binary
Byte
Character
Decimal
Display
Integer
LongIEEE
ShortIEEE
ExtendedIEEE
LongReal
ShortReal

l is the length of the field in bytes for byte strings or character strings, the number of digits for decimal, or the number of bytes for binary, display, and integer numbers. If you omit this entry, binary fields default to two bytes. Long, short, and extended IEEE index fields and long and short real index fields do not use this entry.

o is the byte offset in the record for the index field. It is specified as a decimal number.

a is A if the key order for this key is ascending, or D if the key order is descending.

n is N if null values (binary 0's) are to be indexed, and S if null values are suppressed.

u is U if the key is to be unique (if duplicates are not allowed), and D if duplicate keys are to be permitted.

w is W for non-COBOL application systems, and M for COBOL application systems. This field is optional; the default is W.

[B-tree node size (2 sectors)]
is the number of sectors in the B-tree nodes for the new data set. Maximum value is 12 sectors.

[Data store file growth increment (30 sectors)]
[Index file growth increment (30 sectors)]
[Minimum index file size (30 sectors)]
are values that you use to avoid wasting disk space and to avoid disk fragmentation, which is caused by excessive numbers of disk extents.

[Maximum initial B-tree node fullness (80%)]
is the percentage of allocated space that is the maximum capacity for each B-tree node. The minimum capacity for a B-tree node is 50%. If you specify a percentage less than 50%, in this entry, ISAM Reorganize ignores the entry and uses 50%.

[Overwrite ok?]
is Yes or No (the default).

If you specify Yes, then the reorganization is performed.

If (1) you specify NO, and (2) the ISAM data set or DAM file entry specifies a DAM file, and (3) the index file exists, then ISAM Reorganize issues a prompt to

confirm before overwriting the existing contents of the file. You press **GO** to confirm, and **CANCEL** or **FINISH** to deny.

Loading a Data Set

By using the Executive Copy command, you can load a data set from a DAM file. You simply copy the DAM file to the data store file of the data set. You then use ISAM Reorganize to build the indexes for the new data set and to initialize all of the file structures necessary for access to the data set by ISAM.

Example

If the DAM file, Employee.DAM, contains records to load into the Employee data set, Employee.ISAM, you first copy Employee.DAM to Employee.ISAM and then invoke ISAM Reorganize and fill in the forms as follows:

```
Copy
  File from                          Employee.DAM
  File to                            Employee.ISAM
  [Overwrite ok?]                    _____
  [Confirm each?]                    _____

ISAM Reorganize
  ISAM data set or DAM file          Employee.ISAM
  [Index file]                       _____
  [Work file 1]                      _____
  [Work file 2]                      _____
  [Use parameters from
    ISAM data set?]                  _____
  [Index keys
    (e.g., Byte:10.8.ANU.W)]         @Employee.Keys
  [B-tree node size (2 sectors)]     _____
  [Data store file growth
    increment (30 sectors)]          _____
  [Index file growth increment
    (30 sectors)]                    _____
  [Minimum index file size
    (30 sectors)]                    _____
  [Maximum initial B-tree node
    fullness (80%)]                  _____
  [Overwrite ok?]                    _____
```

Employee.Keys is a text file containing the following key specifications:

```
BYTE:5.4.ANU.W
CHARACTER:30.9.AND.W
BYTE:9.0.ANU.W
```

Changing Indexes and Other ISAM Create Parameters

You use ISAM Reorganize to change the index definitions of an existing data set, to add new indexes, or to change other ISAM Create parameters such as B-tree node sizes.

Example

In the Personnel data sets example, the records contained in Employee.ISAM have the following structure:

Offset	Length	Field	Type
0	4	deptNo	byte
4	5	empNo	byte
9	30	empName	character
39	4	salary	decimal

empNo and empName are defined as keys, and there is also a composite key (deptNo,empNo).

In this example, an index on the salary field is added. All four indexes now list records in ascending order by key field(s), and all four now support indexing of null values. The empName and salary indexes allow duplicates, and the empNo and (deptNo,empNo) keys uniquely identify records.

When the ISAM Reorganize form displays, you enter the name of the data set in the ISAM data set or DAM file field and specify Yes for [Use parameters from ISAM data set?]. Then you fill in the parameters you want to change.

```
ISAM Reorganize
  ISAM data set or DAM file          Employee.ISAM
  [Index file]
  [Work file 1]
  [Work file 2]
  [Use parameters from
    ISAM data set?]                  Yes
  [Index keys
    (e.g., Byte:10.8.ANU.W)]         @Employee.Keys
  [B-tree node size (2 sectors)]
  [Data store file growth
    increment (30 sectors)]
  [Index file growth increment
    (30 sectors)]
  [Minimum index file size
    (30 sectors)]
  [Maximum initial B-tree node
    fullness (80%)]
  [Overwrite ok?]
```

Employee.Keys is a text file containing the following key specifications:

```
BYTE:5.4.ANU.W
CHARACTER:30.9.AND.W
BYTE:9.0.ANU.W
DECIMAL:6.AND.W
```

ISAM Reorganize extracts the old parameters from the data set, replaces any that have new values with the values specified (in this case, only the index keys), and rebuilds the indexes of the data set.

Recovering Records, Reclaiming Space, and Merging Data

You use the Executive's Maintain File utility and ISAM Reorganize to:

- recover records from a data set damaged by a software or hardware failure
- reclaim space in a data set from which you deleted records
- merge several data sets or DAM files into a new data set

Maintain File performs each of these tasks and produces a single file. This file, along with ISAM Reorganize, builds the indexes.

To recover data from Employee.ISAM, you first invoke Maintain File and fill in the form as follows:

```
Maintain File
  Input files   Employee.ISAM
  [Output file] Temp.DAM
  [Log file]
  [Remove deleted records?]
  [Suppress confirmation?]
```

Maintain File scans the data store file of the Employee data set, verifying the file structures, recovering data, and reclaiming the space the deleted records occupied. The records are then copied into Temp.DAM.

Next, you use the Executive Copy or Rename command to replace the data store file of Employee.ISAM with the file that Maintain File produces, as follows:

```
Rename
  Old file name Temp.DAM
  New file name Employee.ISAM
  [Overwrite ok?]Yes
  [Confirm each?]
```

Finally, you use ISAM Reorganize to build the indexes of the merged data set. The existing index file can supply the index keys, node sizes, etc. (If the index file of the old Employee.ISAM data set does not have the default name, Employee.Ind, then you must specify its name in [Index file].)

```
ISAM Reorganize
  ISAM data set or DAM file          Employee.ISAM
  [Index file]
  [Work file 1]
  [Work file 2]
  [Use parameters from
    ISAM data set?]                  Yes
  [Index keys
    (e.g., Byte:10.8.ANU.W)]
  [B-tree node size (2 sectors)]
  [Data store file growth
    increment (30 sectors)]
  [Index file growth increment
    (30 sectors)]
  [Minimum data store
    file size (30 sectors)]
  [Minimum index file size
    (30 sectors)]
  [Maximum initial B-tree node
    fullness (80%)]
  [Overwrite ok?]
```

Sorting Data Set Records

You use the Sort utility and ISAM Reorganize to:

- sort the physical records to organize the data set in order by key values
- recover records, reclaim space, and merge data sets (as with the Maintain File utility previously discussed)
- modify records by changing, adding, deleting, or rearranging fields

You modify records by using the "own code" feature in the Sort utility. After the data store file is sorted by Sort, ISAM Reorganize is used to build the indexes for the data set. See the *B 20 Systems Sort/Merge Reference Manual* for more information.

In the Personnel data sets example, the Employee data set, Employee.ISAM, has the following fields:

Offset	Length	Field	Type
0	4	deptNo	byte
4	5	empNo	byte
9	30	empName	character
39	4	salary	decimal

empNo and empName are defined as key fields, and there is also a composite key (deptNo,empNo).

In this example, a major application system processes records in order by employee number, so it is advantageous to sort the data set by the empNo field. The first step is to invoke Sort and fill in the form as follows:

```
Sort
  Input files                 Employee.ISAM
  Output files                Temp.DAM
  Keys                        BYTE:5.4.A.W
  [Stable sort?]
  [Work file 1]
  [Work file 2]
  [Log file]
  [Suppress confirmation?]
```

Next, you use the Executive **COPY** or **RENAME** command to replace the data store file of Employee.ISAM with the file produced by Sort, as follows:

```
Rename
  Old file name   Temp.DAM
  New file name   Employee.ISAM
  [Overwrite ok?] Yes
  [Confirm each?]
```

Finally, you use ISAM Reorganize to build the indexes of the sorted data set. The existing index file can supply the index keys, node sizes, etc. (If the index file of the old Employee.ISAM data set does not have the default name, Employee.Ind, then you must specify its name in [Index file].)

```
ISAM Reorganize
  ISAM data set or DAM file          Employee.ISAM
  [Index file]
  [Work file 1]
  [Work file 2]
  [Use parameters from
    ISAM data set?]                  Yes
  [Index keys
    (e.g., Byte:10.8.ANU.W)]
  [B-tree node size (2 sectors)]
  [Data store file growth
    increment (30 sectors)]
  [Index file growth increment
    (30 sectors)]
  [Minimum index file size
    (30 sectors)]
  [Maximum initial B-tree node
    fullness (80%)]
  [Overwrite ok?]
```

SECTION 6
ISAM SERVER INSTALLATION

ISAM supports both multiuser and single-user access to the ISAM server. Multiuser ISAM is installed in memory as a system service, while an application system loads single-user ISAM as a task. If multiuser ISAM is installed, application systems can share access to data sets. An application system that opens a data set with single-user ISAM, however, has exclusive use of the data set; no other application system can access that data set.

ISAM is designed to handle the differences between multiuser and single-user access internally. Once access to ISAM is established, application systems are independent of any differences and can run in either environment. (See table 6-1 for more information.) For example, transaction-related constraints associated with multiuser access are also enforced in single-user ISAM. Using the same requirements regardless of access method enables an application system to be written and tested in a single-user environment before being used in a multiuser environment.

Similarly, asynchronous requests from application systems to ISAM use the same procedure, ISAMRequest, in both multiuser and single-user environments.

To support multiuser (shared) access to data sets, you must install the ISAM server in memory as a system service at a standalone or master workstation. You use the ISAM Install utility to install multiuser ISAM. A default configuration file is provided with the ISAM distribution diskettes. You can modify this configuration file to support the requirements of your installation.

You can also load ISAM as a task of an application system by invoking the LoadSingleUserISAM operation. Loading the ISAM server from an application system allows single-user (exclusive) access to data sets located at the workstation where the application is running, without installing ISAM as a system service.

You can install two different forms of the ISAM server: a resident server and a swapping server.

Both forms are included on the distribution diskettes, and the type of system environment in which you are using ISAM determines which form to use. The resident server uses more memory than the swapping form, but the performance is better. Only the resident server can be used on systems equipped with floppy disk drives.

Table 6-1. Differences Between Multiuser and Single-User Access

Multiuser Access	Single-User Access
ISAM is installed as a system service at the master or stand-alone workstation.	ISAM is loaded as a task of the application system.
ISAM services requests from all application systems in the cluster.	ISAM services requests only from the application system that called it.
Both shared and exclusive access to a data set are supported.	Only exclusive access to a data set is supported; no other application system can access a data set that an application system opened.
ISAM uses Request and Respond operations for asynchronous processing.	ISAM uses the Send operation for asynchronous processing.

MULTIUSER INSTALLATION

The following information pertains to multiuser installation in a single-partition BTOS, and in a multipartition BTOS.

Single-Partition BTOS

In a single-partition BTOS, ISAM is permanently installed in memory. Once you install it, you cannot remove it nor reallocate its memory unless you rebootstrap the operating system.

Multipartition BTOS

In a multipartition BTOS, ISAM is installed in a secondary application partition.

ISAM INSTALL

The ISAM Install command installs the ISAM server in memory at a standalone or master workstation. It uses the values contained in a configuration file to determine how memory is allocated.

ISAM Install Form

```
ISAM Install
  [Number of ISAM users (default from OS configuration)]  _______
  [Configuration file (default [Sys]<Sys>ISAM.Config)]    _______
```

where

[Number of ISAM users (default from OS configuration)]
specifies the average number of users who will use ISAM. The default number is the total number of users specified in the system build configuration file that BTOS uses. ISAM determines memory requirements for the specified number of users according to the values specified in the configuration file for the data set.

For example, if a cluster has five users, three of whom use ISAM extensively and two of whom use ISAM only occasionally, then you should specify four. Simultaneous use of ISAM by all five users is allowed, but the users may notice some performance degradation.

[Configuration file (default [Sys]<Sys>ISAM.Config)] specifies the configuration file ISAM is to use to determine memory requirements. The default value is [Sys]<Sys>ISAM.Config, which is the configuration file provided with the distribution diskettes.

Memory Allocation

ISAM server installation requires memory for:

- resident code and data
- a virtual code segment management buffer (swap zone) when the swapping server is used
- a "heap"
- data and index buffers

Resident Code and Data

Following are the code and data requirements for the swapping and resident servers. You can specify the other memory areas, although default values are contained in the distribution configuration file:

Resident code and data
(swapping system service): 37.5K

Resident code and data
(resident system service): 76.5K

Swap Zone

For the resident ISAM server, the swap zone size is always 0. For the swapping form, the size of the swap zone can vary. In general, the more memory allocated for the virtual code segment, the less swapping is performed and the better ISAM performs. For the swapping form of the ISAM server, the size of the swap zone (virtual code segment management buffer) must be at least 8K of memory; the maximum amount is 48K of memory.

Heap

Control blocks for open data sets, indexes, and record locks are allocated from the heap, an area of memory containing internal ISAM data structures. If you need a large number of control blocks, you should increase the heap size. However, the size of the heap can not exceed 40K of memory. Further information on determining heap size is provided in this section.

Data Buffers

Data buffers are fixed-length I/O buffers into which portions of data files are read. Data buffers are allocated as 512-byte sectors. A data buffer must be at least two sectors (1K) and not more than 127 sectors (63.5K). The buffer size should be kept small unless the records in the data set are large. The total memory area allocated for the data buffers is determined by multiplying the number of sectors needed for a buffer by the number of buffers needed. The number of buffers needed depends on the number of users. Refer to ISAM Configure in this section for information on determining how many buffers are needed. Memory Allocation Calculation in this section provides a more detailed explanation of how the data buffer size is determined.

Index buffers are fixed-length I/O buffers into which portions of index files are read. Index buffers are allocated as 512-byte sectors. An index buffer must be at least one sector (512 bytes) and no more than 12 sectors (6K). You cannot access a data set if the largest B-tree node in the index file does not fit in the index buffer. Nodes, like index buffers, are allocated in full sectors, so the index buffer must be no smaller than the largest B-tree node. The total memory area allocated for the index buffers is determined by multiplying the number of sectors needed for a buffer by the number of buffers needed. The number of buffers needed depends on the number of users. Refer to ISAM Configure in this section for information on determining how many buffers are needed.

ISAM CONFIGURE

An ISAM configuration file specifies the sizes of the ISAM server's memory areas, according to the number of users for which the server is installed. The configuration file provided on the distribution diskette has reasonable default values for these memory areas. However, changing the values with the ISAM Configure utility may improve performance, depending on the typical patterns of access in a particular installation.

The ISAM Configure utility creates or changes the configuration file that the ISAM Install command uses to determine memory allocation. (See ISAM Install discussed previously.) ISAM Configure displays the values contained in the specified configuration file and allows the user to change the values. (See figure 6-1 for more information).

If the specified configuration file does not exist, ISAM Configure creates a new configuration file with the specified name, and inserts default values. The following message is displayed before ISAM Configure creates the configuration file (file is the specified configuration file):

```
File file does not exist.  Create?
(Press GO to confirm, CANCEL to stop command)
```

If the specified configuration file exists but is not an ISAM configuration file, ISAM Configure overwrites the contents of the configuration file with default configuration values. The following message is displayed before ISAM Configure overwrites the contents of the specified configuration file:

```
File is not an ISAM configuration file.
Overwrite?
(Press GO to confirm, CANCEL to stop command)
```

By changing the values in the displayed form, you can modify the configuration file values. Detailed information on calculating the size of the memory areas is provided in the Memory Allocation Calculation in this section.

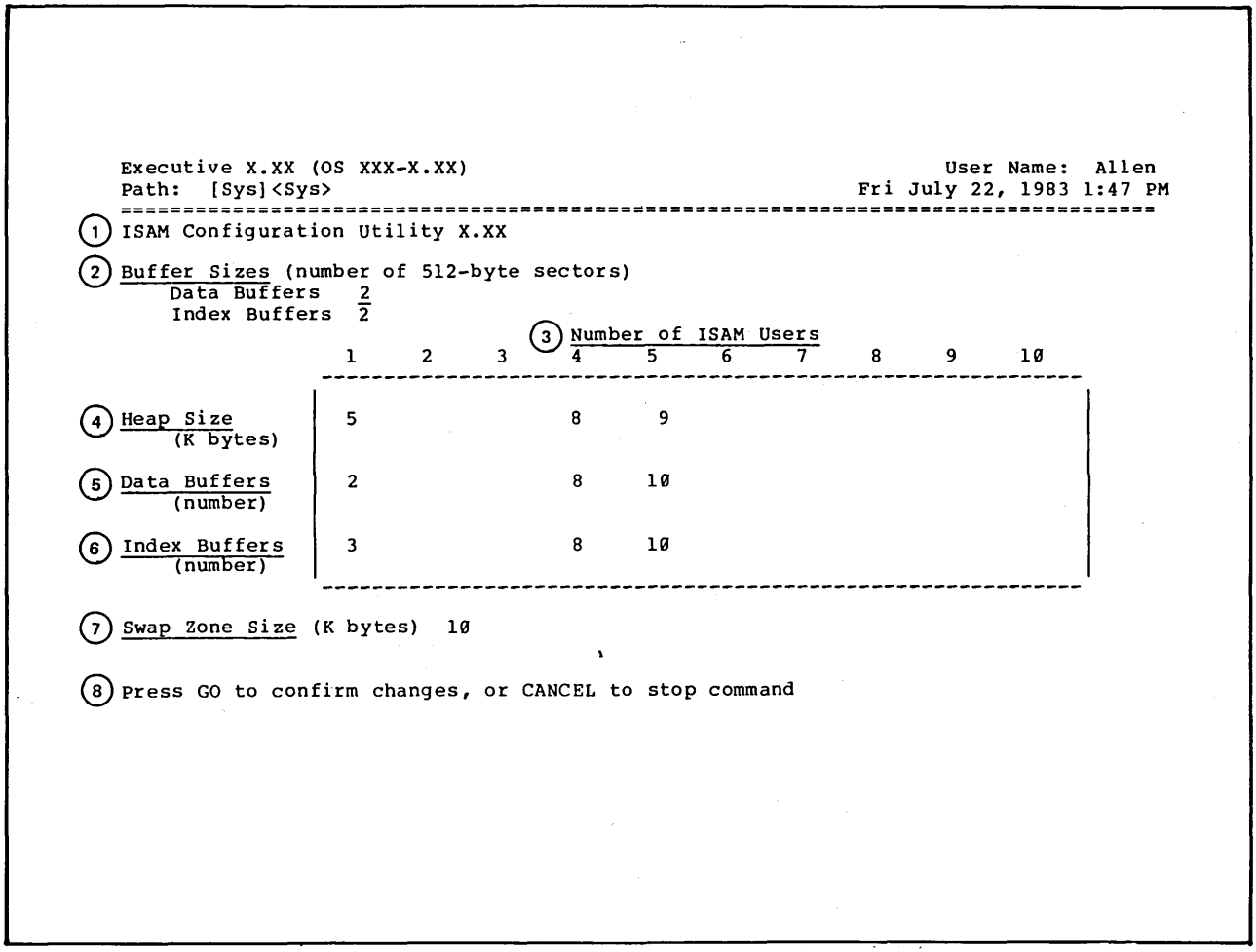

Figure 6-1. ISAM Configure Display

ISAM Configure Form

```
ISAM Configure
  [Configuration file (default [Sys]<Sys>ISAM.Config)]_____
```

where

[Configuration file (default [Sys]<Sys>ISAM.Config)] specifies the configuration file to be created or changed. The default value is [Sys]<Sys>ISAM.Config, which is the configuration file provided with the distribution diskettes.

ISAM Configure Display

The ISAM Configure display is generated when you invoke ISAM Configure. The values from the configuration file specified in ISAM Configure are displayed as part of the form.

By changing the entries in the form, you can change the configuration files.

Cursor Movement

When the ISAM Configure form is first displayed, the cursor is in the first field. Pressing the **NEXT**, **RETURN**, or **TAB** key moves the cursor from field to field. When the last field is reached, pressing the **NEXT** or **RETURN** key brings the cursor to the first field.

The cursor-control keys move the cursor vertically or horizontally. The ← and → keys move the cursor within a field. **SHIFT-** ← moves the cursor to the previous field, and **SHIFT-** → moves the cursor to the next field (the same as the **NEXT** and **RETURN** keys). The ↑ , **SHIFT-**↑ , ↓ , and **SHIFT-**↓ keys move the cursor vertically. **CODE-** ↑ moves the cursor to the first field. **CODE-** ↓ moves it to the last field.

The **DELETE** key deletes one character at a time in a field. The **CODE-DELETE** key deletes all the characters in a field.

The following items describe the parts of the ISAM Configure display. They are keyed to the circled numbers in figure 6-1.

1. **ISAM Configuration Utility X.XX** signifies this form is the ISAM Configuration utility display. X.XX is the version.

2. **Buffer Sizes (number of 512-byte sectors)** specifies the number of 512-byte sectors to be allocated for each data and index buffer. The maximum number of sectors for data buffers is 127, and for index buffers the maximum is 12. The default value for both data and index buffers is 2.

3. **Number of ISAM Users** specifies the range of numbers to be used as column headings for the tabular portion of the form. Each entry is the number of users for which that column provides configuration information. Up to 10 numbers can be specified; the default values are 1 through 10 consecutively. You can change the range of numbers as long as the numbers ascend from left to right.

4. **Heap Size (K bytes)** specifies the number of bytes to be allocated for the heap, based on the number of users. The default values are 5 for one user, 8 for four users, and 9 for five users. The minimum size is 1. For further information, see Memory Allocation Calculation in this section.

   There must be at least one entry in this row.

5. **Data Buffers (number)** specifies the number of data buffers to be allocated, based on the number of users. The default values are 2 for one user, 8 for four users, and 10 for five users. The minimum number is 2. For further information, see Memory Allocation Calculation in this section.

   There must be at least one entry in this row.

6. **Index Buffers (number)** specifies the number of index buffers to be allocated, based on the number of users. The default values are 3 for one user, 8 for four users, and 10 for five users. The minimum number is 3. For further information, see Memory Allocation Calculation in this section.

   There must be at least one entry in this row.

7. **Swap Zone Size (K bytes)** specifies the number of bytes to be allocated for virtual code segment management. The default is 10k; the minimum is 8k. The resident server ignores this entry.

8. **Message line.** Error messages replace this line if any errors occur.

MEMORY ALLOCATION CALCULATION

Before installing ISAM, ISAM Install allocates memory by calculating the memory requirements for resident code and data, a swap zone (if the swapping server is used), a heap, and data and index buffers. Memory allocation for resident code and data is predetermined; see ISAM Install in this section for further information. Memory allocation for the swap zone is specified in the configuration file. ISAM Install uses the configuration file values listed in the ISAM Configure display (see figure 6-1) to calculate memory allocation for the heap and the data and index buffers, based on the number of users for which ISAM is being installed.

ISAM Install calculates the heap size by selecting the Heap Size entry for the appropriate number of users. If the specified number of users is a column entry, ISAM Install then reads the specified heap size from that column in the Heap Size row. (See figure 6-2.) If ISAM Install does not find that number of users, or if the corresponding column in the Heap Size row is blank, ISAM Install calculates the heap size by either interpolating between or extrapolating from the existing entries.

ISAM Install calculates the memory area needed for the data buffers from two entries in the configuration file. (See figure 6-3.) First, ISAM Install determines the actual number of data buffers to use in the same way it determines heap size, using the Data Buffers (number) entries in the tabular section of the form. ISAM Install then multiplies this number by the Buffer Sizes entry for data buffers. The product is the size of the memory area allocated for all the data buffers. Refer to Buffer Size Guidelines in this section for information on calculating the size of the data buffers.

ISAM Install calculates the memory area the index buffers need in the same way it calculates the area the data buffers need; it uses the Index Buffers entries. (See figure 6-4.)

For example, the default configuration file contains the heap size and the number of data and index buffers for one, four, and five users. If four users are specified during installation (see ISAM Install Command previously discussed), then ISAM Install will allocate 8K bytes of memory for the heap, eight 1024-byte data buffers, and eight 1024-byte index buffers.

If three users had been specified in the previous example, ISAM Install would have interpolated values for the heap size and the number of data and index buffers, then calculated their memory allocation. If seven users had been specified, ISAM Install would have extrapolated the values for the heap size and the number of data and index buffers, then calculated their memory allocation.

BUFFER SIZE GUIDELINES

Whenever a record or B-tree node is read into memory, the entire record or node is read in. Therefore, buffers must be large enough to accommodate the largest record or B-tree node. Data buffers must be large enough to hold the largest record in the data set. This requirement is complicated by the fact that records can overlap sector boundaries, but I/O operations always access whole sectors. If a record overlaps two sectors, both of the sectors must be read into the data buffer.

The rule for allocating data buffers is:

> buffer size (bytes) = record size (bytes) + overhead (8 bytes) + overlap sector (511 bytes), rounded up to full sectors.
>
> If the record size + 8-byte overhead is a power of 2, then the records will align on sector boundaries and the extra 511 bytes for sector overlap is not needed in the calculation.

The number of buffers allocated depends on the number of users. The minimum is two data buffers. Performance improves with more buffers because this increases the probability that a record will still be in memory when it is needed a second time.

```
Executive X.XX (OS XXX-X.XX)                                    User Name:  Allen
Path:  [Sys]<Sys>                                          Fri July 22, 1983 1:47 PM
================================================================================
ISAM Configuration Utility X.XX

Buffer Sizes (number of 512-byte sectors)
    Data Buffers   2
    Index Buffers  2
                                        Number of ISAM Users
                    1     2     3     4     5     6     7     8     9     10
                   ---------------------------------------------------------
                  |
Heap Size         |  5                  8     9
    (K bytes)     |
                  |
Data Buffers      |  2                  8     10
    (number)      |
                  |
Index Buffers     |  3                  8     10
    (number)      |
                   ---------------------------------------------------------

Swap Zone Size (K bytes)  10

Press GO to confirm changes, or CANCEL to stop command
```

Figure 6-2. Rows in the ISAM Configure Display Used to Determine

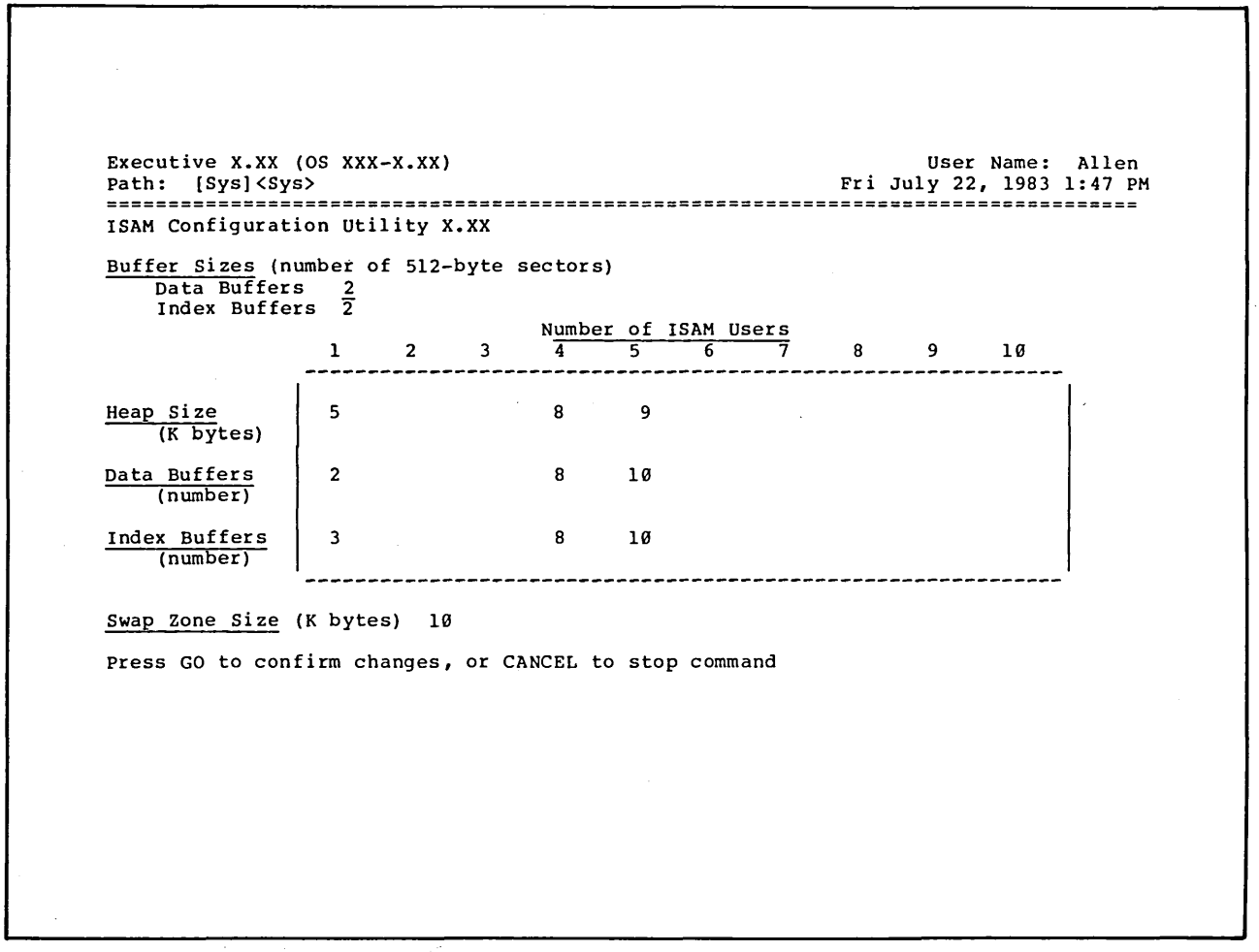

Figure 6-3. Rows in the ISAM Configure Display Used to Determine the Size of the Data Buffers

```
Executive X.XX (OS XXX-X.XX)                                    User Name:  Allen
Path:  [Sys]<Sys>                                        Fri July 22, 1983 1:47 PM
================================================================================
ISAM Configuration Utility X.XX

Buffer Sizes (number of 512-byte sectors)
    Data Buffers   2
    Index Buffers  2

                                  Number of ISAM Users
                    1     2     3     4     5     6     7     8     9     10
                  ---------------------------------------------------------------
Heap Size        |  5                  8     9                                    |
    (K bytes)    |                                                                |
Data Buffers     |  2                  8     10                                   |
    (number)     |                                                                |
Index Buffers    |  3                  8     10                                   |
    (number)     |                                                                |
                  ---------------------------------------------------------------

Swap Zone Size (K bytes)  10

Press GO to confirm changes, or CANCEL to stop command
```

Figure 6-4. Rows in the ISAM Configure Display Used to Determine

APPENDIX A
STATUS CODES

ISAM STATUS CODES (3100 to 3199)

Decimal Value	Hexa-decimal Value	Meaning
3100	0C1C	No such index exists. An operation that includes a key was invoked, but the specified index does not exist.
3101	0C1D	Prefix is not valid. SetUpISAMIterationPrefix was invoked for an index that is neither a byte nor character string.
3102	0C1E	Bad key length. The length of a key is inconsistent with the index type.
3103	0C1F	Bad ISAM or data base handle. The ISAM handle does not identify an open ISAM data set.
3104	0C20	Bad ISAM 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.

ISAM STATUS CODES (3100 to 3199)(Cont)

Decimal Value	Hexa-decimal Value	Meaning
3105	0C21	Bad ISAM header. See 3104
3106	0C22	Internal error.
3107	0C23	ISAM is already installed. This code is generated by the InstallISAM operation or the ISAM Install utility if ISAM is already installed.
3108	0C24	Internal error.
3109	0C25	Internal error.
3110	0C26	Internal error.
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 (256) has been reached.
3112	0C28	Internal error.
3113	0C29	Buffers are too large. The amount of RAM required by either the data store or index cache buffers exceeds a megabyte.
3114	0C2A	Internal error.
3115	0C2B	ISAM terminated abnormally. Following detection of an internal error, all subsequent ISAM operations and services generate this status code.

ISAM STATUS CODES (3100 to 3199)(Cont)

Decimal Value	Hexa-decimal Value	Meaning
3116	0C2C	Internal error.
3117	0C2D	Bad unique record identifier or bad tuple identifier. An incorrect unique record identifier parameter was passed to ISAM.
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.
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.
3120	0C30	Attempted privacy breach. An attempt was made to modify a data set that is open in batch, read or transaction read mode.
3121	0C31	Bad request. Either the request block is invalid, or the parameters of an operation are inconsistent or have invalid values.

ISAM STATUS CODES (3100 to 3199)(Cont)

Decimal Value	Hexa-decimal Value	Meaning
3122	0C32	Data store file error. This is returned as the status code of an ISAM operation. The detail status code refers to a problem with the data store file of the ISAM data set.
3123	0C33	Index to data error. An inconsistency has arisen between the index and data store files of the ISAM data set.
3124	0C34	Record size incorrect. The specified record size is incorrect for the ISAM data set.
3125	0C35	Duplicates allowed. An operation specified a unique key parameter (that is, duplicates are not allowed), but the index allows duplicates.
3126	0C36	No such record or tuple exists. A unique key was used to identify a record, but no record is stored in the ISAM data set with that key.

ISAM STATUS CODES (3100 to 3199)(Cont)

Decimal Value	Hexa-decimal Value	Meaning
3127	0C37	No more records or tuples. 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.
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 is invalid.
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 indexes in the ISAM data set.

ISAM STATUS CODES (3100 to 3199)(Cont)

Decimal Value	Hexa-decimal Value	Meaning
3130	0C3A	Bad ISAM mode. The mode parameter of the OpenISAM operation is invalid.
3131	0C3B	Cannot open ISAM. This is returned as the status code of an OpenISAM operation. The detail status code gives the reason for the failure.
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 12 bytes accepted by the SetISAMProtection operation.
3133	0C3D	Wrong record size. The OpenISAM operation detects the wrong size record.
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.
3135	0C3F	Internal error.
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.

ISAM STATUS CODES (3100 to 3199)(Cont)

Decimal Value	Hexa-decimal Value	Meaning
3137	0C41	Cannot create ISAM. This is returned as the status code of the CreateISAM operation. The detail status code gives the reason for the failure.
3138	0C42	ISAM buffer is too small. The data set being opened or created requires buffers larger than those installed.
3139	0C43	Internal error.
3140	0C44	Small ISAM record. An attempt was made to create an ISAM data set with records shorter than four bytes.

ISAM STATUS CODES (3100 to 3199)(Cont)

Decimal Value	Hexa-decimal Value	Meaning
3141	0C45	Not in transaction. An operation that can be invoked only when the application system is in a transaction was invoked while the application system was not in a transaction.
3142	0C46	Data set or relation is 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.
3143	0C47	Record or tuple is not available. An attempt to read or hold a record failed because the record was held by another user.
3144	0C48	Record or tuple is 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.
3145	0C49	Too many records or tuples are locked. An attempt was made to hold a record when the maximum allowable number of records was already held.

Decimal Value	Hexa-decimal Value	Meaning
3146	0C4A	In transaction. BeginTransaction was invoked during a transaction.
3147	0C4B	Internal error.
3148	0C4C	Transaction purged. The application system finished a transaction while the request was queued. (This status code is generated only if the application system has a request pending when invoking RollBackTransaction or CommitTransaction.)
3149	0C4D	Internal error.
3150	0C4E	Internal error.
3151	0C4F	Data set or relation is not locked. An operation for which the data set must be held was called when the data set was not held.
3152	0C50	ISAM heap is full. The 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.

ISAM STATUS CODES (3100 to 3199)(Cont)

Decimal Value	Hexa-decimal Value	Meaning
3154	0C52	Files are not on same workstation. An attempt was made to either create or rename a data set so that the data store and index files would be on different workstations or to open a data set (in other than administrator mode) that is on a different workstation from the one where the ISAM server is running.
3155	0C53	Bad match kind. A SetUpISAMIterationLimits operation contained an invalid matchKind argument.
3156	0C54	Internal error.
3157	0C55	Transaction barrier after modification. An ISAM transaction barrier operation was called after a modification to the data set or data base in the current transaction, but the SetTransactionParams operation was previously invoked with fBarrierAfterModify set to FALSE (0).

Decimal Value	Hexa-decimal Value	Meaning
3158	0C56	Key is locked. This code is returned as the detail status code when the primary status code is 3143 **(Record or tuple not available)**. It indicates that the record or tuple was not available because of key locking constraints.
3159	0C57	Record is held or tuple is locked. This code is returned as the detail status code when the primary status code is 3143 **(Record or tuple not available)**. It indicates that the record or tuple was not available because another user had locked it.
3160	0C58	Transaction has been rolled back. The application system's current transaction has been rolled back. This status code is returned for every operation by an application system after a lock has been broken, until the application system invokes the RollBackTransaction operation.
3161	0C59	Journal file error. An error has occurred during access to a journal file. See the detail error for more information.

ISAM STATUS CODES (3100 to 3199)(Cont)

Decimal Value	Hexa-decimal Value	Meaning
3162	0C5A	Journal open error. An error has occurred while opening the journal file. See the detail error for more information.
3163	0C5B	Journal close error. An error has occurred while closing the journal file. See the detail error for more information.
3164	0C5C	Journal write error. An error has occurred while writing to the journal file. See the detail error for more information.
3165	0C5D	Internal error. Journal record illegal. An error has occurred while reading the journal file during an attempt to roll back a transaction. A record in the journal file has an incorrect format.
3166-3199	0C5E-0C7F	Internal errors.

ISAM STATUS CODES (3200 to 3399)

Decimal Value	Hexa-decimal Value	Meaning
3200	0C80	Bad key type. The type field of a key specification for Sort/Merge or ISAM is invalid.
3201	0C81	Incorrect key length. The cbKey field of a key specification for a Sort/Merge or ISAM operation does not correspond to the type field of the key specification. (For example, for binary keys, cbKey must be 2.)
3202	0C82	Bad key. A key contained in a record for Sort/Merge or ISAM, or a key described by a parameter of an ISAM operation is not of the correct type. (For example, each digit of a BCD key must be between 0 and 9.)
3203-3299	0C83-0CE3	Internal errors.
3300	0CE4	Not a STAM file. The operation failed because the file did not contain the proper signature.
3301	0CE5	STAM header bad checksum. The operation failed because the checksum computed on the file header did not match the checksum computed when the file was created.
3302	0CE6	Record does not exist. The operation failed because the specified record does not exist.

ISAM STATUS CODES (3200 to 3399)

Decimal Value	Hexa-decimal Value	Meaning
3303	0CE7	Malformed record. The operation failed because data read from the disk contained an inconsistency in the record header and trailer.
3304	0CE8	Not fixed-length record. The operation failed because the access method cannot reference variable-length records.
3305	0CE9	Bad file type. The operation failed because the file cannot be accessed with the specified access method.
3306	0CEA	Bad buffer size. The operation failed because the buffer size was too small or not a multiple of 512.
3307	0CEB	Buffer is not word-aligned. The operation failed because the buffer was not word-aligned.
3308-3399	0CEC-0D47	Internal errors.

ISAM STATUS CODES (3900 to 3999)

Decimal Value	Hexa-decimal Value	Meaning
3900	0F3C	Cannot auto-restart. Because of a problem with the journal file, auto-restart recovery cannot be used to recover a data base (or one or more ISAM data sets). Use DB Recover to recover the data base. Recover ISAM data sets by another mechanism such as restoring archived copies.
3901	0F3D	Bad ISAM configuration file An attempt was made to use a file as an ISAM configuration file, but the file has been damaged, or it was not generated by ISAM Configure.
3902	0F3E	Bad ISAM configuration version An attempt was made to use an ISAM configuration file that was generated by an incompatible version of ISAM Configure. Regenerate the file by using a compatible version.
3903-3999	0F3F-0F9F	Internal errors.

APPENDIX B:
UPWARD COMPATIBILITY SUPPORT

ISAM 4.0 supports the use of application systems written for previous releases of ISAM, even when these application systems use procedures and services that are no longer included in the set of ISAM operations.

SUPERSEDED PROCEDURES

The following eight operations are no longer part of the standard set of ISAM operations:

- EndISAMTransaction
- LockISAM
- PurgeISAMTransaction
- SetupISAMIteration
- SetupISAMIterationKey
- SetupISAMIterationRange
- StartISAMTransaction
- UnlockISAM

All eight operations are supported as object modules only. The request blocks are no longer used. ISAM 4.0 handles each of these operations as follows. See section 3, for descriptions of the current ISAM operations that are mentioned here.

EndISAMTransaction

EndISAMTransaction is equivalent to CommitTransaction.

LockISAM

LockISAM is implemented by BeginTransaction followed by a series of HoldISAMDataSet operations.

PurgeISAMTransaction

PurgeISAMTransaction is equivalent to RollBack-Transaction.

SetupISAMIteration
SetupISAMIterationKey
SetupISAMIterationRange

These operations are implemented by equivalent calls to SetupISAMIterationLimits.

StartISAMTransaction

StartISAMTransaction is implemented by Begin-Transaction followed by a series of HoldISAM-DataSet operations.

UnlockISAM

If the application system uses UnlockISAM in a transaction, then this operation is equivalent to CommitTransaction; otherwise, UnlockISAM has no effect.

InstallISAM SUPPORT

Previous releases of ISAM used two libraries: ISAMSingle.Lib for single-user application systems, and ISAMMulti.Lib for multiuser application systems. ISAMSingle.Lib included the ISAM procedural interfaces and all the implementing modules. The linked run file included all of ISAM. ISAMMulti.Lib included the ISAM procedural interfaces as request interfaces. Requests were sent to the multiuser ISAM service, which is installed at the master workstation or a standalone workstation.

InstallISAM was provided in previous ISAM releases primarily to initialize single-user ISAM. The ISAMMulti.Lib version was provided for compatibility; that is, to enable you to link an application system as either a multiuser or single-user application system. The multiuser version determined whether or not the multiuser server was installed, and returned an error status if it was not installed.

In ISAM 4.0, there is only one library: ISAM.Lib. Single-user application systems now call LoadSingleUserISAM to load the ISAM service as a task. Multiuser applications can call VerifyMultiuserISAM to determine whether or not the multiuser ISAM service is installed.

To provide compatibility for application systems that use InstallISAM, there are two versions of InstallISAM in ISAM.Lib. The module named IsaMin is the multiuser version; it calls Verify-MultiuserISAM. The single-user version, the module IsaSin, calls LoadSingleUserISAM. When an application system that uses InstallISAM is linked, you must specify the appropriate version of InstallISAM in the object modules line of the Link command form. For single-user application systems use:

```
Command  Link
Link
  object modules...ISAM.Lib(IsaSin)...
```

For multiuser application systems use

```
Command  Link
Link
  object modules...ISAM.Lib(IsaMin)...
```

B-TREE NODE SIZES

In ISAM 4.0, the memory requirements for the ISAM server are significantly reduced without degrading performance in most cases. In previous releases, the default B-tree node size was six sectors in ISAM Create, ISAM Reorganize, and the index buffers in ISAM Install. The default B-tree node size in ISAM 4.0 is two sectors.

For existing data sets where the B-tree node sizes were not explicitly set to one or two sectors, you must do one of the following:

- Use ISAM Reorganize to rebuild the index file with 2-sector B-tree nodes or
- Reconfigure ISAM with the ISAM Configure command to use larger index buffers (six sectors).

CONFLICTS AND TIMEOUTS

In previous releases of ISAM, the first request in a transaction to hold a data set or record was queued until the resource became available. Subsequent requests to hold a resource failed with a "data set not available" or "record not available" status message if another user held the resource.

In ISAM 4.0, any request to lock a resource queues only for the amount of time specified in the transaction parameters block. (See section 3 for more information.) The default time is 10 seconds. Either the "data set not available" or "record not available" status message is returned if the resource does not become available within the specified time frame.

In application systems written for previous releases of ISAM, the following rules apply under ISAM 4.0:

- The first request in a transaction to hold a data set or record can fail with a "data set not available" or "record not available" status message.
- Subsequent lock requests are queued.
- Requests cannot queue any longer than the amount of time specified in the transaction parameter block.

APPENDIX C
SOFTWARE REQUIREMENTS

Software requirements are discussed below.

MEMORY

In a cluster environment, ISAM is permanently installed at the master workstation. ISAM requires a minimum RAM memory of 50 kb plus 5 kb for each user.

In a standalone workstation, ISAM requires a minimum RAM memory of 58 bytes.

For configuration requirements, see section 6.

BTOS

ISAM requires Release 4.0 or later of BTOS.

DATA SET LOCATION

In a cluster environment, the files comprising a data set must be located at the master workstation.

ISAM UTILITIES

You can run the ISAM utilities with or without the ISAM server installed.

In a cluster environment, you can run all ISAM utilities at the master workstation or from a cluster workstation. For best performance, however, you should run the ISAM Reorganize, ISAM Copy, and ISAM Status utilities on the master.

ISAM SERVER

When using ISAM from a cluster workstation, you must install the ISAM server at the master workstation. The data files must also be at the master workstation.

The swapping version of the ISAM server cannot be used on standalone workstations that have only floppy disk drives.

APPENDIX D
ESTIMATING INDEX FILE SIZES

INSERTING AND DELETING NODES

B-tree nodes are allowed to fluctuate between 50 percent and 100 percent full. Adding a key to a full node causes the node to split into two 50 percent full nodes. A key is also added to the B-tree node above the full node that has just split. When the full node is the root node, the current root node splits into two 50 percent full nodes, and a new root node with two nodes below it is created. Thus, a new level is introduced in the B-tree.

Removing a key from a 50 percent full node causes the node to be absorbed by its neighbors. Either one or two nodes are used to eliminate the node that is now less than 50 percent full. How the keys are distributed depends on the current size of the neighbor nodes. The result is that one or two nodes become at least two-thirds full.

Whenever a node is eliminated, a key is deleted from the node one level up in the B-tree. If eliminating a node causes the root node to have only one node below it at the next level, the root node is deleted. The next lowest level becomes the root node, and the B-tree has one less level.

INDEX FILE SIZES

The exact size of an index file can be computed only immediately after the indexes are rebuilt. After a series of updates (additions, modifications, deletions), the size can only be estimated.

The estimate is based on:

- the number of tuples in the relation, n
- the definition of the indexes
- the average load factor, f, which is the fraction of each B-tree node that is in use

Whenever the indexes for a relation are rebuilt, the loading factor for the B-tree nodes is 80 percent full. (Rebuilding is done by DB Backup and DB Restore with reorganization, by DB Initialize when using existing data, and by DB Load.) As tuples are stored, modified, and deleted, the portion of the node that is filled varies between 50 percent and 100 percent. The node never falls below 50 percent full, and it is likely to remain near 80percent full most of the time.

Example and Calculations

This example uses a relation with:

- 50,000 tuples
- a 10-byte key
- a 4-byte key
- 6-sector nodes

If the load factor, f, is 80 percent, then the index file size is calculated as follows:

1. the average number of keys per node:

10-byte key: $b = 0.8 * \frac{6*512 - 16}{10 + 4} = 174.6$

4-byte key: $b = 0.8 * \frac{6*512 - 16}{4 + 4} = 305.6$

2. the index sizes:

   10-byte key: $6*50000*(1/174.6 + 2/174.6^2)$
   = 1738 sectors

   4-byte key: $6*50000*(1/305.6 + 2/305.6^2)$
   = 989 sectors

   total size = 2727 sectors

Similar computations for $\underline{f}$ = 50 percent and $\underline{f}$ = 100 percent yield the following:

1. for 50 percent:

   10-byte key: b = 109
   index size = 2803 sectors

   4-byte key: b = 191
   index size = 1587 sectors

   total size = 4390 sectors

2. for 100 percent:

   10-byte key: b = 218
   index size = 1389 sectors

   4-byte key: b = 382
   index size = 789 sectors

   total size = 2178 sectors

APPENDIX E
GLOSSARY OF TERMS

Access Mode
An access mode is the method of opening a data set to read or modify records. The access mode affects the extent to which other application systems can share the data set.

Administrator Mode
Administrator mode is an access mode you use to perform data set-level activities such as deleting, renaming, and setting protection.

Application System
An application system is a task a user invokes to access a data set for a particular application.

Asynchronous Request
An asynchronous request is a method of accessing ISAM system services directly so that data set access and internal computations are overlapped. Asynchronous requests allow application systems to execute more efficiently and rapidly than when a procedural interface is used.

B-Tree
A B-tree is the type of structure used to contain ISAM indexes. A B-tree is usually pictured as an upside down tree, much like a family tree, with a "root" node at the top and "leaf" nodes below the root.

Configuration File
A configuration file specifies the sizes of the ISAM server's memory areas according to the number of users for which the server is installed.

Conflict
Conflict arises when more than one user attempts to lock the same record or data set.

DAM
See Direct Access Method.

Data Buffer
A data buffer is an I/O buffer into which portions of data files are read.

Data Set
A data set contains one type of fixed-length records that are accessed through fixed-length keys.

Data Store File
A data store file is the physical file that holds the records of a data set.

Deadlock
Deadlock, a special case of conflict, occurs when two or more transactions request records or data sets that are already locked by the other transaction. Also see Conflict.

Direct Access Method
The Direct Access Method (DAM) provides random access to disk file records identified by record number.

Exclusive Access
Exclusive access limits the accessibility of a data set or record to a single user. Compare with Shared Access.

File
A file is a set of related records (on a disk) treated as a unit.

Heap
A heap is an area of memory containing internal ISAM data structures.

Index
An index is a structure used to locate particular records within a data set. One index is defined for each key field of a data set. Also see Key and Record.

Index Buffer
An index buffer is an I/O buffer into which B-tree nodes are read. Also see B-Tree and Node.

Index File
An index file holds the indexes for all of the data set's keys.

Indexed Sequential Access Method (ISAM)
The Indexed Sequential Access Method provides random access to fixed-length records identified by multiple keys stored in disk files. Compare with Direct Access Method and Record Sequential Access Method.

ISAM Handle
A data base handle is a word used in ISAM operations to identify an open data set.

Key
Keys are used to access data set records. A key is defined by its position in the record, its length, and type.

Key Type
Key types support the various character and numeric representations used by the Burroughs programming languages and processors.

Locking
Locking is the process of obtaining Exclusive Access to a record or data set in multiuser ISAM.

Node
A node is a portion of a B-tree. It stores index keys.

Password
A password is a text string used to validate an application system or user's access to the data set.

Record
A record is a group of related data fields treated as a unit.

Record Sequential Access Method
The Record Sequential Access Method provides sequential read-only access to the records of a data set.

Reorganization
Reorganization is the process of freeing up space in a data set by removing deleted records and rebuilding the indexes for the data set.

RSAM
See Record Sequential Access Method.

Shared Access
Shared access enables multiple users to simultaneously access the same data set. Compare with Exclusive Access.

Status Block
A status block is a 4-byte memory area that is used to report status codes to an application system.

Swap Zone
A swap zone is a virtual code segment management buffer.

Timeout
Timeouts are used to prevent deadlock. A timeout value specifies the maximum time a request to lock a data set or record can be queued. Also see Deadlock.

Transaction
A transaction is a unit of work. Transactions permit shared access to a data set.

Unique Record Identifier (URI)
A Unique Record Identifier is a 4-byte unsigned integer used to uniquely identify a record in a data set.

Write-Through Cache
The write-through cache is a set of I/O buffers that are used to bring segments of disk records into memory as needed.

INDEX

Documentation Evaluation Form

Title: B 20 Systems Indexed Sequential Access Method (ISAM) Reference Manual

Form No: 1168465

Date: July 1984

Burroughs Corporation is interested in receiving your comments and suggestions, regarding this manual. Comments will be utilized in ensuing revisions to improve this manual.

Please check type of Suggestion:

☐ Addition ☐ Deletion ☐ Revision ☐ Error

Comments:

From:

Name _____

Title _____

Company _____

Address _____

_____

Phone Number _____ Date _____

Remove form and mail to:

Burroughs Corporation
Corporate Documentation – West
1300 John Reed Court
City of Industry, CA 91745
U.S.A.