1-1-1982

The Implementation of a Network Oriented Database Management System Designed to Run Under the Unix Operating System

Mary Dvonch

Rochester Institute of Technology
College of Applied Science and Technology

THE IMPLEMENTATION OF A NETWORK ORIENTED DATABASE MANAGEMENT SYSTEM DESIGNED TO RUN UNDER THE UNIX* OPERATING SYSTEM

A thesis submitted in partial fulfillment of the
Master of Science in Computer Science Degree Program

by

MARY ANN DVONCH

Approved by: Peter H. Lutz
Peter Lutz - Committee Chairman

William Stratton
William Stratton Committee Member

Roy D Czernikowski
Roy Czernikowski Committee Member

Michael J. Lutz
Michael Lutz - Special Technical Advisor

January 1982

*UNIX is a Trademark of Bell Laboratories

Title of Thesis:

THE IMPLEMENTATION OF A NETWORK ORIENTATED DATABASE MANAGEMENT SYSTEM DESIGNED TO RUN UNDER THE UNIX* OPERATING SYSTEM

I, Mary Ann Dvonch, prefer to be contacted each time a request for reproduction is made. I can be reached at the following address:

Computer Science Department
Rochester Institute of Technology
One Lomb Memorial Drive
Rochester, New York 14623

*UNIX is a Trademark of Bell Laboratories

ACKNOWLEDGEMENTS

I would like to thank my thesis advisor, Dr. Peter Lutz, for the endless hours he spent directing my efforts on this project. Without his help and guidance, this project could not have been completed. I would also like to thank Mr. William Stratton for the ideas for this project and for the use of his B-tree programs. I wish to thank Dr. Roy Czernikowski for the time he has spent as a member of my committee. I would like to thank Mr. Michael Lutz for the time he spent as special technical advisor. Last, but certainly not least, I would like to thank my husband, Jerry, and my children for their help and support during the evolution of this project.

ABSTRACT:

THE IMPLEMENTATION OF A NETWORK ORIENTED
DATABASE MANAGEMENT SYSTEM
DESIGNED TO RUN UNDER THE UNIX* OPERATING SYSTEM

We have designed a network approach, low level access system. Our conceptual model is specifically designed to run under the UNIX operating system. Our model has been tooled in the UNIX tradition. We have also supplied a manipulation language for use on the database. Our system does its own input and output buffering.

*UNIX is a Trademark of Bell Laboratories

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

1. INTRODUCTION

Out of the ever growing use of computers to store and manipulate data has grown the need to impose some structure on the data, facilitating the use of the data. To answer to those needs, the concepts of database systems and database management have been developed. A database is a collection of stored operational data used by an application system [Date 77]. A database management system provides the user or users with control of this operational data. The heart of any database system is the data model (or conceptual model). Although there are three well known approaches, relational, hierarchical, and network, we have decided to direct our attention to the network approach for our data model.

In a network approach, an entity is represented by a "record" and the association between entities is represented with a "link". The links are used to group records into sets. Each set type may have any number of occurrences. Each set occurrence has one owner record and any number of member records. However, a specific record type may not be an owner and a member in the same set. A network database system can implement many to many mappings more efficiently than a relational system, since a relational system would require more space to represent the mappings. Also, the network model tends to be more symmetric than the hierarchical model. Although a network model may offer more generality than a hierarchical model and may be more efficiently implemented than a relational model, there is little doubt that a network approach is more difficult to conceptualize [Date 77]. A network model requires an understanding of both record types and links, and their interrelationship. The implementation of many-to-many relationships is not straightforward, often requiring dummy record types. With practice one gets used to this technique. While the network approach to databases may be

conceptually more difficult to understand, we have made every effort to ensure simplicity and clarity of usage.

1.1. DBTG NETWORK MODEL OVERVIEW

A network data model is backed by the Conference on Data Systems Language (CODASYL) Data Base Task Group (DBTG). The reports of DBTG in 1971 and 1976 have influenced the national and international standards for databases. We have been guided in the design of our database system by the DBTG reports. The schema basically consists of the implementation of (1) schema declarations (DDL) and (2) a set of primitives for manipulation (DML). The following paragraphs compare our system with the DBTG proposal.

While the DBTG proposal included dividing the total storage space into a number of "areas", our system has only one area in which all stored data resides. We feel this single area is reasonable in our model since we do not envision the use of the present model as a large scale database.

The DBTG proposal allows the user to specify a membership class by means of an INSERTION/RETENTION entry. The INSERTION entry determines the storage class for a record type while the RETENTION parameter determines a removal class [Date 77]. The DBTG proposal allows AUTOMATIC or MANUAL as a storage class and FIXED, MANDATORY, or OPTIONAL for a removal class, with any combination of storage/removal class for any set type. That is, a given record type may have a different membership class in different set types. MANUAL storage requires that the user insert a member into a set occurrence with a specific "insert member" command. With AUTOMATIC storage, the database management system automatically inserts the member into the appropriate set.

FIXED retention dictates that once a record has been entered as a member in a set occurrence, that member can never have any existence in the database not as a member in that set occurrence. MANDATORY retention is like fixed except that a member is allowed to be moved to a new owner within a set type. OPTIONAL retention allows a member to be removed from a set occurrence without completely removing the member from the database. Given the time limitations of this project, we could not begin to support all of these membership classes. Instead, we chose a specific membership class (MANUAL/MANDATORY - See Chapter 2, Section 1) and implemented it across all record and set types. We have made every effort to tool the model in such a way as to allow future extensions to provide a richer set of membership classes.

DBTG allows several forms of set selection. The set selection clause may be thought of as defining an "access strategy" for the set [Date 77]. Our system places the responsibility for selecting the correct set occurrence on the user. He must supply the name of a set he expects to use. However, all of our routines operate through user established currency pointers, so it would be a relatively easy modification to allow the user the option of supplying the set name or using the current of run set type as a default.

The DBTG data model allows for both ACTUAL and VIRTUAL sources for data items. A VIRTUAL data item is one that is not physically stored in a record, but rather could be produced by applying some function to existing fields. Alternatively, the source of a data item could be declared as ACTUAL, which indicates that the data item actually exists in the database. The data item could be a field or a group of fields. Our network model does not deal in fields, other than to produce keys and to make sure that the number of fields in a record matches the

number of fields in its type definition. It would seem reasonable that the software that provides a data sublanguage to the user might be able to provide some virtual capabilities.

1.2. PROPOSED SYSTEM

Figure 1-1 depicts the proposed system. The user/database system interface includes software to provide a subschema definition language and a stand alone user query language. It is hoped that this software will be provided as a future extension of this system. See Section 5-1 for further information and clarification. The Database Management System presently implemented provides definition directives, which allow the creation of record and set types, and manipulation commands or primitives, which allow the manipulation of the data stored in the database. The definition directives are accepted by the DBMS and the appropriate record or set types are defined. The record and set definitions are maintained in a binary tree in memory while the Database Management System is in use. When the DBMS is not being used, the definitions are stored on a file. See Section 2.4.2 for further information. The manipulation primitives work through an input/output buffering system maintained by the DBMS. See Section 2.4 for further information. The manipulation primitives are discussed in detail in Section 4.

1.3. ASSUMPTIONS AND DESIGN PRIORITIES

Our network database model assumes that any record stored in the database has a unique key. Under our present system, "unique key" is defined as a combination of one to ten fields to produce a record identification which is unique throughout the system. Of course the limit of ten fields is merely a parameter and easily changed. It would also be a minor change to require the key to be

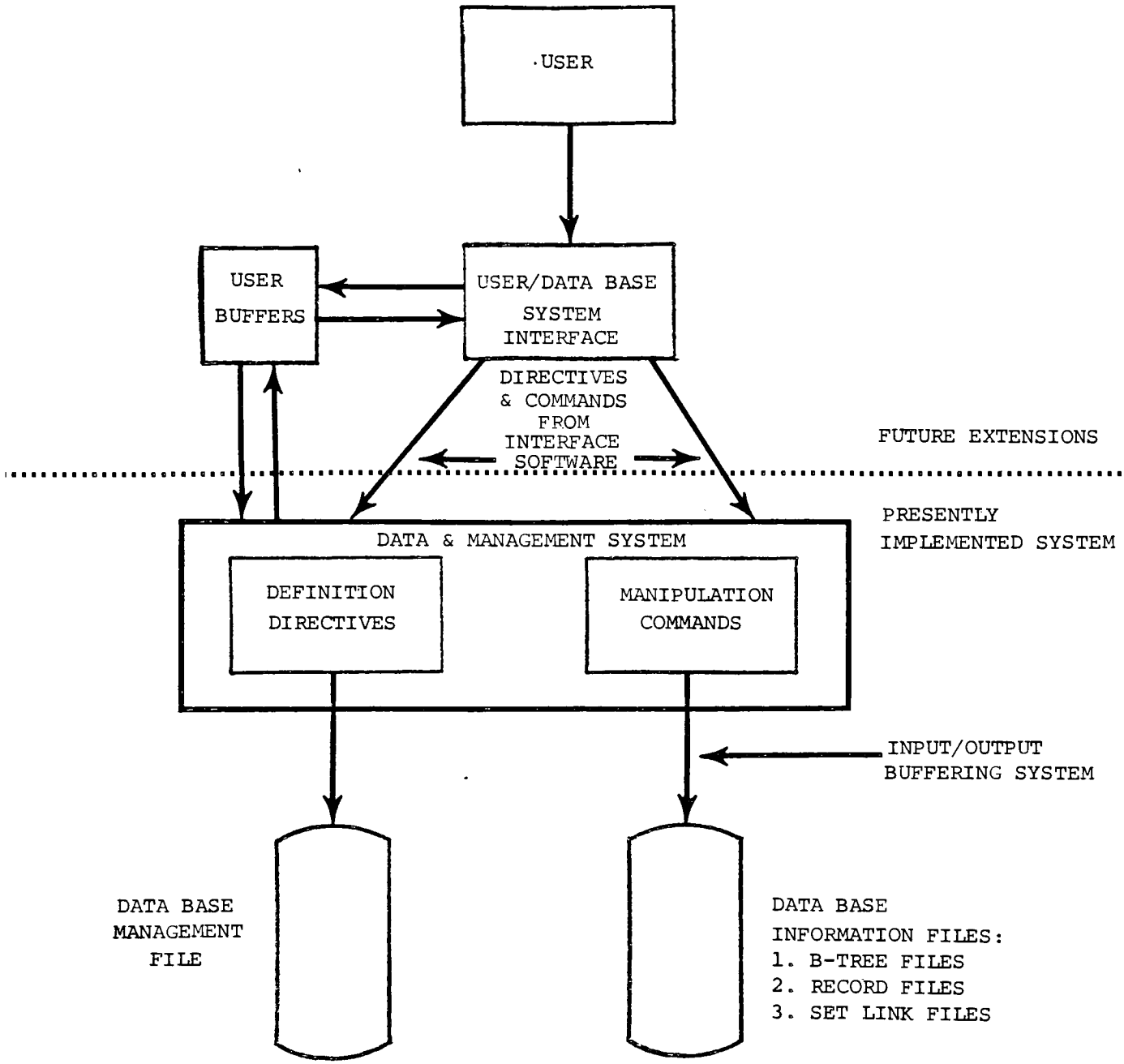


FIGURE 1-1

PROPOSED SYSTEM

unique only within a record type and not throughout the entire system. Null keys are not allowed, nor is any field that is part of a key allowed to be null. We assume that only one user will be using the database at a time, so as to avoid the simultaneous update problem.

In general, the goal of efficient operation of a database system is bought at the cost of loss of flexibility of design, and flexibility of design causes some loss of efficiency of operation [Inmon 81]. In the trade-offs between performance versus flexibility, it is easy to opt for performance because the utility of performance is very apparent and immediately recognizable. On the other hand, the utility gained by building elasticity into a data structure doesn't blossom until a later time. Data that is flexible survives best in the face of environmental change. It is only when data that is inflexible undergoes change that the wisdom of building flexible data structures becomes apparent. Along with data flexibility, or perhaps as an extension of it, we have added the goal of ease of extension. Our conception of this system is not that it is a total database package that will serve the needs of all users. Rather, this system is meant to serve as the kernel of a much larger database system. Thus, our efforts were centered on producing a system that would allow maximum flexibility and ease of extension. Obviously, these were paid for at times at the cost of loss of efficiency in either space or time.

A second and very real priority was the optimization of application development time. We attempted to design and implement a well defined system within the given time frame. Given unlimited time and resources, a larger, more powerful system could have been developed. However, it is hoped that the flexibility built into our system will allow it to meet changing user needs and expectations.

Since a network model is at best intricate, we emphasized ease of modification when designing our data structures and the create, delete, and find functions. One of our major concerns was minimizing pointer chasing. C was most helpful in dealing with the pointer problems, and we feel that the pointer design and implementation is one of the strengths of our system. We also attempted to keep the data structures as straight forward as possible in order to ease understanding. Although network models, by their very nature, are more difficult to understand than relational models, we hope that by emphasizing ease of manipulation in the project, we have also added ease of understanding.

Since optimization of execution time and minimizing data storage were not our main concerns, they have undoubtedly suffered. We did supply our own input/output buffering system in an attempt to lower execution time. We stored pointers to data instead of copies of data whenever it was feasible, in order to minimize data storage. However, if it came to a choice between flexibility and optimized efficiency, we invariably chose flexibility.

Our implementation of a network model has been specifically designed to run under the UNIX operating system. In keeping with the UNIX overall approach, we have designed tools which can be used to define record and set types (DDL), and to create, manipulate and delete record and set occurrences from a database (DML). For our purpose, a "tool" is a specialized function that solves a general problem and is easy to use.

Our tools are written in the C Programming Language. C is well supported under the UNIX system. The UNIX operating system itself is written in C, as is most of the UNIX applications software. C is independent of any particular machine architecture and every effort has been made to make this system

"portable". C's recursive ability has been used throughout our network system, allowing small pieces of code to perform significant tasks.

Our database system handles its own buffering for input and output. The reads and writes through the UNIX operating system are accomplished in page size blocks. Stonebraker [81] compared and contrasted the way operating systems, especially UNIX, handled buffer pool management for database systems. He concluded, "The strategy used by most DBMS's is to maintain a separate cache in user space. This buffer pool is managed by a DBMS specific algorithm." Therefore, we feel that handling our own input/output buffering is an efficiency enhancement to our system.

2. A NETWORK APPROACH DATA MODEL

We have provided a network approach, low level definition and manipulation system for a DBTG like network database, that operates under the UNIX operating system. A manipulation language has been provided to communicate with the storage and retrieval routines. However, it should be understood that the manipulation language is not written for the "user" of the database. It is assumed that another layer of software will take user commands and communicate them to this system. This software will provide a data sublanguage for the user, will parse the user commands, and then use the manipulation language to satisfy the user requests. The following is a list of the definition and manipulation primitives provided by this database management system. The primitives are explained in detail in Chapter 4.

Primitives to create record and set types:
- record type add
- set type add

Primitives to manipulate record and set occurrences:
- add record
- add member
- delete record
- delete member
- delete owner
- find record
- find owner
- find first
- find next
- change owner
- change all
- end update

2.1. MEMBERSHIP CLASS

The membership class for our system is MANUAL/MANDATORY. The MANUAL storage class implies that storing a record, R, in the database, does not

automatically insert R into a set occurrence in which it participates. To insert R into a set occurrence, an explicit "add member" command must be given. Note, however, that if R's record type participates as an owner in any set type, S, R is automatically an owner occurrence in S. The MANDATORY removal class dictates that once an occurrence of a member, say A, has been entered into a set type, say S, as a member in a set occurrence, A can never have any existence in the database not as a member of some occurrence in S [Date 77]. Specifically, A can be moved to a new owner in S with a "change owner" or "change all" command, but may not be removed from S without being removed from every set occurrence in which it participates. Note that the removal of an owner in a set occurrence then causes the recursive removal of all its member records. This removal in the hands of the untutored could create havoc in the database. MANDATORY removal class protects the integrity of the data, in that once a record is removed all traces of it are deleted.

2.2. MODEL ENTITIES

This model uses two different entity types to represent the information contained in the database. Record types are used to represent stored user data and set types are used to store relationships between record types.

2.2.1. RECORD TYPES

Record types represent occurrences of some entity that the user expects to store and manipulate. Before a user can store a record of a specific type in the database, he must define that record type for the database management system. A record definition consists of a record type name, a field delimiter used for that record type, a specification of the number of fields the record type will contain, a

specification of the number of fields that will make up the key for the record type, and the location of the key fields in the record type. The system creates the record type by inserting the record type name with ".rf" appended to it into the binary tree containing record type definitions. The user is allowed to use variable length records within the same record type.

Once the user has defined a record type, the system will store the information about that type for future use. The information is stored in a table (implemented as a binary tree), which is used by the Database Management System. The next time the system is used, it will recall all previous record type definitions. When the user then issues an "add record" command for a particular record type, the database system fills an input/output buffer with the desired record, checks the field count on the record for an error, and builds the record's key. The key is also error checked for duplicity. The record is then passed to the input/output buffering system where it is stored at the end of a file whose name is the record type name with the suffix ".rf". As the record is passed to the buffering system, the location of the record within the file is determined. The location of a record is the block offset from the beginning of the file and then an offset to the position within the block. Once the address of the record within the file is known, the address information and the record key is stored in the record type B-tree. If the record type participates as an owner in any set type, the new record is added as an owner occurrence each such set.

In an effort to keep these records "clean", we have stored them in regular flat files in normal UNIX tradition. "Clean" means that no database pointers, id numbers, or other database management information is stored in the user record files. This allows the stored database files to be used with any other UNIX tools.

For example, a record file could be printed directly. This separation of record files from database information is one of the strengths of our system, for it is in keeping with the UNIX idea of software tools. However, the user should remember that deleted records still remain on the record files until garbage collection is done.

2.2.2. SET TYPES

Set type occurrences are represented by links stored in set type information records. The set type information allows the user to walk around the links that associate a particular owner with members in his set. Such set type information is maintained separately from record type information.

Before a user can store any set occurrences for a set type, he must define the set type. This is done with a "set add" command, followed by the set type name, the owner record type name, and the member record type name. Once the set type is defined, the database management system stores the set type information in a table along with the currency information for that set type. The set type definition must be given before any records of the set owner record type have been added to the database. The system creates a set type by inserting the set type name, with a ".sf" appended to it, into the set type table (implemented as a binary tree). Once a set type has been defined in the database, it may not be removed. The next time the database is used, the database management system will recall all previous set type definitions.

After the set type has been defined, owner occurrences are automatically created whenever a record of the owner record type is added to the database. When an owner record occurrence is added to the database, a link-record for the

owner is built (See Section 4.3). This link-record is then given to the i/o buffer system to be added to a file named by the set type name with ".sf" appended to it. As the record is passed to the i/o buffer system, information is gathered on the location of the pointer record within the set type file. This location information and the owner key are stored in the set type B-tree. The link-records are stored as regular UNIX files and, therefore, can be used as arguments for other UNIX tools.

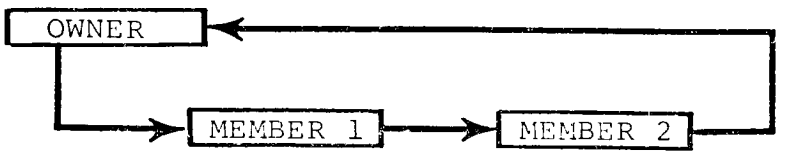
If the user wishes to add a member to a set occurrence, he must give an explicit "add member" command. The user specifies the set type and owner key of the appropriate set occurrence. Inserting the new member into a set occurrence is accomplished by building a link-record for the new member. These link-records make up a chain that connect the set occurrences, and can be traversed to obtain all the members of a set occurrence. (For exact information on link-record insertion, see Section 4.3.) The new member occurrence is added as the immediate successor to the owner. Order of retrieval is therefore LIFO (Figure 2-1).

As with the owner's link-record, the member's link-record are stored through the buffering system, on a file named by the set type name with ".sf" appended to it. Once the link-record is stored, the file offset information along with the new member key is stored in the set type B-tree.

2.3. DATA STRUCTURES EMPLOYED

We have used three major data structures: B-trees, binary trees, and doubly linked lists. Obviously, other data structures were used in the coding of the system, however, they have little impact on the design of the overall system.

ORIGINAL SET:



SET AFTER ADDITION OF MEMBER3:

Figure 2-1
ADDITION OF NEW MEMBER TO SET OCCURRENCE

2.3.1. B-TREE INDEXES

We used the B-tree index [Wirth 76] to store our key and offset information for both the record types and the set types. The B-tree is, de facto, the standard organization for indexes in a database system [Comer 79].

2.3.2. TABLE STORAGE IN BINARY TREES

We used binary trees to store the database management record type and set type tables. Separate trees were maintained for the record type table and the set type table. Each tree contains one node for each record/set type. The nodes are maintained so that at any node, the left subtree contains only record/set types whose name precedes the node's record/set type name with a string comparison; the right subtree contains only types that are greater. Each record type node contains:

the record type name,
the field delimiter for this record type,
the number of fields to expect in the record,
the number of key fields, ie, the number of fields used to build the key,
an array containing the position of the key fields,
a structure of currency pointers,

pointers to the left and right record type binary subtree.

Each set type node contains:

the set type name,
a pointer to the record table entry for the owner record type,
a pointer to the record table entry for the member record type,
the file descriptor and root for the B-tree index for this set type,
current of set type key
a structure of currency pointers,
pointers to the left and right set type binary subtree.

"Currency pointer" structures are used primarily by the input/output buffering routines to place records in the buffers and ultimately into files. The structures contain:

logical file and block offsets,
physical file and block offsets,
final file and block offsets,
a switch that indicates if a record has just been added to the file,
a pointer to the file buffer for this record/set type.

The logical offsets indicate the beginning of the record that was most recently referenced for the record/set type. This supplies the current of run for this record/set type. The physical offsets indicate the end of the record that was most recently referenced for the record/set type. The final offsets indicate the end of the file for the record/set type.

When a new record or set type is defined, a node is created for it in the appropriate binary tree. At the conclusion of the database update, the pertinent information from the binary type trees is stored in a file named dbm_file. (This name is a parameter, and easily changed.) The first field on the dbm_file is the number of record types defined within the database, followed by the number of set types defined. After these come the record type information and then the set type

information. When the database management system is run, the system builds the new binary type trees from the information stored on the dbm_file. At this time the record type B-tree is opened and its file descriptor and root are stored in global storage. Again, the dbm_file is a normal UNIX file that can be examined by and used with other UNIX tools. If no record types or set types have been defined, the system will print a message informing the user of the fact.

2.3.3. EXPRESSING SET RELATIONSHIPS

Our implementation uses doubly linked lists to represent set occurrences. The front pointer of any owner or member in a set occurrence contains the key of the next owner or member in the set occurrence. The back pointer of any member contains the key of the prior member or the owner. The owner's back pointer is always null. These pointers are placed in a link-record, with the forward pointer first and backward pointer second. The link-record is then stored on the set type file, and the owner/member key and the file offset of the link-record are stored in the set type B-tree. As an example, suppose owner A1 owns members B1 and 2B. Then A1's forward pointer would be 2B and backward pointer would be null. B1's fp would be A1 and bp would be 2B. 2B's fp would be B1 and bp would be A1. In order to follow the link to the next owner or member link, the key is found in the set type B-tree, and the offset for the next link-record is obtained. When a member is deleted from a set, the previous and following member's links are reset to exclude the member, and the deleted member's key is removed from the set type B-tree. The member is also removed from the record B-tree and is then removed from any other set where the member participates as an owner or member. When an owner is deleted from the set, the owner's key is removed from the set B-tree, then removed from the record B-tree, and finally removed from any

other set where the owner participates as an owner or member. When an owner is deleted from a set, each of his members is also deleted.

2.4. INPUT/OUTPUT BUFFERING

As a time efficiency measure, we have implemented buffered input and output under the control of the database management system. Many DBMSs, including INGRES and SYSTEM R, have chosen to put a DBMS managed buffer pool in user space to reduce overhead [Stonebraker 81]. Either at the beginning of the update session for predefined types, or at the time a new record type or set type is defined, a page-size block of memory is allocated for its i/o buffering. A pointer to the buffer block is stored in the set/record type table for use by the i/o system. All reads and writes are accomplished in page-size blocks. When the i/o system is asked by the calling routines to store a record, the i/o system expects to find the record in a sending buffer. The i/o routines will then add the record to the end of the appropriate file and return the location of the record to the calling routine. This is accomplished in the buffer area for the record/set type and does not always require actual writes to the file. When the i/o system is requested by the data model to retrieve a record, the i/o system will try to fill the order from data already resident in main store in the buffer area for that record/set type. This will be possible frequently, especially when link-records are sought. When the i/o system locates the appropriate record, it is placed in a receiving buffer supplied by the calling routine. The i/o system updates the currency pointers for record/set types as it does its storage and retrieval functions.

2.5. ERROR HANDLING

All error handling is done through two routines. One error routine accepts a single string as the error message, prints the message, and returns an error indicator. The other error routine accepts an error message and a key value. Both the message and the key value is printed and an error indicator is returned. It would be very easy to change these routines so that no error message was printed, but rather a specialized code would be returned.

3. PROTOTYPE

We built a simple prototype database to be manipulated with our database management system. The prototype consists of four record types and three set types with certain relationships expressed through set occurrences.

3.1. RECORD TYPE DESCRIPTIONS AND CONTENTS

Record Type Name: faculty
Field Delimiter: *
Number of Fields: 5
Number of Key Fields: 1
Position of Key Fields: 2

Contents:

Peter*A1*10*A186*25
Bill*A2*10*2132*57
Roy*3A*10*A285*72
Jack*4A*10*1116*13

Record Type Name: student
Field Delimiter: :
Number of Fields: 4
Number of Key Fields: 1
Position of Key Fields: 3

Contents:

Mary:CAST:B1:Comp Scie
Leslie:CAST:B2:Comp Scie
John:SP:3B:PPPD
Tom:CAST:4B:Syst Soft
Mary:SP:5B:PPPD

Record Type Name: housing
Field Delimiter: *
Number of Fields: 3
Number of Key Fields: 1
Position of Key Fields: 1

Contents:

405*Billings*25
216*Watson*1105

Record Type Name: courses
Field Delimiter: *
Number of Fields: 8
Number of Key Fields: 4
Position of Key Fields: 5 1 3 4

Contents:

B1*0601*81*1*875*1*D*r
B2*0601*81*2*875*1*A*nr
B1*0601*81*1*720*2*B*nr
3B*0532*81*2*875*1*A*nr
B2*0532*81*2*850*1*B*r
B1*0601*81*2*875*1*B*r
5B*0601*80*2*875*1*D*r
5B*0601*81*3*875*1*B*r

3.2. SET TYPE DESCRIPTIONS AND CONTENTS

Set Type Name: fs
Owner Record Type: faculty
Member Record Type: student
Relationship: the faculty member is the advisor of the student

Contents:

A1 owns
  B1
  3B
A2 owns
  B2
  4B
3A owns
  5B
4A has no members

Set Type Name: sc
Owner Record Type: student
Member Record Type: courses
Relationship: the student has taken the course

Contents:

B1 owns
  875B1811
  720B1812
  875B1812
B2 owns
  875B2812
  850B2812
3B owns
  8753B812
4B has no members
5B owns
  8755B802
  8755B813

Set Type Name: hs

Owner Record Type: housing
Member Record Type: student
Relationship: the student is a resident of the housing complex

Contents:

405 owns
B1
5B
216 owns
3B

See Figures 3-1 and 3-2 for a graphic representation of these relationships.

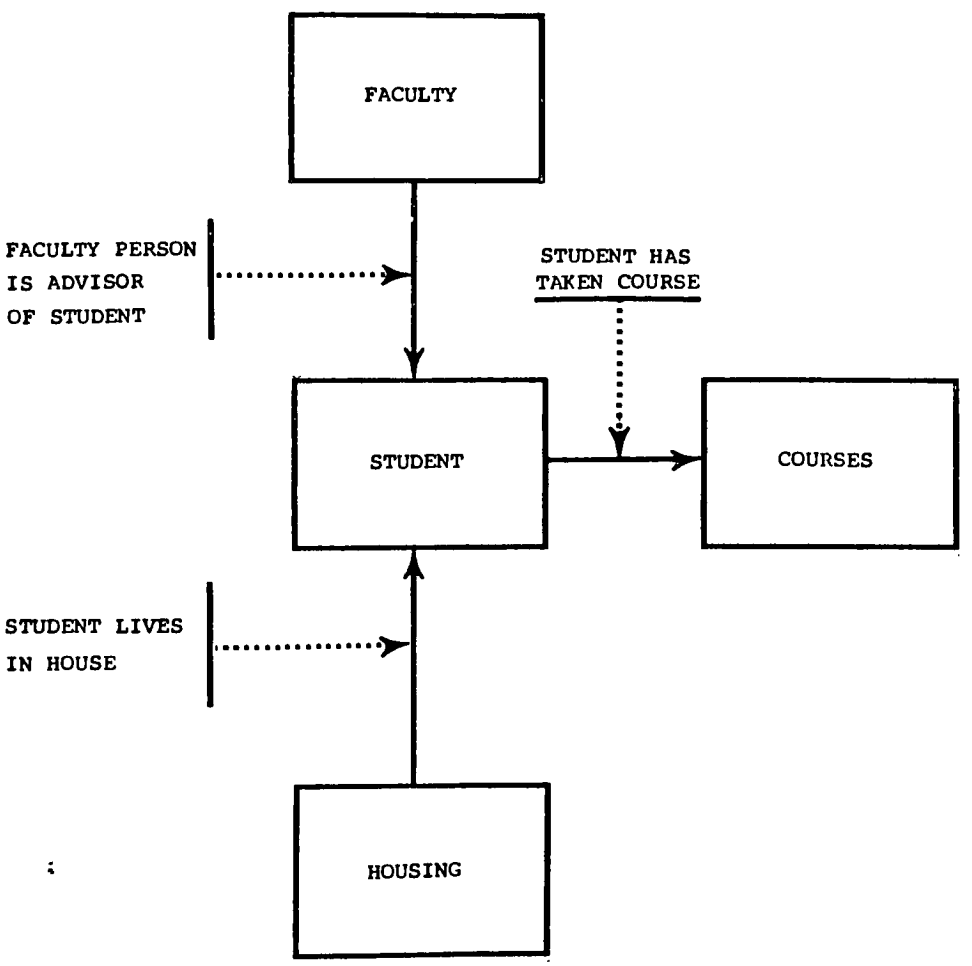


FIGURE 3-1

PROTOTYPE SET RELATIONSHIPS

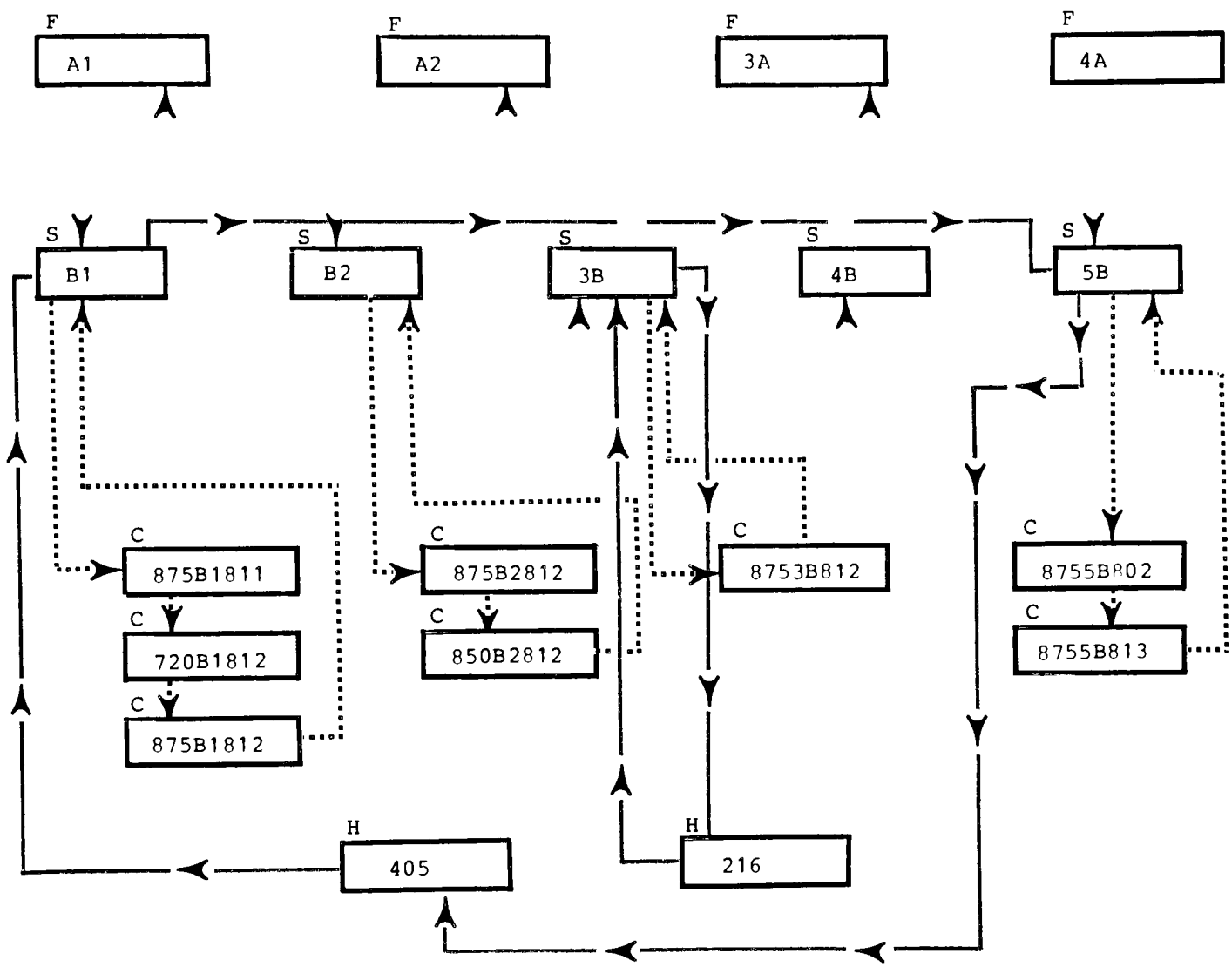


FIGURE 3-2

PROTOTYPE SET OCCURRENCES

4. AVAILABLE PRIMITIVES

We have made every attempt to "tool" our primitives in the UNIX tradition. We have tried to develop tools that are easy to use, work well within the overall UNIX environment, and efficiently accomplish the task for which they are designed. The manipulation language was designed to be terse, as most UNIX commands are. However, in most cases the user is allowed to be more verbose. As an example, the "ra" or record add command, is actually parsed by the leading "r". Anything else can follow within the word; thus "r", "ra", "recadd", and "recordadd" would all be accepted as a record add command.

All record type names and set type names consist of the first ten characters of the word supplied. The number ten exists as a parameter in the system and is easily changed. (UNIX allows eleven characters.) As an example, if the name RochesterInstitute was input as the name of a record type, the record type would be known to the system as RochesterI.

The maximum number of key fields for a record type in the system is set at ten, with the maximum number of characters in a key set at twenty. Again, these numbers are parameters and easily changed. If more than the maximum number of characters are entered for a key value, an error message is printed and the command is aborted. As an example, if A12345678901234567890 was entered as a key value in a command, an error message would be printed.

Some of the commands are allowed to include optional input or output file names. The file name, which can be any UNIX pathname, may be up to forty characters long; extra characters are ignored. Note: all examples in the following tool descriptions will refer to the prototype described in Chapter 3.

4.1. DATABASE DEFINITION DIRECTIVES

The database definition primitives deal with defining the record types and set types.

4.1.1. RECORD TYPE ADD

The record type add command describes a record type for the database management system. The DBMS then stores the information as a node in the binary record type tree. The call to this tool has the form:

ra <name> <field delimiter> <# of fields> <# of key fields> <position of kf>

A record type add command for courses would look like:

ra courses * 8 4 5 1 3 4

4.1.2. SET TYPE ADD

The set type add tool allows the user to define a set type to the database management system. The DBMS then stores the information as a node in the binary set type tree.

The call to the set type add tool has the form:

sa <name> <owner record type name> <member record type name>

The set type fs would have been defined by the command:

sa fs faculty student

4.2. DATABASE MANIPULATION PRIMITIVES

The database manipulation primitives allow the user to add items to the database, remove items from the database, find items within the database, and reorganize members within set types.

4.2.1. DATABASE EXPANSION

When an owner or a member is added to a set, a link-record is built and integrated into the appropriate set. Link records consist of back and forward pointers and are discussed in detail in Chapter 2, Section 4.3. Once the link-record is built and added to the file for the set type, the key for the owner/member along with the file offset of the link-record is stored in the set type B-tree.

4.2.1.1. ADD RECORD

The add record tool allows the user to store a record for a particular record type in the database. If that record type is the owner in any set, another tool, "add owner", is called and the new record key is automatically added as an owner in the set. The call to the add record tool has the form:

```
ar <record type name> [<input file name>]
```

If the input file name is supplied, all the records on the file are added to the database. Otherwise, records are taken from the standard input until an EOF is read. An add record command for a faculty record would look like:

```
ar faculty
Peter*A1*10*A186*25
EOF
```

4.2.1.2. ADD OWNER

If a newly added record is an owner in a set type, the system will automatically generate an add owner call. The add owner tool call has the form:

ao <set type> <owner key>

For the above add record to faculty, a generated add owner tool call would look like:

ao fs A1

4.2.1.3. ADD MEMBER

The add member tool adds a member key to a set occurrence. The system will check the member record to make sure it conforms to set and record definitions. The call to the add member tool has the form:

am <member key> <set type name> <owner key>

A command to add student B1 to faculty A1 would be:

am B1 fs A1

4.2.2. DATABASE CONTRACTION

The database management system's deletion tools are a recursive delight. Since deleting a record implies possible deletion of set occurrences and vice versa, recursion is required. Each deletion tool was designed to do its own particular job and then call a friend to do the rest.

4.2.2.1. DELETE RECORD

When a record is deleted from the database, the record key and file offset are removed from the record type B-tree. Since the offsets for the actual record can no longer be referenced, the record is logically deleted from the database. However, the actual physical record will still remain on the record type file. This record then becomes the province of the garbage collection system which is discussed in detail in Chapter 5. The call to the delete record tool has the form:

dr <record type> <key>

A delete record command to delete student B1's record would consist of:

dr student B1

Once the record is logically deleted, delete record recursively searches the set type binary tree looking for sets in which this record type participates as an owner or member. If a set is found, delete owner or delete member is called to remove the discarded record key from the set. For instance, the "dr student B1" command above would cause B1 to be removed from the fs set and the hs set as a member and the sc set as an owner. The removal from the sc set has further ramifications which are covered in detain under "delete owner".

4.2.2.2. DELETE MEMBER

The delete member tool causes a member's key and link-record offset to be removed from the set type B-tree. The pointers in the link-records of prior and post set occurrence members are reset to exclude the deleted member. The actual link-record is left on the set type file, requiring garbage collection at a later time. However, since this link-record can no longer be referenced, it is effectively

deleted from the database. The call to the delete member tool has the form:

dm <set type> <key>

A delete member command for student B1 in the fs set would consist of:

dm fs B1

First, B1's key and link-record file offset would be deleted from the fs set type, then B1's record would be removed from the student record file, then B1 would be removed as a member from the hs set, and finally, B1 would be removed as an owner from the sc set, triggering the deletion of three course records.

4.2.2.3. DELETE OWNER

The delete owner tool is probably the most powerful tool in the database management system. Somewhat like the A-bomb, the delete owner command can cause fallout to the farthest reaches of the database. When the delete owner command is given, the owner's key and link-record file offset are removed from the set type B-tree. Next, "delete member" is repeatedly called until all of this owner's members are deleted. Finally, "delete record" is called to delete the owner's record from the record type file. The delete owner command has the form:

do <set type> <owner key>

The command to delete owner A1 in set fs would consist of:

do fs A1

This small, unimpressive looking command would cause A1 to be removed from the fs set. Next, B1 would be removed from the fs set and the hs set and courses

875B1811, 720B1812, and 875B1812 would be removed from the sc set and the courses record file, and B1 would be removed from the student record file. Then, 3B would be removed from the fs and hs set files, courses record 8753B812 would be removed from the sc set and the courses record file, and 3B would be removed from the student record file. Finally, A1 would be removed from the faculty record file. Obviously, when it comes to the deletion tools, looks are very deceiving, and caution is the key word.

4.2.3. DATABASE NAVIGATION

The find tools allow the user to navigate the database set occurrences. With the proper sequence of find commands, the user is able to extract information from the database. If an output file name is specified with the find command, the requested record will be output to the named file. Otherwise, the record is sent to standard output.

4.2.3.1. FIND RECORD

The find record tool will find a record given a specific record type and key. The current of run unit is set to the key of the located record. The find record command has the form:

fr <record type name> <key> [<output file name>]

The find record command:

fr housing 405

would cause the following record to be output:

405*Billings*25

4.2.3.2. FIND OWNER

The find owner tool finds the owner record for a member key in a particular set. The current of run unit and the current of set type is set to the owner key.

fo <set type> <member key> [<output file name>]

The find owner command for student B2 in the fs set would consist of:

fo fs B2

and would yield the record:

Bill*A2*10*2132*57

The current of run unit and the current of fs member would be: B2.

4.2.3.3. FIND FIRST

The find first tool finds the first member of an owner in a specific set. Both the current of run unit and the current of set type, are set to the key of the first member. The record of the first member is output. The find first command has the form:

ff <set type> <owner key> [<output file name>]

The command to find the first member of B1 in sc would consist of:

ff sc B1

and would yield the record:

B1*0601*81*1*875*1*d*r

and would leave the current of run unit and current of set type set at 875B1811.

4.2.3.4. FIND NEXT

The find next tool finds the next member in a specified set. The next member is defined to be the next member in the set occurrence of the current of set key. The find next command resets the current of run unit and the current of set to the next member's key. If the next item in the set occurrence is the owner, a message is returned. Otherwise, the record of the next member is returned. The find next command has the form:

fn <set type> [<output file name>]

Assuming that the above "ff fs A2" command was given, followed by:

```
fn sc
```

The following record would be returned:

```
B1*0601*81*1*720*2*B*nr
```

If the following lists of commands were given:

```
ff fs A1
ff sc 3B
fn fs
fn sc
```

the output would be:

```
Mary:CAST:B1:Comp Scie
3B*0532*81*2*875*1*A*nr
John:SP:3B:PPPD
No more members
```

It should be noted that with these basic find commands, the software package interacting with the user, will be able to create different find macros, and therefore, offer the user a more extensive set of find commands. Also, since the record type files are regular UNIX files, the user can simply cat or print the record type file, if he wants a list of all the records in a particular file.

4.2.4. DATABASE REORGANIZATION

The database management system will allow the user to reorganize the set occurrences within a given set. A member can be moved from one owner to another owner within a given set.

4.2.4.1. CHANGE OWNER

The change owner tool moves a member from one set occurrence to another set occurrence. This is accomplished by resetting pointers in the link-records so that the member is linked to a new owner. The change owner command has the form:

co <new owner key> <set type> <member key>

The command to change student B1 in the set fs to a new advisor 4A would be:

co 4A fs B1

After this command, A1 would have only one advisee, 3B, and 4A would now be the advisor of B1.

4.2.4.2. CHANGE ALL

The change all tool moves all the members of an owner to a new owner. This is accomplished by repeated calls to change owner until all the members are moved. The change all command has the form:

ca <new owner key> <set type> <old owner key>

The command to move all of owner 405's members in hs to owner 216 would look like:

ca 216 hs 405

After this command was executed, 216 will have B1, 3B, and 5B as members, and 405 will have no members.

4.2.5. SAY GOOD-BYE

The final tool supplied by the database management system is a tool that allows the user to gracefully exit the updating session. The exit tool has the form:

q

After the exit command, the database management system stores record and set type definitions on a file, closes all record type, set type and B-tree files, and closes down the system.

5. CONCLUSIONS AND FUTURE EXTENSIONS

With our data model, we have tried to provide a solid foundation on which a flexible, user oriented sub-schema can sit. We have attempted to tailor our database management tools to accomplish specific tasks, yet allot them the power to do that task well. Our database record files are "clean" and fit well into the overall UNIX file system. However, as with all things this side of Paradise, certain extensions and enhancements would increase the effectiveness of our database management system. Our hope is that all or part of the following extensions will be implemented at a future time.

5.1. IMPLEMENT A SUB-SCHEMA WITH A DATA SUBLANGUAGE

We need to implement a subschema definition language and a stand alone user query language. This sublanguage should offer a greatly simplified view of the database to the user.

Within the user query language, certain enhancements could be made to the database management system. "Find" commands defined at this level could take some of the navigational tediousness away from the user. For example, the query language might allow the user to just say "find next" without presetting the current of set with a "find first". By keeping track of previous commands, the software could change the "find next" to a "find first". Also, a find macro might be nice. When a user wants to walk around a set occurrence, instead of repeated calls to "find next", the macro could expand a command, say "find all", to a "find first" followed by as many calls to "find next"as are needed. Perhaps, the software might even provide some safety alarms for the delete commands, to save the user from himself. Although it is not immediately apparent how this could be done, it

is a matter that should be pursued. Otherwise, the insatiable appetite of the delete commands will be a constant threat to the unwary user.

The sub-schema would have to implement some sort of field manipulation for the user. Our database management system deals in records. An add command, adds a whole record. A find command, finds a whole record. There will be times when the user wants to see a specific field of a record or will want to change a field. Obviously, we can't allow the user to change key fields, but he should be allowed to change non key fields. This option should be supplied to the user by the sub-schema.

5.2. TAKING OUT THE TRASH

When a delete of a set owner/member or the delete of a record is enacted under the database management system, the set occurrence or record is logically deleted from the database. That is, any reference to the deleted item will not be satisfied. However, the actual physical file record is not deleted. Over time, the database management system will cease to operate efficiently. Storage will be wasted on inaccessible and useless data. Time will be wasted since the buffering system will be required to do extra reads and writes to carry the defunct information. Obviously, we will want to create some vehicle for reclaiming this storage. There are many ways this garbage collection could be implemented.

The most elegant way to do garbage collection within the database management system would be to garbage collect while the user was inputing commands or perusing recently returned records. Some type of timing element would have to be built, so that every so often the garbage collection routine would check to see if the user had input a new command. This garbage collector would clean a certain

section of a file by moving the good records as far a possible toward the front of the file. The B-tree offset would then have to be reset. The real key to this approach would be the timing. If timed improperly, the garbage collector could clean to the end of file, be interrupted by the user who adds a new record to the end of the file, then start up again and blow away the recently added record. Granted, this approach would be a challenging design problem. However, it would buy you free garbage collection during user "think" time.

Another approach to the garbage collection problem is a B-tree walk and move procedure. A tool could be built that would visit each node in the B-tree index for a file. With the file offset gathered from the B-tree, the record could be moved to a new file and the offsets readjusted to fit the record's new placement. Once the new "clean" file is built, the old file space is returned to the system. This tool would have to be run without interruption and could tend to be expensive time wise if the database files got very large. However, it has the advantage of providing a means to the user to be absolutely certain that only valid records appear on a record file at a given time. Once the user ran this tool, he could "cat" the record file and have a copy of all the records of a certain record type.

Yet another approach to the refuse problem for record files would be the inspect and slide function. Armed with the record type definition, the garbage collection machine could move down the record file, inspecting each record as it moved. The key for the record would be recreated using the record type definition. Then the B-tree would be searched for that key. If the key was present in the B-tree, the record would slide as far forward on the file as possible, and then the offsets in the B-tree would be reset to reflect the new location. If the key

was not in the B-tree, the record would be overlaid when a valid record from farther down the file was slid over it. This method works well for the record type files; however, it will not work for the set type files, since the link-records carry no indication of their owners. At the same time, the link-records will generally take up less space than the record type records, and it is doubtful that a user would ever want to "cat" the link-records. Therefore, garbage collecting just the record files may be enough in some instances.

There are undoubtedly many more ways to collect wasted file space. The designer of the garbage collection tool will have to weigh the merits of each possibility, and implement the most reasonable approach. However, we know that garbage collection can be effectively done on the database files.

5.3. DELETION OF RECORD AND SET TYPES

It would be nice it the database management system allowed the user to delete a record or set type once it had been defined. Deletion of a record or set type would automatically carry with it deletion of any existing records or sets of that type. This deletion could be accomplished by just removing the name of the record/set type from the binary tree node where it is stored. The record/set type would then be logically deleted. With minor changes, the end of update routine could be designed to ignore any record or set type with a null name and not store it on the dbm_file. The next time the database management system was started, the deleted record or set type would not be in the binary tree. This approach gets complicated if the user is allowed to delete a record or set type anytime, because then the database management system has to contend with searching binary trees with null names scattered about. However, if the the user is only allowed to delete a record or set type at the end of an updating session, the implementation

becomes trivial.

5.4. SPIFFY OPTIONS

There are several options that would be convenient for the user and relatively easy to implement. It might be nice for the user to be able to get a copy of all deleted records. This could be accomplished by using a find to get the record for the user and then doing the delete. It could be implemented as a macro at either the schema or sub-schema level. A tool to allow the user to question the DBMS about a record type or set type definition would be useful. Perhaps the user would like to know what the field delimiter is for a particular record type. Along the same line, the user might want to change the field delimiter for a particular record type. In terms of implementation, these options are time consuming rather than difficult.

A very desirable tool would be a listing tool. The tool would allow the user to walk around a set type and list all the owners with their members. A variation of this tool, could list the record type files, with the fields listed in columns and the key fields indicated. A listing of all the set type and/or record type definitions would be useful.

Undoubtedly, pages and pages could be written on possible extensions the would be helpful to the user. Many users will have special requirements that make some options more important than others. However, as in all things, it comes down to time and resources. We have made every effort to make our data model flexible enough to handle future options and extensible enough to make the implementation easy. It is our fondest hope that this embryo will grow and thrive, and that it will eventually mature into a contributing adult in the field of database

management systems.

BIBLIOGRAPHY

Atre, S., Database - Structured Techniques For Design, Performance, and Management, Wiley-Interscience Publication, 1980

Comer, Douglas, "The Ubiquitous B-Tree", Computing Surveys, Vol. 11, No. 2, June 1979

Date, C. J., An Introduction To Database Systems, Addison-Wasley Publishing Company, 1977

Inmon, William H., Effective Database Design Prentice-Hall Inc., 1981

Joy, William, "An Introduction To The C Shell"

Kernighan and Plauger, Software Tools, Addison-Wesley Publishing Company, 1976

Kernighan and Plauger, The Elements Of Programming Style, McGraw-Hill Book Company, 1978

Kernighan and Ritchie, The C Programming Language, Prentice-Hall, Inc., 1978

Kernighan and Ritchie, "UNIX Programming - Second Edition"

Lehman and Yao, "Efficient Locking For Concurrent Operations On B-trees", 1979

Ritchie and Johnson, "The C Programming Language", The Bell System Technical Journal, Vol. 57, No. 6, Part 2

Ritchie and Thompson, "The UNIX Time-Sharing System", The Bell System Technical Journal, Vol. 57, No. 6, Part 2

Stonebraker, Michael, "Operating System Support For Database Management", Communications of the ACM Vol. 24 No. 7, July 1981

Thompson, K., "UNIX implementation", The Bell System Technical Journal, Vol. 57, No. 6, Part 2

Tremblay and Sorenson, An Introduction To Data Structures With Applications McGraw-Hill Book Company, 1976

BIBLIOGRAPHY

Ullman, Jeffry D., Principles of Database Subsystems, Computer Science Press, 1980

Wirth, Niklaus, Algorithms + Data Structures = Programs Prentice-Hall, Inc., 1976

APPENDIX A: LISTING OF DATABASE MANAGEMENT PROGRAMS

FILES LISTED

main.c
definitions
p_c_dec
parse_dbup
rec_t
set_t
init_concl
utility
io
dbup_c_dec
dbup

```
file          line level  source

main.c           1     0
main.c           2     0
main.c           3     0      /* This file contains the main driver for the database management system. */
main.c           4     0
main.c           5     0      #include <stdio.h>
main.c           6     0      #include "definitions"
main.c           7     0      #include "parse_dbup"
main.c           8     0      #include "rec_t"
main.c           9     0      #include "set_t"
main.c          10     0      #include "init_concl"
main.c          11     0      #include "utility"
main.c          12     0      #include "io"
main.c          13     0      #include "dbup"
main.c          14     0      main()
main.c          15     0      {
main.c          16     1              if (init() == OK)
main.c          17     1              {
main.c          18     2                      dbupdate();
main.c          19     2                      store_dbmf();
main.c          20     2              }
main.c          21     1      }
main.c          22     0
main.c          23     0
```

```
file          line level  source

definitions     51     0
definitions     52     0
definitions     53     0      struct ci                /*currency information for record or set types*/
definitions     54     0      {
definitions     55     1              int fd;
definitions     56     1              long log_fo;
definitions     57     1              long phy_fo;
definitions     58     1              long final_fo;
definitions     59     1              int log_bo;
definitions     60     1              int phy_bo;
definitions     61     1              int final_bo;
definitions     62     1              int add_on;
definitions     63     1              char file_buf[BLOCKSIZE];
definitions     64     1      };
definitions     65     0
definitions     66     0
definitions     67     0
definitions     68     0
definitions     69     0
definitions     70     0      struct rec_type
definitions     71     0      {                        /*structure for information for a record type*/
definitions     72     1              char name[MAX_TNAME + 1];
definitions     73     1              char fdlim;              /*field delimiter for this record type*/
definitions     74     1              int nosfield;
definitions     75     1              int noskeyf;
definitions     76     1              int keyf[MAX_KEY_FIELDS];
definitions     77     1              struct ci rci;           /*record currency information*/
definitions     78     1              struct rec_type *left;
definitions     79     1              struct rec_type *right;
definitions     80     1      };
definitions     81     0
definitions     82     0
definitions     83     0
definitions     84     0      struct set_type
definitions     85     0      {
definitions     86     1              char sname[MAX_TNAME + 1];
definitions     87     1              struct rec_type *owner;
definitions     88     1              struct rec_type *member;
definitions     89     1              char cos [MAX_KEY + 1];          /*current of set type*/
definitions     90     1              int sbtfd;
definitions     91     1              long sbtroot;
definitions     92     1              struct ci sci;           /*set currency information*/
definitions     93     1              struct set_type *sleft;
definitions     94     1              struct set_type *sright;
definitions     95     1      };
definitions     96     0
definitions     97     0
definitions     98     0
definitions     99     0      struct ptrs
definitions    100     0      {
```

```
file          line level  source

definitions    101     1              long btp;              /*struct used for communication with btree routines*/
definitions    102     1              int bts;
definitions    103     1              long btd;
definitions    104     1              int btb;
definitions    105     1      };
definitions    106     0                          /*Global Declarations*/
definitions    107     0      struct rec_type *rt_root = NULL;        /*root for record type*/
definitions    108     0      struct rec_type *cor_rt = NULL;         /*current of run record type*/
definitions    109     0      struct set_type *st_root = NULL;        /*root for set type*/
definitions    110     0      struct set_type *cor_st = NULL;         /*current of run set type*/
definitions    111     0      char cor_unit[MAX_KEY+1];               /*current of run unit key*/
definitions    112     0      int cnt_rt=0;                           /*number of record types in db*/
definitions    113     0      int cnt_st=0;                           /*number of set types in db*/
definitions    114     0      long rbtroot;                           /*root for record btree*/
definitions    115     0      int rbtfd;                              /*file descriptor for record btree*/
```

```
file          line level  source

p_c_dec          1     0       /*This file contains common declarations for the parse routines.*/
p_c_dec          2     0
p_c_dec          3     0
p_c_dec          4     0       char key[MAX_KEY + 1];
p_c_dec          5     0       char sname[MAX_TNAME + 1];
p_c_dec          6     0       char rname[MAX_TNAME + 1];
p_c_dec          7     0       char fname[MAX_TNAME + 1];
p_c_dec          8     0       extern struct set_type *cor_st;
p_c_dec          9     0       extern struct rec_type *cor_rt;
p_c_dec         10     0       extern char cor_unit[MAX_KEY + 1];
p_c_dec         11     0       struct rec_type *prname(),*pr;
p_c_dec         12     0       struct set_type *psname(),*ps;
p_c_dec         13     0       int spec;
```

```
file          line level  source

parse_dbup       1     0
parse_dbup       2     0      /*Dbupdate parses user commands and calls appropriate subroutines*/
parse_dbup       3     0
parse_dbup       4     0
parse_dbup       5     0      dbupdate()
parse_dbup       6     0      {
parse_dbup       7     1              int more=TRUE;
parse_dbup       8     1              char cmd[MAX_CMD + 21];
parse_dbup       9     1              int i,cond,status;
parse_dbup      10     1              char c;
parse_dbup      11     1
parse_dbup      12     1      #ifdef DEBUG
parse_dbup      13     1      printf("e dbupdate\n");
parse_dbup      14     1      #endif
parse_dbup      15     1
parse_dbup      16     1              cond=OK;
parse_dbup      17     1              while (more)
parse_dbup      18     1              {
parse_dbup      19     2                      printf("%c", PROMPT);
parse_dbup      20     2                      if (scanf(FMT3, cmd) != NULL)
parse_dbup      21     2                          switch (*cmd)
parse_dbup      22     2                          {
parse_dbup      23     3                              case 'q':       /*parse command*/
parse_dbup      24     3                                      printf("end of update\n");
parse_dbup      25     3                                      more = FALSE;
parse_dbup      26     3                                      status = LNP;
parse_dbup      27     3                                      break;
parse_dbup      28     3                              case 'r':       /*record type add*/
parse_dbup      29     3                                      status = prect_add();
parse_dbup      30     3                                      break;
parse_dbup      31     3                              case 's':       /*set type add*/
parse_dbup      32     3                                      status = psett_add();
parse_dbup      33     3                                      break;
parse_dbup      34     3                              case 'a':
parse_dbup      35     3                                  switch (*(cmd+1))
parse_dbup      36     3                                  {
parse_dbup      37     4                                      case 'o':       /*owner add*/
parse_dbup      38     4                                              status = pownadd();
parse_dbup      39     4                                              break;
parse_dbup      40     4                                      case 'm':       /*member add*/
parse_dbup      41     4                                              status = pmebadd();
parse_dbup      42     4                                              break;
parse_dbup      43     4                                      case 'r':       /*record add*/
parse_dbup      44     4                                              status = precadd();
parse_dbup      45     4                                              break;
parse_dbup      46     4                                      default:
parse_dbup      47     4                                              cond = ERROR;
parse_dbup      48     4                                              break;
parse_dbup      49     4                                  }
parse_dbup      50     3                                  break;
```

```
file          line level  source

parse_dbup      51     3                        case 'c':
parse_dbup      52     3                            status = pchange(*(cmd+1));          /*parse change command*/
parse_dbup      53     3                            break;
parse_dbup      54     3                        case 'd':
parse_dbup      55     3                            switch (*(cmd+1))
parse_dbup      56     3                            {
parse_dbup      57     4                                case 'm':        /*delete member*/
parse_dbup      58     4                                        status = pdel(*(cmd+1));
parse_dbup      59     4                                        break;
parse_dbup      60     4                                case 'o':        /*delete owner*/
parse_dbup      61     4                                        status = pdel(*(cmd+1));
parse_dbup      62     4                                        break;
parse_dbup      63     4                                case 'r':        /*delete record*/
parse_dbup      64     4                                        status = precdel();
parse_dbup      65     4                                        break;
parse_dbup      66     4                                default:
parse_dbup      67     4                                        cond = ERROR;
parse_dbup      68     4                                        break;
parse_dbup      69     4                            }
parse_dbup      70     3                            break;
parse_dbup      71     3                        case 'f':
parse_dbup      72     3                            switch (*(cmd+1))
parse_dbup      73     3                            {
parse_dbup      74     4                                case 'r':        /*find record*/
parse_dbup      75     4                                        status = precfind();
parse_dbup      76     4                                        break;
parse_dbup      77     4                                default:
parse_dbup      78     4                                        status = pfind(*(cmd+1));
parse_dbup      79     4                                        break;
parse_dbup      80     4                            }
parse_dbup      81     3                            break;
parse_dbup      82     3                    default:
parse_dbup      83     3                        cond = ERROR;
parse_dbup      84     3                        break;
parse_dbup      85     3                    }
parse_dbup      86     2                else
parse_dbup      87     2                        cond = ERROR;
parse_dbup      88     2                if (cond == ERROR)
parse_dbup      89     2                {
parse_dbup      90     3                        printf("illegal command\n");
parse_dbup      91     3                        cond = OK;
parse_dbup      92     3                        status = LNP;            /*full line not parsed*/
parse_dbup      93     3                }
parse_dbup      94     2                if (status != LP)
parse_dbup      95     2                        while ((c = getc(stdin)) != END_OF_LINE)
parse_dbup      96     2                        ;                        /*scan the rest of the line*/
parse_dbup      97     2
parse_dbup      98     2            }
parse_dbup      99     1        }
parse_dbup     100     0
```

```
file          line level  source

parse_dbup     101     0
parse_dbup     102     0
parse_dbup     103     0
parse_dbup     104     0
```

```
file          line level  source

parse_dbup     106     0      /*Prect_add parses the command line for a record type add command, gathering
parse_dbup     107     0        information.  If the information is complete, rect_add is called.*/
parse_dbup     108     0
parse_dbup     109     0
parse_dbup     110     0      prect_add()
parse_dbup     111     0      {
parse_dbup     112     1              int nosfld, noskeyf;
parse_dbup     113     1              int keyf[MAX_KEY_FIELDS];
parse_dbup     114     1              struct rec_type *prname();
parse_dbup     115     1              char rtname[MAX_TNAME + 1];
parse_dbup     116     1              char fdlm;
parse_dbup     117     1              int i;
parse_dbup     118     1              int spec;
parse_dbup     119     1
parse_dbup     120     1
parse_dbup     121     1      #ifdef DEBUG
parse_dbup     122     1      printf("e prect_add\n");
parse_dbup     123     1      #endif
parse_dbup     124     1
parse_dbup     125     1              if (prname(rtname, &spec) == NULL && spec != PRESENT)
parse_dbup     126     1                      return(error("usage: ra new-rec-type-name #fields #key-fields list-key-fields"));
parse_dbup     127     1              else if (jumpup() != PRESENT ||scanf(" %c", &fdlm) == NULL)
parse_dbup     128     1                      return(error("missing field delimiter"));
parse_dbup     129     1              else
parse_dbup     130     1              {                       /*read and validate key field information*/
parse_dbup     131     2                      scanf(" %d %d", &nosfld, &noskeyf);
parse_dbup     132     2                      if (noskeyf > MAX_KEY_FIELDS || noskeyf < 1 || noskeyf > nosfld)
parse_dbup     133     2                              return(error("illegal number of key fields"));
parse_dbup     134     2                      else if (nosfld <1)
parse_dbup     135     2                              return(error("illegal number of fields"));
parse_dbup     136     2                      else
parse_dbup     137     2                      {
parse_dbup     138     3                              for (i=0;i<MAX_KEY_FIELD; i++)
parse_dbup     139     3                                      *(keyf+i) = 0;          /*zero key fld array*/
parse_dbup     140     3                              for (i=0; i<noskeyf; i++)
parse_dbup     141     3                              {
parse_dbup     142     4                                      scanf(" %d", keyf+i);
parse_dbup     143     4                                      if (*(keyf+i) < 1 || *(keyf+i) >nosfld)
parse_dbup     144     4                                              return(error("illegal key field specification"));
parse_dbup     145     4                              }
parse_dbup     146     3                              return(rect_add(rtname,fdlm,nosfld,noskeyf,keyf));  /*add the record ty
parse_dbup     147     3                      }
parse_dbup     148     2              }
parse_dbup     149     1      }
parse_dbup     150     0
parse_dbup     151     0
parse_dbup     152     0
parse_dbup     153     0
parse_dbup     154     0
```

```
file          line level  source

parse_dbup     156     0       /*Psett_add parses the command line for a set type add command, gathering
parse_dbup     157     0         information.  If the information is complete, sett_add is called.*/
parse_dbup     158     0
parse_dbup     159     0
parse_dbup     160     0       psett_add()
parse_dbup     161     0       {
parse_dbup     162     1               char stname[MAX_TNAME + 1];
parse_dbup     163     1               char owner [MAX_TNAME + 1], member [MAX_TNAME + 1];
parse_dbup     164     1               struct set_type *psname();
parse_dbup     165     1               struct rec_type *prname();
parse_dbup     166     1               int spec;
parse_dbup     167     1
parse_dbup     168     1
parse_dbup     169     1       #ifdef DEBUG
parse_dbup     170     1       printf("e psett_add\n");
parse_dbup     171     1       #endif
parse_dbup     172     1
parse_dbup     173     1               if(psname(stname,&spec) == NULL && spec != PRESENT)
parse_dbup     174     1                       return(error("usage: sa new-set-name owner-name member-name"));
parse_dbup     175     1               else
parse_dbup     176     1               {
parse_dbup     177     2                       /*scan owner and member names*/
parse_dbup     178     2                       if (prname(owner,&spec) == NULL && spec != PRESENT ||
parse_dbup     179     2                               prname(member,&spec) == NULL && spec != PRESENT)
parse_dbup     180     2                               return(error("missing owner or member name"));
parse_dbup     181     2                       else
parse_dbup     182     2                               return(sett_add(stname,owner,member));          /*add new set type*/
parse_dbup     183     2               }
parse_dbup     184     1       }
parse_dbup     185     0
parse_dbup     186     0
parse_dbup     187     0
parse_dbup     188     0
parse_dbup     189     0
```

```
file          line level  source

parse_dbup     191     0       /*Pownadd parses the command line for an owner add command, sets the current
parse_dbup     192     0         of run pointers, and calls ownadd*/
parse_dbup     193     0
parse_dbup     194     0
parse_dbup     195     0       pownadd()
parse_dbup     196     0       {
parse_dbup     197     1       #include "p_c_dec"      /*include common parse declarations*/
parse_dbup     198     1
parse_dbup     199     1       #ifdef DEBUG
parse_dbup     200     1       printf("e pownadd\n");
parse_dbup     201     1       #endif
parse_dbup     202     1
parse_dbup     203     1
parse_dbup     204     1               if ((ps = psname(sname,&spec)) == NULL &&
parse_dbup     205     1                       spec != PRESENT)        /*scan the set type name*/
parse_dbup     206     1                       return(error("missing set name"));
parse_dbup     207     1               else if (ps == NULL)
parse_dbup     208     1                       return(error("set type not defined"));
parse_dbup     209     1               else if (pkey(key) == ERROR)            /*scan occurrence key*/
parse_dbup     210     1                       return(ERROR);
parse_dbup     211     1               else
parse_dbup     212     1               {
parse_dbup     213     2                       cor_st = ps;            /*set the cor pointers*/
parse_dbup     214     2                       strcpy(cor_unit,key);
parse_dbup     215     2                       return(ownadd());       /*add an owner set occurrence*/
parse_dbup     216     2               }
parse_dbup     217     1       }
parse_dbup     218     0
parse_dbup     219     0
parse_dbup     220     0
parse_dbup     221     0
parse_dbup     222     0
```

```
file          line level  source

parse_dbup     224     0       /*Pmebadd parses the command line for an add member occurrence command, sets
parse_dbup     225     0         the current of run pointers, and calls member add.*/
parse_dbup     226     0
parse_dbup     227     0
parse_dbup     228     0       pmebadd()
parse_dbup     229     0       {
parse_dbup     230     1       #include "p_c_dec"                          /*common parse declarations*/
parse_dbup     231     1               char mkey[MAX_KEY + 1];
parse_dbup     232     1
parse_dbup     233     1       #ifdef DEBUG
parse_dbup     234     1       printf("e pmebadd\n");
parse_dbup     235     1       #endif
parse_dbup     236     1
parse_dbup     237     1
parse_dbup     238     1               if (pkey(mkey) == ERROR)          /*parse member key*/
parse_dbup     239     1                       return(ERROR);
parse_dbup     240     1                                        /*parse record type name*/
parse_dbup     241     1               else if ((ps = psname(sname,&spec)) == NULL && spec != PRESENT)
parse_dbup     242     1                       return(error("missing record type"));
parse_dbup     243     1               else if (ps == NULL)
parse_dbup     244     1                       return(error("set type not definded"));
parse_dbup     245     1               else if (pkey(key) == ERROR)                /*parse owner key*/
parse_dbup     246     1                       return(ERROR);
parse_dbup     247     1               else
parse_dbup     248     1               {
parse_dbup     249     2                       cor_st = ps;              /*set currency pointers*/
parse_dbup     250     2                       strcpy(cor_unit,key);
parse_dbup     251     2                       return(mebadd(mkey));
parse_dbup     252     2               }
parse_dbup     253     1       }
parse_dbup     254     0
parse_dbup     255     0
parse_dbup     256     0
parse_dbup     257     0
parse_dbup     258     0
```

```
file          line level  source

parse_dbup     260     0      /*Precadd parses the command line for a record occurrence add, sets the
parse_dbup     261     0        currency pointers, and calls recadd.*/
parse_dbup     262     0
parse_dbup     263     0
parse_dbup     264     0      precadd()
parse_dbup     265     0      {
parse_dbup     266     1              FILE *ifp;
parse_dbup     267     1      #include "p_c_dec"              /*common parse definitions*/
parse_dbup     268     1              char c;
parse_dbup     269     1
parse_dbup     270     1
parse_dbup     271     1      #ifdef DEBUG
parse_dbup     272     1      printf("e precadd\n");
parse_dbup     273     1      #endif
parse_dbup     274     1
parse_dbup     275     1                                     /*parse record type name*/
parse_dbup     276     1              if ((pr = prname(rname,&spec)) == NULL && spec != PRESENT)
parse_dbup     277     1                      return(error("missing record type"));
parse_dbup     278     1              else if(pr == NULL)
parse_dbup     279     1                      return(error("record type not defined"));
parse_dbup     280     1              else if (pfname (fname, READ, &ifp) == ERROR)   /*parse file name*/
parse_dbup     281     1                      return(ERROR);
parse_dbup     282     1              else
parse_dbup     283     1              {
parse_dbup     284     2                      cor_rt = pr;            /*set currency pointers*/
parse_dbup     285     2                      while ((c = getc(stdin)) != END_OF_LINE)
parse_dbup     286     2                              ;                       /*scan rest of line*/
parse_dbup     287     2                      recadd(ifp);    /*add record occurrence*/
parse_dbup     288     2                      return(LP);             /*have read past end of line*/
parse_dbup     289     2              }
parse_dbup     290     1      }
parse_dbup     291     0
parse_dbup     292     0
parse_dbup     293     0
parse_dbup     294     0
parse_dbup     295     0
```

```
file          line level  source

parse_dbup     297     0       /*Precdel parses the command line for a record occurrence delete, sets the
parse_dbup     298     0         currency pointers, and calls record occurrence delete.*/
parse_dbup     299     0
parse_dbup     300     0
parse_dbup     301     0       precdel()
parse_dbup     302     0       {
parse_dbup     303     1       #include "p_c_dec"
parse_dbup     304     1
parse_dbup     305     1       #ifdef DEBUG
parse_dbup     306     1       printf("e precdel\n");
parse_dbup     307     1       #endif
parse_dbup     308     1
parse_dbup     309     1
parse_dbup     310     1                                       /*parse record type name*/
parse_dbup     311     1               if ((pr= prname(rname,&spec)) == NULL && spec != PRESENT)
parse_dbup     312     1                       return(error("missing record type"));
parse_dbup     313     1               else if (pr == NULL)
parse_dbup     314     1                       return(error("record type not defined"));
parse_dbup     315     1               else if (pkey (key) == ERROR)
parse_dbup     316     1                       return(ERROR);
parse_dbup     317     1               else
parse_dbup     318     1               {
parse_dbup     319     2                       cor_rt = pr;            /*set currency pointers*/
parse_dbup     320     2                       strcpy(cor_unit,key);
parse_dbup     321     2                       return (recdel());      /*delete record occurrence*/
parse_dbup     322     2               }
parse_dbup     323     1       }
parse_dbup     324     0
parse_dbup     325     0
parse_dbup     326     0
parse_dbup     327     0
parse_dbup     328     0
```

```
file          line level  source

parse_dbup     330     0       /*Pdel parses a command line for an owner or member occurrence delete, sets
parse_dbup     331     0         the currency pointers, and calls owner delete or member delete depending
parse_dbup     332     0         on the cmd parameter.*/
parse_dbup     333     0
parse_dbup     334     0
parse_dbup     335     0       pdel(cmd)
parse_dbup     336     0       char cmd;
parse_dbup     337     0       {
parse_dbup     338     1       #include "p_c_dec"              /*common parse declarations*/
parse_dbup     339     1
parse_dbup     340     1       #ifdef DEBUG
parse_dbup     341     1       printf("e pdel\n");
parse_dbup     342     1       #endif
parse_dbup     343     1
parse_dbup     344     1
parse_dbup     345     1               if ((ps = psname(sname,&spec)) == NULL && spec != PRESENT)
parse_dbup     346     1                       return(error("missing record type"));
parse_dbup     347     1               else if (ps == NULL)
parse_dbup     348     1                       return(error("set name undefined"));
parse_dbup     349     1               else if (pkey(key) == ERROR)
parse_dbup     350     1                       return(ERROR);
parse_dbup     351     1               else
parse_dbup     352     1               {
parse_dbup     353     2                       cor_st = ps;            /*set currency pointers*/
parse_dbup     354     2                       strcpy (cor_unit,key);
parse_dbup     355     2                       if (cmd == 'o')         /*call owner occurrence delete*/
parse_dbup     356     2                               return (owndel());
parse_dbup     357     2                       else
parse_dbup     358     2                               return (mebdel(NEWL,NEW));
parse_dbup     359     2               }
parse_dbup     360     1       }
parse_dbup     361     0
parse_dbup     362     0
parse_dbup     363     0
parse_dbup     364     0
parse_dbup     365     0
```

```
file          line level  source

parse_dbup     367     0      /*Precfind parses the command line for a find record occurrence command, sets
parse_dbup     368     0        currency pointers, and call record find*/
parse_dbup     369     0
parse_dbup     370     0
parse_dbup     371     0      precfind()
parse_dbup     372     0      {
parse_dbup     373     1      #include "p_c_dec"
parse_dbup     374     1              FILE *ofp;
parse_dbup     375     1
parse_dbup     376     1
parse_dbup     377     1      #ifdef DEBUG
parse_dbup     378     1      printf ("e precfind\n");
parse_dbup     379     1      #endif
parse_dbup     380     1
parse_dbup     381     1                                          /*parse record type name*/
parse_dbup     382     1              if ((pr = prname(rname,&spec)) == NULL && spec != PRESENT)
parse_dbup     383     1                      return(error("missing record type"));
parse_dbup     384     1              else if (pr == NULL)
parse_dbup     385     1                      return(error("record type not defined"));
parse_dbup     386     1                                          /*parse key and optional file name*/
parse_dbup     387     1              else if (pkey(key) == ERROR || pfname(fname,"APPEND",&ofp) == ERROR)
parse_dbup     388     1                      return(ERROR);
parse_dbup     389     1              else
parse_dbup     390     1              {
parse_dbup     391     2                      cor_rt = pr;            /*set currency pointers*/
parse_dbup     392     2                      strcpy (cor_unit,key);
parse_dbup     393     2                      return (recfind(ofp));
parse_dbup     394     2              }
parse_dbup     395     1      }
parse_dbup     396     0
parse_dbup     397     0
parse_dbup     398     0
parse_dbup     399     0
parse_dbup     400     0
```

```
file        line level  source

parse_dbup   402    0       /*Pfind parses a command line for a find owner, a find first member, or a find
parse_dbup   403    0         next command, sets currency pointers, and appropriate find routines.*/
parse_dbup   404    0
parse_dbup   405    0
parse_dbup   406    0       pfind (cmd)
parse_dbup   407    0       char cmd;
parse_dbup   408    0       {
parse_dbup   409    1       #include "p_c_dec"
parse_dbup   410    1               FILE *ofp;              /*output file descriptor*/
parse_dbup   411    1
parse_dbup   412    1       #ifdef DEBUG
parse_dbup   413    1       printf ("e pfind\n");
parse_dbup   414    1       #endif
parse_dbup   415    1
parse_dbup   416    1
parse_dbup   417    1                               /*parse set name, key, and optional output file name*/
parse_dbup   418    1               if ((ps = psname(sname,&spec)) == NULL && spec != PRESENT)
parse_dbup   419    1                       return(error("missing set type"));
parse_dbup   420    1               else if (ps == NULL)
parse_dbup   421    1                       return(error("set type undefined"));
parse_dbup   422    1               else if (cmd == 'n')
parse_dbup   423    1               {                                       /*handle find next command*/
parse_dbup   424    2                       cor_st = ps;
parse_dbup   425    2                       if (pfname(fname, APPEND, &ofp) == ERROR)
parse_dbup   426    2                               return (ERROR);
parse_dbup   427    2                       else
parse_dbup   428    2                               return (nextfind (ofp));
parse_dbup   429    2               }
parse_dbup   430    1               else if (pkey(key) == ERROR || pfname(fname, APPEND, &ofp) == ERROR)
parse_dbup   431    1                       return(ERROR);
parse_dbup   432    1               else
parse_dbup   433    1               {
parse_dbup   434    2                       cor_st = ps;                    /*set currency pointers*/
parse_dbup   435    2                       strcpy(cor_unit,key);
parse_dbup   436    2                       switch (cmd)
parse_dbup   437    2                       {
parse_dbup   438    3                               case 'f':               /*find first member*/
parse_dbup   439    3                                       return (mebfind(ofp));
parse_dbup   440    3                               case 'o':
parse_dbup   441    3                                       return (ownfind(ofp));
parse_dbup   442    3                               default:
parse_dbup   443    3                                       return(error("illegal command"));
parse_dbup   444    3                       }
parse_dbup   445    2               }
parse_dbup   446    1       }
parse_dbup   447    0
parse_dbup   448    0
parse_dbup   449    0
parse_dbup   450    0
parse_dbup   451    0
```

```
file          line level  source

parse_dbup     453     0      /*Pchange parses the command line for a change all or change owner command,
parse_dbup     454     0        sets the currency pointers, and calls the appropriate change routine.*/
parse_dbup     455     0
parse_dbup     456     0      pchange (cmd)
parse_dbup     457     0      char cmd;
parse_dbup     458     0      {
parse_dbup     459     1      #include "p_c_dec"
parse_dbup     460     1              char newokey[MAX_KEY + 1];      /*new owner/member key*/
parse_dbup     461     1
parse_dbup     462     1      #ifdef DEBUG
parse_dbup     463     1      printf ("e pchange\n");
parse_dbup     464     1      #endif
parse_dbup     465     1
parse_dbup     466     1
parse_dbup     467     1              if (pkey(newokey) == ERROR)             /*parse new owner/member key*/
parse_dbup     468     1                      return(ERROR);
parse_dbup     469     1                                                      /*parse set type name*/
parse_dbup     470     1              else if ((ps = psname(sname,&spec)) == NULL && spec != PRESENT)
parse_dbup     471     1                      return(error("missing set type name"));
parse_dbup     472     1              else if (ps == NULL)
parse_dbup     473     1                      return(error("set type undefined"));
parse_dbup     474     1              else if (pkey(key) == ERROR)                    /*parse old owner/member key*/
parse_dbup     475     1                      return(ERROR);
parse_dbup     476     1
parse_dbup     477     1              else
parse_dbup     478     1              {
parse_dbup     479     2                      cor_st = ps;                    /*set currency pointers*/
parse_dbup     480     2                      strcpy(cor_unit,key);           /*set at old owner/member*/
parse_dbup     481     2                      switch(cmd)
parse_dbup     482     2                      {                       /*call appropriate change routine*/
parse_dbup     483     3                              case 'a':       /*change all members to new owner*/
parse_dbup     484     3                                      return (cngall(newokey));
parse_dbup     485     3                              case 'o':       /*change member to new owner*/
parse_dbup     486     3                                      return (cngown(newokey));
parse_dbup     487     3                              default:
parse_dbup     488     3                                      return(error("illegal command"));
parse_dbup     489     3                      }
parse_dbup     490     2              }
parse_dbup     491     1      }
parse_dbup     492     0
parse_dbup     493     0
parse_dbup     494     0
parse_dbup     495     0
parse_dbup     496     0
parse_dbup     497     0                      /*parse utility routines*/
parse_dbup     498     0
parse_dbup     499     0
```

```
file          line level  source

parse_dbup     501     0      /*Prname scans standard input for a string, truncates the string to MAX_TNAME
parse_dbup     502     0        and places it in string passed in as a parameter.  Prname searches the
parse_dbup     503     0        record type tree for the record name indicated by the string.
parse_dbup     504     0             prname returns: NULL if the string is empty
parse_dbup     505     0                             NULL if the string is not a rname in the record type tree
parse_dbup     506     0                             pointer to record type if rname is in the record type tree*/
parse_dbup     507     0
parse_dbup     508     0
parse_dbup     509     0      struct rec_type *prname(rtname, spec)
parse_dbup     510     0      char *rtname;
parse_dbup     511     0      int *spec;
parse_dbup     512     0      {
parse_dbup     513     1              extern struct rec_type *rt_root;
parse_dbup     514     1              struct rec_type *rt_search();
parse_dbup     515     1
parse_dbup     516     1      #ifdef DEBUG
parse_dbup     517     1      printf ("e prname \n");
parse_dbup     518     1      #endif
parse_dbup     519     1
parse_dbup     520     1              if((*spec = jumpup()) == PRESENT)
parse_dbup     521     1              {
parse_dbup     522     2                      scanf (FMT1, rtname);
parse_dbup     523     2                      return(rt_search(rt_root, rtname));
parse_dbup     524     2              }
parse_dbup     525     1              else
parse_dbup     526     1                      return(NULL);
parse_dbup     527     1      }
parse_dbup     528     0
parse_dbup     529     0
parse_dbup     530     0
parse_dbup     531     0
parse_dbup     532     0
```

```
file          line level  source

parse_dbup     534     0       /*Psname scans standard input for a string, truncates the string to MAX_TNAME,
parse_dbup     535     0         and places the string into the parameter passed into the routine.  Psname
parse_dbup     536     0         searches the set type tree for the set name indicated by the string.
parse_dbup     537     0                       psname returns: NULL if the string is empty
parse_dbup     538     0                                       NULL if the string is not a sname in the set tree
parse_dbup     539     0                                       pointer to set type if sname is in set tree*/
parse_dbup     540     0
parse_dbup     541     0
parse_dbup     542     0       struct set_type *psname (stname,spec)
parse_dbup     543     0       char *stname;
parse_dbup     544     0       int *spec;
parse_dbup     545     0       {
parse_dbup     546     1               extern struct set_type *st_root;
parse_dbup     547     1               struct set_type *st_search();
parse_dbup     548     1
parse_dbup     549     1       #ifdef DEBUG
parse_dbup     550     1       printf("entering set type parse\n");
parse_dbup     551     1       #endif
parse_dbup     552     1
parse_dbup     553     1               if ((*spec = jumpup()) == PRESENT)
parse_dbup     554     1               {
parse_dbup     555     2                       scanf(FMT1, stname);            /*scan name*/
parse_dbup     556     2                       return(st_search(st_root, stname));
parse_dbup     557     2               }
parse_dbup     558     1               else
parse_dbup     559     1                       return(NULL);
parse_dbup     560     1       }
parse_dbup     561     0
parse_dbup     562     0
parse_dbup     563     0
parse_dbup     564     0
parse_dbup     565     0
```

```
file          line level  source

parse_dbup     567     0       /*Pfname scans a file name.  If the file name is null, fp is set to stdin
parse_dbup     568     0         or stdout depending on the mode. Otherwise, the file is opened.*/
parse_dbup     569     0
parse_dbup     570     0       pfname (fname, mode, fp)
parse_dbup     571     0       char *mode;
parse_dbup     572     0       char *fname;
parse_dbup     573     0       FILE **fp;
parse_dbup     574     0       {
parse_dbup     575     1               FILE *fopen();
parse_dbup     576     1
parse_dbup     577     1       #ifdef DEBUG
parse_dbup     578     1       printf("entering parse file name\n");
parse_dbup     579     1       #endif
parse_dbup     580     1
parse_dbup     581     1               if (jumpup() != PRESENT)
parse_dbup     582     1                       if (strcmp (mode, READ) == MATCH)
parse_dbup     583     1                               *fp = stdin;
parse_dbup     584     1                       else
parse_dbup     585     1                               *fp = stdout;
parse_dbup     586     1               else
parse_dbup     587     1               {
parse_dbup     588     2                       scanf (FMT1, fname);
parse_dbup     589     2                       if ((*fp = fopen(fname,mode)) == NULL)
parse_dbup     590     2                               return(error("unable to open file"));
parse_dbup     591     2               }
parse_dbup     592     1               return(OK);
parse_dbup     593     1       }
parse_dbup     594     0
parse_dbup     595     0
parse_dbup     596     0
parse_dbup     597     0
parse_dbup     598     0
parse_dbup     599     0
```

```
file          line level  source

parse_dbup     601     0      /*Pkey parses an input key.  If key entered is too long an error message is
parse_dbup     602     0        returned. Otherwise, OK is returned.*/
parse_dbup     603     0
parse_dbup     604     0      pkey (key)
parse_dbup     605     0      char *key;
parse_dbup     606     0      {
parse_dbup     607     1              char inkey[MAX_KEY + 6];
parse_dbup     608     1
parse_dbup     609     1      #ifdef DEBUG
parse_dbup     610     1      printf ("entering pkey\n");
parse_dbup     611     1      #endif
parse_dbup     612     1
parse_dbup     613     1              if (jumpup() == PRESENT)
parse_dbup     614     1              {
parse_dbup     615     2                      scanf(FMT2, inkey);
parse_dbup     616     2                      if (strlen(inkey) > MAX_KEY)
parse_dbup     617     2                              return(error("input key too long"));
parse_dbup     618     2                      else
parse_dbup     619     2                      {
parse_dbup     620     3                              strcpy(key, inkey);
parse_dbup     621     3                              return (OK);
parse_dbup     622     3                      }
parse_dbup     623     2              }
parse_dbup     624     1              else
parse_dbup     625     1                      return(error("missing key"));
parse_dbup     626     1      }
parse_dbup     627     0
parse_dbup     628     0
parse_dbup     629     0
parse_dbup     630     0
parse_dbup     631     0
```

```
file            line level  source

parse_dbup       633     0      /*Jumpup scans blanks and tabs, and returns Present(1) if the next character is
parse_dbup       634     0        not a new line and NOTPRESENT if the next character is a new line.*/
parse_dbup       635     0
parse_dbup       636     0
parse_dbup       637     0      jumpup()
parse_dbup       638     0      {
parse_dbup       639     1              char c;
parse_dbup       640     1
parse_dbup       641     1      #ifdef DEBUG
parse_dbup       642     1      printf("e jumpup\n");
parse_dbup       643     1      #endif
parse_dbup       644     1
parse_dbup       645     1              while (((c = getc(stdin)) == BLANK) || (c == TAB))
parse_dbup       646     1                      ;               /*skip blanks and tabs*/
parse_dbup       647     1              ungetc(c, stdin);
parse_dbup       648     1              if (c == '\n')
parse_dbup       649     1                      return (NOTPRESENT);
parse_dbup       650     1              else
parse_dbup       651     1                      return (PRESENT);
parse_dbup       652     1      }
```

```
file          line level  source

rec_t            1     0       /*This file contains the routines to build the record type tree, insert a
rec_t            2     0         record type into the tree, and search the tree for a particular record type.*/
rec_t            3     0
rec_t            4     0       /*MAD 10-81*/
rec_t            5     0
rec_t            6     0       /*Brec_tree reads the data base management file and builds a tree contain-
rec_t            7     0         ing the information on the different record types in the data base*/
rec_t            8     0
rec_t            9     0       brec_tree (fp,number)
rec_t           10     0       FILE *fp;
rec_t           11     0       int number;
rec_t           12     0       {
rec_t           13     1               int i;
rec_t           14     1
rec_t           15     1       #ifdef DEBUG
rec_t           16     1       printf("entering brec_tree\n");
rec_t           17     1       #endif
rec_t           18     1
rec_t           19     1               for (i=1; i<= number; i++)
rec_t           20     1                       if (arec_tt (fp) == ERROR)
rec_t           21     1                               return(ERROR);
rec_t           22     1               return(OK);
rec_t           23     1       }
rec_t           24     0
rec_t           25     0
rec_t           26     0
rec_t           27     0
```

```
file            line level  source

rec_t             29     0       /*Arec_tt will scan the dbm file and build a record type and place it in
rec_t             30     0         the record type b-tree.  Returns -1 if error occurs.*/
rec_t             31     0
rec_t             32     0       arec_tt (fp)
rec_t             33     0       FILE *fp;                          /*dbm file pointer*/
rec_t             34     0       {
rec_t             35     1               int i,cond;
rec_t             36     1               extern struct rec_type *rt_root;
rec_t             37     1               struct rec_type r;
rec_t             38     1               struct rec_type *rt_insert();
rec_t             39     1                                                  /*read information for one record type*/
rec_t             40     1               fscanf(fp,"%s %c %d %d", r.name, &r.fdlim, &r.nosfield, &r.noskeyf);
rec_t             41     1               for (i=0; i<r.noskeyf; i++)
rec_t             42     1                       fscanf(fp,"%d ", r.keyf+i);
rec_t             43     1               if ((r.rci.fd = openfile(r.name, RT)) == ERROR)
rec_t             44     1                       return(ERROR);
rec_t             45     1               else
rec_t             46     1               {
rec_t             47     2                       fscanf(fp,"%ld %d\n", &r.rci.final_fo,&r.rci.final_bo);
rec_t             48     2
rec_t             49     2                                                  /*add node to tree*/
rec_t             50     2                       rt_root = rt_insert(rt_root, &r, &cond);
rec_t             51     2                       if (cond == MATCH)
rec_t             52     2                               return(error("unable to build record type tree"));
rec_t             53     2                       else
rec_t             54     2                               return(OK);
rec_t             55     2               }
rec_t             56     1       }
rec_t             57     0
rec_t             58     0
rec_t             59     0
rec_t             60     0
rec_t             61     0
```

```
file        line level  source

rec_t         63     0       /*Rt_insert inserts a record type into the record type b-tree.  If already
rec_t         64     0         present, condition is set to error. A pointer to the root is returned if
rec_t         65     0         if record type is inserted.*/
rec_t         66     0
rec_t         67     0       struct rec_type *rt_insert(prec, r, cond)
rec_t         68     0       struct rec_type *prec;
rec_t         69     0       struct rec_type *r;
rec_t         70     0       int *cond;
rec_t         71     0       {
rec_t         72     1       #ifdef DEBUG
rec_t         73     1       printf("e rt_insert\n");
rec_t         74     1       #endif
rec_t         75     1
rec_t         76     1               if(prec == NULL)
rec_t         77     1               {
rec_t         78     2                       prec = RTALLOC;
rec_t         79     2                       strcpy(prec->name,r->name);
rec_t         80     2                       prec->fdlim = r->fdlim;
rec_t         81     2                       prec->nosfield = r->nosfield;
rec_t         82     2                       prec->noskeyf = r->noskeyf;
rec_t         83     2                       arrcpy(prec->keyf,r->keyf,MAX_KEY_FIELDS);
rec_t         84     2                       prec->rci.fi = r->rci.fd;
rec_t         85     2                       prec->rci.log_fo = 0L;
rec_t         86     2                       prec->rci.phy_fo = 0L;
rec_t         87     2                       prec->rci.final_fo = r->rci.final_fo;
rec_t         88     2                       prec->rci.log_bo = 0;
rec_t         89     2                       prec->rci.phy_bo = 0;
rec_t         90     2                       prec->rci.final_bo = r->rci.final_bo;
rec_t         91     2                       prec->rci.add_on = FALSE;
rec_t         92     2                       prec->left = prec->right = NULL;
rec_t         93     2                       *cond = NO_MATCH;
rec_t         94     2               }
rec_t         95     1               else if ((*cond = strcmp(r->name,prec->name)) == MATCH)
rec_t         96     1                       error("record type already defined");
rec_t         97     1               else if (*cond < 0)             /*go to left subtree*/
rec_t         98     1                       prec->left = rt_insert(prec->left, r, cond);
rec_t         99     1               else                            /*go to right subtree*/
rec_t        100     1                       prec->right = rt_insert(prec->right, r, cond);
rec_t        101     1               return(prec);
rec_t        102     1       }
rec_t        103     0
rec_t        104     0
rec_t        105     0
rec_t        106     0
rec_t        107     0
```

```
file        line level  source

rec_t        109     0      /*Rect_add will add a record type to the data base*/
rec_t        110     0
rec_t        111     0      rect_add (rtname,rfdlim,nosfld,noskeyf,keyf)
rec_t        112     0      char *rtname;
rec_t        113     0      char rfdlim;
rec_t        114     0      int nosfld,noskeyf;
rec_t        115     0      int *keyf;
rec_t        116     0      {
rec_t        117     1              struct rec_type r;
rec_t        118     1              int i,cond;
rec_t        119     1              extern struct rec_type *rt_root;
rec_t        120     1              extern cnt_rt;
rec_t        121     1              extern long rbtroot;
rec_t        122     1              extern int rbtfd;
rec_t        123     1              long initbt();
rec_t        124     1              int verbose = NONE;
rec_t        125     1              struct rec_type *rt_insert();
rec_t        126     1              struct rec_type *rt_search();
rec_t        127     1
rec_t        128     1      #ifdef DEBUG
rec_t        129     1      printf ("e rect_add\n");
rec_t        130     1      #endif
rec_t        131     1
rec_t        132     1              if (cnt_rt == NONE)
rec_t        133     1              {               /*create btree for record keys and offsets*/
rec_t        134     2                      if((rbtfd = creat(RBT,PMODE)) == ERROR)
rec_t        135     2                              return(error("unable to create dbm.rec.bt file"));
rec_t        136     2                      buildfl(rbtfd,SIZE,verbose);
rec_t        137     2                      close(rbtfd);
rec_t        138     2                      if((rbtfd = open (RBT,RW)) == ERROR)
rec_t        139     2                              return(error("unable to open dbm.rec.bt file"));
rec_t        140     2                      rbtroot = initbt(rbtfd);
rec_t        141     2              }
rec_t        142     1              if (rt_search(rt_root,rtname) != NULL)
rec_t        143     1                      return(error("record type already defined"));
rec_t        144     1              if ((r.rci.fd = create_file(rtname,RT)) == EPROR)
rec_t        145     1                      return(ERROR);
rec_t        146     1              else
rec_t        147     1              {
rec_t        148     2                      strcpy(r.name, rtname);
rec_t        149     2                      r.fdlim = rfdlim;
rec_t        150     2                      r.nosfield = nosfld;
rec_t        151     2                      r.noskeyf = noskeyf;
rec_t        152     2                      arrcpy(r.keyf, keyf, MAX_KEY_FIELDS);
rec_t        153     2                      r.rci.final_fo = 0L;
rec_t        154     2                      r.rci.final_bo = 0;
rec_t        155     2                      rt_root = rt_insert(rt_root, &r, &cond);
rec_t        156     2                      if(cond == MATCH)
rec_t        157     2                              return(ERROR);
rec_t        158     2                      else
```

```
file          line level  source

rec_t          159     2                        {
rec_t          160     3                            cnt_rt += 1;
rec_t          161     3                            return(OK);
rec_t          162     3                        }
rec_t          163     2                    }
rec_t          164     1        }
rec_t          165     0
rec_t          166     0
rec_t          167     0
rec_t          168     0
rec_t          169     0
```

```
file          line level  source

rec_t          171     0      /*Rt_search searches the record_type b-tree for a specific occurrence of a
rec_t          172     0        record type name.  If present, a pointer to the set type is returned.
rec_t          173     0        Otherwise, null is returned*/
rec_t          174     0
rec_t          175     0      struct rec_type *rt_search(prec,rname)
rec_t          176     0      struct rec_type *prec;
rec_t          177     0      char *rname;
rec_t          178     0      {
rec_t          179     1              int cond;
rec_t          180     1
rec_t          181     1      #ifdef DEBUG
rec_t          182     1      printf("e rt_search\n");
rec_t          183     1      #endif
rec_t          184     1
rec_t          185     1              if (prec == NULL)
rec_t          186     1                      return(NULL);
rec_t          187     1              else if ((cond = strcmp(rname,prec->name)) == MATCH)
rec_t          188     1                      return(prec);                   /*name found - return pointer*/
rec_t          189     1              else if (cond < 0)                      /*look in left sub tree*/
rec_t          190     1                      return(rt_search(prec->left,rname));
rec_t          191     1              else                                    /*look in right sub tree*/
rec_t          192     1                      return(rt_search(prec->right,rname));
rec_t          193     1      }
rec_t          194     0
rec_t          195     0
rec_t          196     0
rec_t          197     0
rec_t          198     0
```

```
file            line level  source

rec_t            200     0      /*Rt_write saves information on the record types by writing it to the DBMF.*/
rec_t            201     0
rec_t            202     0
rec_t            203     0      rt_write(fp,prec,cnt)
rec_t            204     0      FILE *fp;
rec_t            205     0      struct rec_type *prec;
rec_t            206     0      int *cnt;
rec_t            207     0      {
rec_t            208     1              int i;
rec_t            209     1
rec_t            210     1      #ifdef DEBUG
rec_t            211     1      printf("e rt_write\n");
rec_t            212     1      #endif
rec_t            213     1
rec_t            214     1              if (prec == NULL)
rec_t            215     1                      return;
rec_t            216     1              else
rec_t            217     1              {
rec_t            218     2              if (prec->rci.add_on)           /*flush buffer*/
rec_t            219     2                      write_buf(prec->rci.fd,prec->rci.final_fo,prec->rci.file_buf,
rec_t            220     2                                      prec->rci.final_bo);
rec_t            221     2              fprintf (fp, "%s %c %d %d ", prec->name,prec->fdlim,prec->nosfield, prec->noskeyf);
rec_t            222     2              for (i=0; i<prec->noskeyf; i++)
rec_t            223     2                      fprintf(fp,"%d ",*(prec->keyf+i));
rec_t            224     2              fprintf(fp, "%ld %d\n", prec->rci.final_fo, prec->rci.final_bo);
rec_t            225     2              *cnt += 1;
rec_t            226     2              rt_write (fp,prec->left,cnt);
rec_t            227     2              rt_write (fp,prec->right,cnt);
rec_t            228     2              return;
rec_t            229     2              }
```

file	line	level	source
rec_t	231	1	}

```
file          line level  source

set_t            1     0       /*This file contains the routines to build the set type tree, insert a set
set_t            2     0         type into the tree, and search the tree for a particular set type.
set_t            3     0         A routine to write the information in the set type tree out to the DBM file
set_t            4     0         is included.*/
set_t            5     0
set_t            6     0       /*MAD 10-81*/
set_t            7     0
set_t            8     0       /*Bset_tree reads the data base management file and builds a tree
set_t            9     0         containing the information on the different set types in the data base*/
set_t           10     0
set_t           11     0       bset_tree (fp, number)
set_t           12     0       int *fp;
set_t           13     0       int number;
set_t           14     0       {
set_t           15     1               int i;
set_t           16     1
set_t           17     1       #ifdef DEBUG
set_t           18     1       printf("e bset_tree\n");
set_t           19     1       #endif
set_t           20     1
set_t           21     1               for (i=1;i<=number; i++)
set_t           22     1                       if (aset_tt (fp) == ERROR)
set_t           23     1                               return(ERROR);
set_t           24     1               return(OK);
set_t           25     1       }
set_t           26     0
set_t           27     0
set_t           28     0
set_t           29     0
set_t           30     0
```

```
file            line level  source

set_t             32     0       /*Aset_tt will scan the dbm file and build a set type and place it in the
set_t             33     0         set type b-tree.  Returns -1 if error occurs*/
set_t             34     0
set_t             35     0       aset_tt (fp)            /*dbm file pointer*/
set_t             36     0       FILE *fp;
set_t             37     0       {
set_t             38     1               int i,cond;
set_t             39     1               char nowner[MAX_TNAME + 1];
set_t             40     1               char nmember[MAX_TNAME + 1];
set_t             41     1               extern struct set_type *st_root;
set_t             42     1               extern struct rec_type *rt_root;
set_t             43     1               struct set_type s;
set_t             44     1               struct set_type *st_insert();
set_t             45     1               struct rec_type *rt_search();
set_t             46     1
set_t             47     1       #ifdef DEBUG
set_t             48     1       printf("e aset_tt\n");
set_t             49     1       #endif
set_t             50     1
set_t             51     1                               /*read information for one set type*/
set_t             52     1               fscanf(fp,"%s %s %s %ld %d\n",s.sname,nowner,nmember,&s.sci.final_fo,
set_t             53     1                       &s.sci.final_bo);
set_t             54     1               if ((s.owner = rt_search(rt_root,nowner)) == NULL  ||
set_t             55     1                       (s.member = rt_search(rt_root,nmember)) == NULL)
set_t             56     1                       return(error("unable to build set type - owner/member not a record type"));
set_t             57     1               if ((s.sci.fd = openfile(s.sname, ST)) == ERROR ||
set_t             58     1                       (s.sbtfd =openfile(s.sname, BT)) == ERROR)
set_t             59     1                       return(ERROR);
set_t             60     1               else
set_t             61     1               {
set_t             62     2                       st_root = st_insert(st_root, &s, &cond);
set_t             63     2                       if (cond == MATCH)
set_t             64     2                               return(error("unable to build set type tree"));
set_t             65     2                       else
set_t             66     2                               return(OK);
set_t             67     2               }
set_t             68     1       }
set_t             69     0
set_t             70     0
set_t             71     0
set_t             72     0
set_t             73     0
```

```
file          line level  source

set_t           75     0       /*St_insert inserts a new set type into the set type b-tree.  If already
set_t           76     0         present, condition is set to match(0).  A pointer to the root is returned*/
set_t           77     0
set_t           78     0
set_t           79     0       struct set_type *st_insert(pset, s, cond)
set_t           80     0       struct set_type *pset;
set_t           81     0       struct set_type *s;
set_t           82     0       int *cond;
set_t           83     0       {
set_t           84     1               long initbt();
set_t           85     1               if (pset == NULL)
set_t           86     1               {                                       /*build and add a new set type*/
set_t           87     2                       pset = STALLOC;
set_t           88     2                       strcpy(pset->sname,s->name);
set_t           89     2                       pset->owner = s->owner;
set_t           90     2                       pset->member = s->member;
set_t           91     2                       *pset->cos = NULL;
set_t           92     2                       pset->sci.fd = s->sci.fd;
set_t           93     2                       pset->sbtfd = s->sbtfd;
set_t           94     2                       pset->sbtroot = initbt(s->sbtfd);
set_t           95     2                       pset->sci.log_fo = 0;
set_t           96     2                       pset->sci.phy_fo = 0;
set_t           97     2                       pset->sci.final_fo = s->sci.final_fo;
set_t           98     2                       pset->sci.log_bo = 0;
set_t           99     2                       pset->sci.phy_bo = 0;
set_t          100     2                       pset->sci.final_bo = s->sci.final_bo;
set_t          101     2                       pset->sci.add_on = 0;
set_t          102     2                       pset->sleft = NULL;
set_t          103     2                       pset->sright = NULL;
set_t          104     2                       *cond = NO_MATCH;
set_t          105     2               }
set_t          106     1               else if ((*cond = strcmp(s->sname,pset->sname)) == MATCH)
set_t          107     1                       error("set type already defined");
set_t          108     1               else if (*cond < 0)                     /*move left to place set*/
set_t          109     1                       pset->sleft = st_insert(pset->sleft, s, cond);
set_t          110     1               else                                    /*move right to place set*/
set_t          111     1                       pset->sright = st_insert(pset->sright, s, cond);
set_t          112     1               return(pset);
set_t          113     1       }
set_t          114     0
set_t          115     0
set_t          116     0
set_t          117     0
set_t          118     0
```

```
file    line level  source

set_t    120    0      /*Sett_add will add a set type to the data base.*/
set_t    121    0
set_t    122    0
set_t    123    0      sett_add(stname,owner,member)
set_t    124    0      char *stname, *owner, *member;
set_t    125    0      {
set_t    126    1              extern struct rec_type *rt_root;
set_t    127    1              extern struct set_type *st_root;
set_t    128    1              struct set_type *st_search();
set_t    129    1              extern cnt_st;
set_t    130    1              struct set_type s;
set_t    131    1              struct set_type *st_insert();
set_t    132    1              struct rec_type *rt_search();
set_t    133    1              int cond,fd;
set_t    134    1              char *concatn();
set_t    135    1              char newn[MAX_TNAME + APN + 1];
set_t    136    1              int verbose=NONE;
set_t    137    1
set_t    138    1      #ifdef DEBUG
set_t    139    1      printf("e sett_add \n");
set_t    140    1      #endif
set_t    141    1
set_t    142    1              if ((s.owner = rt_search(rt_root,owner)) == NULL ||
set_t    143    1                      (s.member = rt_search(rt_root,member)) == NULL)
set_t    144    1                      return(error("unable to build set type - owner/member not a record type"));
set_t    145    1              if (st_search(st_root,stname) != NULL)
set_t    146    1                      return(error("set type already defined"));
set_t    147    1              if ((s.sc1.fd = create_file(stname, ST)) == ERROR)
set_t    148    1                      return(ERROR);
set_t    149    1              concatn(newn,stname,BT);
set_t    150    1              if ((s.sbtfd = creat(newn,PMODE)) ==ERROR)
set_t    151    1                      return(error("cannot create the set btree"));
set_t    152    1              buildfl(s.sbtfd,SIZE,verbose);
set_t    153    1              close(s.sbtfd);
set_t    154    1              if ((s.sbtfd = open(newn,RW)) == ERROR)
set_t    155    1                      return(error("cannot open the set btree"));
set_t    156    1              else
set_t    157    1              {
set_t    158    2                      strcpy(s.name, stname);
set_t    159    2                      s.sc1.final_fo = 0L;
set_t    160    2                      s.sc1.final_bo = 0;
set_t    161    2                      st_root = st_insert(st_root, &s, &cond);
set_t    162    2                      if (cond == MATCH)
set_t    163    2                              return(ERROR);
set_t    164    2                      else
set_t    165    2                      {
set_t    166    3                              cnt_st +=1;
set_t    167    3                              return(OK);
set_t    168    3                      }
```

file	line	level	source
set_t	170	1	}
set_t	171	0	
set_t	172	0	
set_t	173	0	
set_t	174	0	
set_t	175	0	

```
file        line level  source

set_t        177     0      /*St_search searches the set type b-tree for a specific occurrence by
set_t        178     0        set type name.  If present, a pointer to the set type is returned.
set_t        179     0        Otherwise, null is returned*/
set_t        180     0
set_t        181     0      struct set_type *st_search(pset,name)
set_t        182     0      struct set_type *pset;
set_t        183     0      char *name;
set_t        184     0      {
set_t        185     1              int cond;
set_t        186     1      #ifdef DEBUG
set_t        187     1      printf("e st_search\n");
set_t        188     1      #endif
set_t        189     1
set_t        190     1              if (pset == NULL)
set_t        191     1                      return(NULL);           /*set not in tree*/
set_t        192     1              else if ((cond = strcmp(name,pset->sname)) == MATCH)
set_t        193     1                      return(pset);           /*match found*/
set_t        194     1              else if (cond < 0)              /*look left*/
set_t        195     1                      return(st_search(pset->sleft,name));
set_t        196     1              else                            /*look right*/
set_t        197     1                      return(st_search(pset->sright,name));
set_t        198     1      }
set_t        199     0
set_t        200     0
set_t        201     0
set_t        202     0
set_t        203     0
```

```
file            line level  source

set_t            205     0      /*Write_st saves information on the set types by writing it to the DBMF.*/
set_t            206     0
set_t            207     0
set_t            208     0      st_write(fp,pset,cnt)
set_t            209     0      FILE *fp;
set_t            210     0      struct set_type *pset;
set_t            211     0      int *cnt;
set_t            212     0      {
set_t            213     1              int i;
set_t            214     1              struct rec_type *p;
set_t            215     1              char *owner;
set_t            216     1              char *member;
set_t            217     1
set_t            218     1      #ifdef DEBUG
set_t            219     1      printf("e write_st\n");
set_t            220     1      #endif
set_t            221     1
set_t            222     1              if (pset == NULL)
set_t            223     1                      return;
set_t            224     1              else
set_t            225     1              {
set_t            226     2                      if (pset->sci.add_on)                   /*flush buffer*/
set_t            227     2                              write_buf(pset->sci.fd,pset->sci.final_fo,
set_t            228     2                              pset->sci.file_buf,pset->sci.final_bo);
set_t            229     2                      p = pset->owner;
set_t            230     2                      close(pset->sbtfd);
set_t            231     2                      owner = p->name;                /*get name of owner record type*/
set_t            232     2                      p = pset->member;
set_t            233     2                      member = p->name;               /*get name of member record type*/
set_t            234     2                      fprintf(fp,"%s %s %s %ld %d\n",pset->sname,owner,member,
set_t            235     2                              pset->sci.final_fo,pset->sci.final_bo);
set_t            236     2                      *cnt +=1;
set_t            237     2                      st_write(fp,pset->sleft,cnt);
set_t            238     2                      st_write(fp,pset->sright,cnt);
set_t            239     2                      return;
set_t            240     2              }
set_t            241     1      }
```

file line level source

```
file          line level  source

init_concl      1     0       /*This file contains the routines to initialize the data base management system
init_concl      2     0         and to store information on record and set types when use of system concludes.*/
init_concl      3     0
init_concl      4     0
init_concl      5     0       /*MAD 10-81*/
init_concl      6     0
init_concl      7     0       /*Init opens the data base management file and sets up the record type b-tree
init_concl      8     0         and the set type b-tree.*/
init_concl      9     0
init_concl     10     0
init_concl     11     0       init()
init_concl     12     0       {
init_concl     13     1               FILE *fopen(),*fp,fclose();
init_concl     14     1               extern int cnt_rt;
init_concl     15     1               extern long rbtroot;
init_concl     16     1               extern int rbtfd;
init_concl     17     1               extern int cnt_st;
init_concl     18     1               long initbt();
init_concl     19     1
init_concl     20     1       #ifdef DEBUG
init_concl     21     1       printf("e init\n");
init_concl     22     1       #endif
init_concl     23     1
init_concl     24     1               if ((fp = fopen(DBMF, "r")) == NULL)
init_concl     25     1               {
init_concl     26     2                       printf("record and set types have not been defined\n");
init_concl     27     2                       return(OK);
init_concl     28     2               }
init_concl     29     1               fscanf(fp,"%d %d\n", &cnt_rt, &cnt_st);
init_concl     30     1               if (brec_tree (fp,cnt_rt) == ERROR)
init_concl     31     1                       return(ERROR);
init_concl     32     1               if (bset_tree (fp,cnt_st) == ERROR)
init_concl     33     1                       return(ERROR);
init_concl     34     1               fclose (fp);
init_concl     35     1               if ((rbtfd = open(RBT,RW)) == ERROR)            /*open and set record btree*/
init_concl     36     1                       return(error("unable to open dbm.rec.bt"));
init_concl     37     1               rbtroot = initbt(rbtfd);
init_concl     38     1               return(OK);
init_concl     39     1       }
init_concl     40     0
init_concl     41     0
init_concl     42     0
init_concl     43     0
init_concl     44     0
```

```
file          line level  source

init_concl      46     0       /*Store_dbmf stores the information about record types and set types on the
init_concl      47     0         data base management file.*/
init_concl      48     0
init_concl      49     0
init_concl      50     0       store_dbmf()
init_concl      51     0       {
init_concl      52     1               FILE *fopen(),*fp;
init_concl      53     1               int cnt;
irit_concl      54     1               extern int cnt_rt;
init_concl      55     1               extern int cnt_st;
init_concl      56     1               extern struct rec_type *rt_root;
init_concl      57     1               extern struct set_type *st_root;
init_concl      58     1               extern int rbtfd;
init_concl      59     1
init_concl      60     1       #ifdef DEBUG
init_concl      61     1       printf("e store_dbmf");
init_concl      62     1       #endif
init_concl      63     1
init_concl      64     1               fp = fopen(DBMF,"w");
init_concl      65     1               fprintf(fp,"%d %d\n",cnt_rt,cnt_st);
init_concl      66     1               cnt = 0;
init_concl      67     1               rt_write(fp,rt_root,&cnt);
init_concl      68     1               if (cnt != cnt_rt)
init_concl      69     1                       fprintf(stderr,"error during write of record tree on DBMF\n");
init_concl      70     1               cnt = 0;
init_concl      71     1               st_write(fp,st_root,&cnt);
init_concl      72     1               if (cnt != cnt_st)
init_concl      73     1                       fprintf(stderr,"error during write of set tree on DBMF\n");
init_concl      74     1               fclose (fp);
init_concl      75     1               close (rbtfd);
init_concl      76     1               return;
init_concl      77     1       }
```

file line level source

```
file            line level  source

utility            1     0
utility            2     0        /*Utility contains functions that serve as utilities for the main program
utility            3     0          segments.*/
utility            4     0
utility            5     0
utility            6     0        /*Error prints and error message and returns an error flag.*/
utility            7     0
utility            8     0        /*MAD 10/81*/
utility            9     0
utility           10     0        error(msg)
utility           11     0        char msg[];              /*error message*/
utility           12     0        {
utility           13     1                fprintf(stderr,"%s\n",msg);
utility           14     1                return(ERROR);
utility           15     1        }
utility           16     0
utility           17     0
```

```
file            line level  source

utility           19     0        /*kerror prints an error message and the accompanying key and returns an error
utility           20     0          flag.*/
utility           21     0
utility           22     0        kerror(msg,key)
utility           23     0        char *msg;                /*error message*/
utility           24     0        char *key;
utility           25     0        {
utility           26     1                fprintf(stderr,"%s Key = %s\n",msg,key);
utility           27     1                return(ERROR);
utility           28     1        }
utility           29     0
utility           30     0
utility           31     0
utility           32     0
utility           33     0
utility           34     0
```

```
file            line level  source

utility           36     0        /*Rec_str replaces the end of record mark with a null bit and therefore, in
utility           37     0                essence changes a record to a string.*/
utility           38     0
utility           39     0        /*MAD 10-81*/
utility           40     0
utility           41     0        rec_str(buf)
utility           42     0        char *buf;
utility           43     0        {
utility           44     1                int i;
utility           45     1                for (i=0; i< MAX_RECORD; i++)
utility           46     1                        if(*(buf+i) == END_OF_RECORD)
utility           47     1                        {
utility           48     2                                *(buf+1) = NULL;
utility           49     2                                return;
utility           50     2                        }
utility           51     1                return(error("record too long"));
utility           52     1        }
utility           53     0
utility           54     0
utility           55     0
```

```
file            line level  source

utility           57     0       /*Str_rec replaces the null bit at end of string with an end of record mark
utility           58     0              and in essence changes a string to a record.*/
utility           59     0
utility           60     0       /*MAD 10-81*/
utility           61     0
utility           62     0       str_rec(buf)
utility           63     0       char *buf;
utility           64     0       {
utility           65     1               int i;
utility           66     1               for (i=0; i < MAX_RECORD; i++)
utility           67     1                       if (*(buf+i) == NULL)
utility           68     1                       {
utility           69     2                               *(buf+i) = END_OF_RECORD;
utility           70     2                               return;
utility           71     2                       }
utility           72     1               return(error("string too long"));
utility           73     1       }
utility           74     0
utility           75     0
utility           76     0
utility           77     0
utility           78     0
```

```
file            line level  source

utility           80     0       /*Arrcpy copies one array of integers to another array.*/
utility           81     0
utility           82     0       /*MAD 10-81*/
utility           83     0
utility           84     0       arrcpy (top,fromp,size)
utility           85     0       int *top,*fromp;                    /*array copied to and from*/
utility           86     0       int size;                           /*size of array to be copied*/
utility           87     0       {
utility           88     1               int i;
utility           89     1               for (i=0; i< size; i++)
utility           90     1                       *top++ = *fromp++;
utility           91     1       }
utility           92     0
utility           93     0
utility           94     0
utility           95     0
utility           96     0
```

```
file            line level  source

utility           98     0        /*Create_file will create and then open a file.  The file opened will have a
utility           99     0          name consisting of a name with a string appended to the end of it.*/
utility          100     0
utility          101     0        /*MAD 10-81*/
utility          102     0
utility          103     0        create_file (filen, append)
utility          104     0        char *filen,*append;
utility          105     0        {
utility          106     1                int fd;
utility          107     1                char catfname[MAX_TNAME + APN + 1];
utility          108     1                strcpy (catfname,filen);
utility          109     1                strcat(catfname,append);
utility          110     1                if ((fd = creat(catfname,PMODE)) == ERROR)
utility          111     1                        return(error("unable to create file"));
utility          112     1                else if ((fd = open(catfname,RW))== ERROR)
utility          113     1                        return(error("unable to open file"));
utility          114     1                else
utility          115     1                        return(fd);
utility          116     1        }
utility          117     0
utility          118     0
utility          119     0
utility          120     0
utility          121     0
```

```
file            line level  source

utility          123     0      /*Openfile opens a character file for read/write options.  The filename is
utility          124     0        a combination of a name and a string appended to the end of the name.
utility          125     0        If no error occurs, a file descriptor is returned.*/
utility          126     0
utility          127     0      /*MAD 10-81*/
utility          128     0
utility          129     0      openfile (filen, append)
utility          130     0      char *filen, *append;
utility          131     0      {
utility          132     1              int fd;
utility          133     1              char catfname[MAX_TNAME + APN + 1];
utility          134     1              strcpy (catfname,filen);
utility          135     1              strcat (catfname,append);
utility          136     1              if ((fd = open(catfname,RW)) == ERROR)
utility          137     1                      return(error("unable to open file\n"));
utility          138     1              else
utility          139     1                      return(fd);
utility          140     1      }
utility          141     0
utility          142     0
utility          143     0
utility          144     0
utility          145     0
```

```
file            line level  source

utility          147     0      /* Concatn concatenates two strings. */
utility          148     0
utility          149     0      char *concatn(newstring,bstring,estring)
utility          150     0      char *newstring,*bstring,*estring;
utility          151     0      {
utility          152     1              strcpy(newstring, bstring);
utility          153     1              strcat(newstring, estring);
utility          154     1              return(newstring);
utility          155     1      }
```

file line level source

```
file          line level  source

io              1     0
io              2     0       /*Read_rec reads a record from the file buffer into the record input/output
io              3     0         buffer.*/
io              4     0
io              5     0       /*MAD 10/81*/
io              6     0
io              7     0       read_rec(f_offset,b_offset,rec_io,pci)
io              8     0
io              9     0       long f_offset;                  /*file offset of record to be read*/
io             10     0       int b_offset;                   /*block offset                    */
io             11     0       char *rec_io;                   /*pointer to record input/output buffer*/
io             12     0       struct ci *pci;                 /*currency information for rec/set type*/
io             13     0                       /*log_fo, log_bo points at start of current of record type.
io             14     0                         phy_fo, phy_bo points at end of current of record type.
io             15     0                         final_fo, final_bo points at next place on file to add a
io             16     0                            record. */
io             17     0       {
io             18     1               int size;
io             19     1
io             20     1       #ifdef DEBUG
io             21     1       printf("entering read_rec\n");
io             22     1       #endif
io             23     1
io             24     1               if (f_offset < 0 || b_offset < 0 || pci->fd < 0)
io             25     1                       return(error("Illegal input to read_rec"));
io             26     1               if (pci->phy_fo == 0L && pci->log_fo == 0L && pci->phy_bo == 0 &&
io             27     1                               pci->log_bo == 0 && pci->add_on == FALSE)
io             28     1                                       /*prime buffer the first time through*/
io             29     1               {
io             30     2                       pci->phy_fo = pci->final_fo;
io             31     2                       pci->phy_bo = pci->final_bo;
io             32     2                       read_buf(pci->fd, pci->phy_fo, pci->file_buf, pci->phy_bo);
io             33     2               }
io             34     1                       if(f_offset != pci->phy_fo && pci->add_on)
io             35     1                       {               /*write to file previously added record*/
io             36     2                               if(pci->final_bo == write_buf(pci->fd,pci->final_fo,pci->file_buf,pci->final
io             37     2                                       pci->add_on = FALSE;
io             38     2                               else
io             39     2                                       return(error("write incomplete"));
io             40     2                       }
io             41     1                       pci->log_fo = f_offset;
io             42     1                       if (f_offset != pci->phy_fo)
io             43     1                       {               /*reset currency pointers*/
io             44     2                               pci->phy_fo = f_offset;
io             45     2                               if (pci->phy_fo != pci->final_fo)
io             46     2                                       size = BLOCKSIZE;
io             47     2                               else
io             48     2                                       size = pci->final_bo;
io             49     2                               if (read_buf(pci->fd,pci->phy_fo,pci->file_buf,size) == ERROR)
io             50     2                                       return(error("unable to read input file properly"));
```

```
file            line level  source

io                51     2                    }
io                52     1                    if (pci->add_on && b_offset >= pci->final_bo || (f_offset == pci->final_fo && b_offs
io                53     1                          return(error("unable to read input-record never written "));
io                54     1                    pci->log_bo = pci->phy_bo = b_offset;
io                55     1                    return(copy_in(rec_io,rec_io+MAX_RECORD,pci));
io                56     1          }
io                57     0
io                58     0
io                59     0
io                60     0
io                61     0
```

```
file            line level  source

io                63     0        /*Copy_in copies a record, character by character, from the file buffer into
io                64     0          the record input/output buffer.*/
io                65     0
io                66     0        /*MAD 10/81*/
io                67     0
io                68     0        copy_in(rec_io,max_io,pci)
io                69     0        char    *rec_io;                              /*   "                  */
io                70     0        char    *max_io;                              /*   "     to record input/output buffer*/
io                71     0        struct ci *pci;                               /*   "     to 1st char beyond i/o buffer*/
io                72     0                                                      /*currency pointers for rec/set type*/
io                73     0        {                                                    .
io                74     1                int size;                             /*move record to i/o buffer*/
io                75     1                                                      /*number of characters to read*/
io                76     1        #ifdef DEBUG
io                77     1        printf ("e copy_in\n");
io                78     1        #endif
io                79     1
io                80     1                while (pci->phy_bo < BLOCKSIZE &&
io                81     1                        (*rec_io++ = *(pci->file_buf+(pci->phy_bo)++)) != END_OF_RECORD &&
io                82     1                        rec_io < max_io)
io                83     1                        ;
io                84     1                if (rec_io >= max_io)
io                85     1                        return(error("record to large for input buffer"));
io                86     1                else
io                87     1                        if (pci->phy_bo == BLOCKSIZE)
io                88     1                        {
io                89     2                                /*need to read next block from file*/
io                90     2                                pci->phy_fo += BLOCKSIZE;
io                91     2                                pci->phy_bo = 0;
io                92     2                                if(pci->phy_fo != pci->final_fo)
io                93     2                                        size = BLOCKSIZE;
io                94     2                                else
io                95     2                                        size = pci->final_bo;
io                96     2                                if (read_buf(pci->fd,pci->phy_fo,pci->file_buf,size) != ERROR)
io                97     2                                        return(copy_in(rec_io,max_io,pci));
io                98     2                                else
io                99     2                                        return(error("read into buffer incomplete"));
io               100     1                        }
io               101     1                        else
io               102     1                                return(OK);
io               103     0        }
io               104     0
io               105     0
io               106     0
io               107     0
```

```
file        line level  source

io           109     0      /*Read_buf reads one block from a file into a buffer*/
io           110     0
io           111     0      /*MAD 10/81 */
io           112     0
io           113     0      read_buf (fd,offset,file_buf,size)
io           114     0      int fd;                          /*file descriptor for input file*/
io           115     0      long offset;                     /*offset of desired block of file*/
io           116     0      char *file_buf;                  /*file input buffer*/
io           117     0      int size;                        /*number of characters to read*/
io           118     0
io           119     0      {
io           120     1      #ifdef DEBUG
io           121     1      char *i;
io           122     1      printf("read=\n");
io           123     1      for (i=file_buf;i<file_buf+size;i++)
io           124     1      printf("%c",*i);
io           125     1      printf("\n");
io           126     1      #endif
io           127     1
io           128     1              lseek(fd,offset,START); /*move to offset in file*/
io           129     1              return(read(fd,file_buf,size));
io           130     1      }
io           131     0
io           132     0
io           133     0
io           134     0
io           135     0
```

```
file            line level  source

io               137     0      /*Write_rec writes one record from the input/output buffer to the file buffer*/
io               138     0
io               139     0      /*MAD 10/81*/
io               140     0
io               141     0      write_rec (f_offset,b_offset,rec_io,pci)
io               142     0      long *f_offset;          /*pointer to file offset of record to be written*/
io               143     0      int *b_offset;           /*pointer to block offset                       */
io               144     0      char *rec_io;            /*pointer to record input/output buffer*/
io               145     0      struct ci *pci;                  /*currency information for rec/set type*/
io               146     0
io               147     0                     /*log_fo, log_bo points at start of current of record type.
io               148     0                       phy_fo, phy_bo points at end of current of record type.
io               149     0                       final_fo, final_bo points at next place on file to add
io               150     0                       a record.*/
io               151     0
io               152     0      {
io               153     1              int cond;
io               154     1      #ifdef DEBUG
io               155     1      printf("entering write_rec\n");
io               156     1      #endif
io               157     1
io               158     1              if (*f_offset == NEWL)
io               159     1              {
io               160     2                                      /*add new record to file*/
io               161     2                      pci->log_fo = pci->phy_fo = *f_offset = pci->final_fo;
io               162     2                      pci->log_bo = pci->phy_bo = *b_offset = pci->final_bo;
io               163     2                      if (!pci->add_on)                 /*final record not presently in core*/
io               164     3                      {
io               165     3                              if (pci->phy_fo != pci->final_fo)
io               166     3                                      cond = read_buf(pci->fd,pci->phy_fo,pci->file_buf,BLOCKSIZE);
io               167     3                              else
io               168     3                                      cond = read_buf(pci->fd,pci->phy_fo,pci->file_buf,pci->final_bo);
io               169     3                              if (cond != ERROR ||
io               170     3                                (cond == ERROR && pci->final_fo == START && pci->final_bo == START))
io               171     3                                      pci->add_on = TRUE;
io               172     3                              else
io               173     3                                      return(error("can not read output file"));
io               174     2                      }
io               175     2                      copy_out(rec_io, rec_io+MAX_RECORD, pci);
io               176     2                      if (pci->add_on)
io               177     3                      {
io               178     3                              pci->final_fo = pci->phy_fo;
io               179     3                              pci->final_bo = pci->phy_bo;
io               180     2                      }
io               181     2                      return (OK);
io               182     1              }
io               183     1              else
io               184     2                      {                       /*write over existing record*/
io               185     2                      if (pci->add_on && pci->final_fo != *f_offset)          /*write out new record left
io               186     2                              if (write_buf(pci->fd,pci->final_fo,pci->file_buf,pci->final_bo) != pci->fin
                                                               return(error("file write improperly done"));
```

```
file            line level  source

io               187     2                              else
io               188     2                                      pci->add_on = FALSE;
io               189     2                      if (pci->phy_fo != *f_offset)
io               190     2                      {
io               191     3                              if (*f_offset != pci->final_fo)
io               192     3                                      cond = read_buf(pci->fd,f_offset,pci->file_buf,BLOCKSIZE);
io               193     3                              else
io               194     3                                      cond = read_buf(pci->fd,f_offset,pci->file_buf,pci->final_bo);
io               195     3                              if (cond == ERROR)
io               196     3                                      return(error("file read error"));
io               197     3                      }
io               198     2                      pci->log_fo = pci->phy_fo = *f_offset;
io               199     2                      pci->log_bo = pci->phy_bo = *b_offset;
io               200     2                      copy_out(rec_io,rec_io+MAX_RECORD,pci);
io               201     2                      if (pci->phy_fo != pci->final_fo)       /*write updated record*/
io               202     2                              cond = write_buf(pci->fd,pci->phy_fo,pci->file_buf,BLOCKSIZE);
io               203     2                      else
io               204     2                              cond = write_buf(pci->fd,pci->phy_fo,pci->file_buf,pci->final_bo);
io               205     2                      if (cond == BLOCKSIZE || cond == pci->final_bo)
io               206     2                              return (OK);
io               207     2                      else
io               208     2                              return(error("write not completed"));
io               209     2              }
io               210     1      }
io               211     0
io               212     0
io               213     0
io               214     0
io               215     0
```

```
file            line level  source

io               217     0      /*Copy_out copies a record from an input/output area and places it in the file
io               218     0        buffer.  If need be, the file buffer is written out and a new one is made
io               219     0        available*/
io               220     0
io               221     0      /*MAD 10/81*/
io               222     0
io               223     0      copy_out (rec_io,max_io,pci)
io               224     0      char *rec_io;           /*  "       "  record input/output buffer */
io               225     0      char *max_io;           /*  "       "  1st character past record i/o buffer*/
io               226     0      struct ci *pci;         /*currency information for record/set type*/
io               227     0      {
io               228     1              int cond;
io               229     1              char *i;
io               230     1
io               231     1      #ifdef DEBUG
io               232     1      printf("entering copy_out\n");
io               233     1      #endif
io               234     1
io               235     1              for (i=rec_io; i<max_io && *i != END_OF_RECORD; ++i)
io               236     1                      ;                       /*error check record size*/
io               237     1              if (i == max_io)
io               238     1                      return(error("output record to large"));
io               239     1              while (pci->phy_bo < BLOCKSIZE &&
io               240     1                      (*(pci->file_buf + (pci->phy_bo)++) = *rec_io++) != END_OF_RECORD)
io               241     1                      ;
io               242     1              if (*(rec_io-1) = END_OF_RECORD && pci->phy_bo != BLOCKSIZE)
io               243     1              {
io               244     2                      if (! pci->add_on)              /*write revised record*/
io               245     2                      {
io               246     3                              if (pci->phy_fo != pci->final_fo)
io               247     3                                      if(BLOCKSIZE == write_buf(pci->fd,pci->phy_fo,pci->file_buf,BLOCKSIZ
io               248     3                                              return(OK);
io               249     3                              else
io               250     3                                      if (pci->final_bo == write_buf(pci->fd,pci->phy_fo,pci->file_buf,pci
io               251     3                                              return(OK);
io               252     3                              return(error("write not completed"));
io               253     3                      }
io               254     2              }
io               255     1              else if (BLOCKSIZE == write_buf(pci->fd,pci->phy_fo,pci->file_buf,BLOCKSIZE))
io               256     1              {                               /*write buffer area to output*/
io               257     2                      pci->phy_fo += BLOCKSIZE;
io               258     2                      pci->phy_bo = 0;
io               259     2                      if ( pci->add_on)
io               260     2                      {
io               261     3                              pci->final_fo = pci->phy_fo;
io               262     3                              pci->final_bo = 0;
io               263     3                      }
io               264     2                      else
io               265     2                      {
io               266     3                              if(pci->phy_fo != pci->final_fo)
```

```
file            line level  source

io               267     3                                  cond = read_buf(pci->fd,pci->phy_fo,pci->file_buf,BLOCKSIZE);
io               268     3                          else
io               269     3                                  cond = read_buf(pci->fd,pci->phy_fo,pci->file_buf,pci->final_bo);
io               270     3                          if (cond == ERROR)
io               271     3                                  return(error("read incomplete"));
io               272     3                  }
io               273     2                  return(copy_out(rec_io,max_io,pci));
io               274     2          }
io               275     1          else
io               276     1                  return(error("incomplete write to output file"));
io               277     1      }
io               278     0
io               279     0
io               280     0
io               281     0
```

```
file            line level  source

io               283     0      /*Write_buf writes one block onto a file from a buffer*/
io               284     0
io               285     0      /*MAD 10/81*/
io               286     0
io               287     0      write_buf (fd,offset,file_buf,size)
io               288     0      int fd;                              /*file descriptor for output file*/
io               289     0      long offset;                         /*offset of desired block on file*/
io               290     0      char *file_buf;                      /*buffer for file*/
io               291     0      int size;
io               292     0      {
io               293     1      #ifdef DEBUG
io               294     1      char *i;
io               295     1      printf("write=\n");
io               296     1      for(i=file_buf; i<file_buf+size; i++)
io               297     1      printf("%c",*i);
io               298     1      printf("\n");
io               299     1      #endif
io               300     1              lseek(fd,offset,START);          /*move to correct position in file*/
io               301     1              return(write(fd,file_buf,size));
io               302     1      }
```

file line level source

```
file          line level  source

dbup_c_dec       1     0
dbup_c_dec       2     0       /* These variables are used in the dbup routines. */
dbup_c_dec       3     0
dbup_c_dec       4     0       #include "ext_dec"
dbup_c_dec       5     0       char iobuf[MAX_RECORD + 2];
dbup_c_dec       6     0       char key[MAX_KEY + 1];
dbup_c_dec       7     0       char fp[MAX_KEY +1];
dbup_c_dec       8     0       char bp[MAX_KEY + 1];
dbup_c_dec       9     0       char savekey[MAX_KEY + 1];
dbup_c_dec      10     0       char savenext[MAX_KEY + 1];
dbup_c_dec      11     0       char unit[MAX_KEY + 1];
dbup_c_dec      12     0       struct ptrs os,saveos,saveos2;
dbup_c_dec      13     0       struct rec_type *rt;
dbup_c_dec      14     0       struct set_type *st;
```

```
file            line level  source

dbup               1     0       /*MAD 11/81*/
dbup               2     0
dbup               3     0
dbup               4     0       /*This file contains the following routines to update the database.*/
dbup               5     0       /*      ra - add a record occurrence to the database - recadd
dbup               6     0              ao - add an owner set occurrence  - ownadd
dbup               7     0              am - add a member set occurrence - mebadd
dbup               8     0              dr - delete a record occurrence from the database - recdel
dbup               9     0              do - delete an owner set occurrence - owndel
dbup              10     0              dm - delete a member set occurrence -.mebdel
dbup              11     0              fr - find a record - recfind
dbup              12     0              fo - find an owner record of a specific member in a set - ownfind
dbup              13     0              ff - find the first member of an owrer in a set - mebfind
dbup              14     0              fn - find next owner or member in a set - nextfind
dbup              15     0              co - change owner of specific member to new owner in same set - cngown
dbup              16     0              ca - change all members of an owner to new owner in same set - cngall
dbup              17     0
dbup              18     0
dbup              19     0          ------------------------------------------------------------------
dbup              20     0
dbup              21     0                              Service routines called by the update routines
dbup              22     0
dbup              23     0              setoadd
dbup              24     0              owntdel
dbup              25     0              mebtdel
dbup              26     0              rscor
dbup              27     0              savecor
dbup              28     0              putpts
dbup              29     0              getpts
dbup              30     0              valkr
dbup              31     0              valks
dbup              32     0              bldkey
dbup              33     0              nfields
dbup              34     0
dbup              35     0
dbup              36     0       NOTE: dbup_c_dec includes declarations that are common to the update routines
dbup              37     0              and includes declarations of all external variables.*/
dbup              38     0
dbup              39     0
```

```
file            line level  source

dbup              41     0       /*Recadd will add records to a record file type as designated by the cor
dbup              42     0         record type.  If a file is specified, all the records in the file
dbup              43     0         are added.  Otherwise, records are added from the standard input until
dbup              44     0         EOF is read.  The cor unit is set to the key of the last record added.*/
dbup              45     0
dbup              46     0
dbup              47     0       recadd (ifp)
dbup              48     0       FILE *ifp;
dbup              49     0       {
dbup              50     1       #include "dbup_c_dec"
dbup              51     1               char c;
dbup              52     1               int more;
dbup              53     1               char *i;
dbup              54     1               struct set_type *pst;
dbup              55     1       int n;
dbup              56     1
dbup              57     1       #ifdef DEBUG
dbup              58     1       printf ("entering record add\n");
dbup              59     1       #endif
dbup              60     1
dbup              61     1               more = TRUE;
dbup              62     1               while (more)
dbup              63     1               {
dbup              64     2                       if ((c = getc(ifp)) == EOF)
dbup              65     2                               more = FALSE;
dbup              66     2                       else
dbup              67     2                       {
dbup              68     3                               ungetc(c,ifp);
dbup              69     3                               i = iobuf;
dbup              70     3                               while ((c = getc(ifp)) != END_OF_RECORD &&
dbup              71     3                                       i < iobuf + MAX_RECORD)
dbup              72     3                                       *i++ = c;
dbup              73     3                               if (i == iobuf + MAX_RECORD)
dbup              74     3                                       return(error("input record too long"));
dbup              75     3                               *i++ = END_OF_RECORD;
dbup              76     3                               *i = NULL;
dbup              77     3                                               /*validate input record*/
dbup              78     3                               if((n=nfields(iobuf,cor_rt->fdlim)) != cor_rt->nosfield)
dbup              79     3                                       return(error("illegal # of fields for record type"));
dbup              80     3                               else if (bldkey(iobuf,key) == ERROR)
dbup              81     3                                       return(ERROR);
dbup              82     3                               else if (item_search(rbtfd,rbtroot,key,&os) == PRESENT)
dbup              83     3                                       fprintf(stderr,"%s duplicate key\n", key);
dbup              84     3                               else
dbup              85     3                               {               /*write rec to file and save offsets*/
dbup              86     4
dbup              87     4       #ifdef DEBUG
dbup              88     4       printf("key= %s\n",key);
dbup              89     4       #endif
dbup              90     4                                       os.btd = NEWL;
```

```
file            line level  source

dbup              91     4                                os.btb = NEW;
dbup              92     4                                if (write_rec(&os.btd,&os.btb,iobuf,&cor_rt->rc1)
dbup              93     4                                     == ERROR)
dbup              94     4                                    return(ERROR);
dbup              95     4                                else if (item_insert(rbtfd,&rbtroot,key,&os) == NULL)
dbup              96     4                                    return(ERROR);
dbup              97     4                                else
dbup              98     4                                {
dbup              99     5                                    strcpy(cor_unit, key);
dbup             100     5                                    pst = st_root;
dbup             101     5                                    setoadd (pst);          /*add key as owner in sets*/
dbup             102     5                                }
dbup             103     4                            }
dbup             104     3                        }
dbup             105     2                    }
dbup             106     1                    return(OK);
dbup             107     1            }
dbup             108     0
dbup             109     0
dbup             110     0
dbup             111     0
dbup             112     0
```

```
file        line level  source

dbup         114     0      /*Ownadd adds an owner set occurrence to a set type*/
dbup         115     0
dbup         116     0
dbup         117     0      ownadd()
dbup         118     0      {
dbup         119     1      #include "dbup_c_dec"
dbup         120     1
dbup         121     1
dbup         122     1      #ifdef DEBUG
dbup         123     1      printf("e ownadd\n");
dbup         124     1      #endif
dbup         125     1                                  /*set cor record type*/
dbup         126     1              cor_rt = cor_st->owner;
dbup         127     1              if (valkr(&os, iobuf) != ERROR)         /*validate key and record format*/
dbup         128     1              {
dbup         129     2                      os.btd = NEWL;          /*build new owner record occur*/
dbup         130     2                      os.btb = NEW;
dbup         131     2                      *fp = NULL;
dbup         132     2                      *bp = NULL;
dbup         133     2                                  /*store the new owner record and offset*/
dbup         134     2                      if (putpts(fp,bp,&os) == ERROR ||
dbup         135     2                          item_insert(cor_st->sbtfd,&cor_st->sbtroot,cor_unit,&os)
dbup         136     2                              != PRESENT)
dbup         137     2                              return(ERROR);
dbup         138     2                      else
dbup         139     2                              return(OK);
dbup         140     2              }
dbup         141     1              else
dbup         142     1                      return(ERROR);          /*invalid key or record format*/
dbup         143     1      }
dbup         144     0
dbup         145     0
dbup         146     0
dbup         147     0
dbup         148     0
```

```
dbup        150   0     /*Mebadd adds amember set occurrence to a set type*/
dbup        151   0
dbup        152   0
dbup        153   0     mebadd(mkey)
dbup        154   0     char *mkey;
dbup        155   0     {
dbup        156   1     #include "dbup_c_dec"
dbup        157   1
dbup        158   1     #ifdef DEBUG
dbup        159   1     printf("e mebadd\n");
dbup        160   1     #endif
dbup        161   1
dbup        162   1             cor_rt = cor_st->member;
dbup        163   1             strcpy (savekey, cor_unit);     /*save owner key*/
dbup        164   1             strcpy (cor_unit, mkey);        /*set currency  to member key*/
dbup        165   1             if (valkr(&os,iobuf) == ERROR)
dbup        166   1             {
dbup        167   2                     strcpy(cor_unit,savekey);       /*invalid key or record format*/
dbup        168   2                     return(OK);
dbup        169   2             }
dbup        170   1             else if (item_search(cor_st->sbtfd,cor_st->sbtroot,savekey,&os)
dbup        171   1                     != PRESENT)             /*make sure owner is in set type*/
dbup        172   1             {
dbup        173   2                     strcpy(cor_unit,savekey);
dbup        174   2                     return(kerror("owner not a member of settype", cor_unit));
dbup        175   2             }
dbup        176   1             else if (getpts(fp,bp,&os) == ERROR)    /*get owner pointers*/
dbup        177   1                     return(ERROR);
dbup        178   1             else if (*bp != NULL)
dbup        179   1                     return(error("owner key is member in the set"));
dbup        180   1             else
dbup        181   1             {               /*set owner forward pointer to new member*/
dbup        182   2                     strcpy(savenext,fp);
dbup        183   2                     strcpy(fp,cor_unit);
dbup        184   2                     putpts(fp,bp,&os);
dbup        185   2                     if (*savenext == NULL)  /*build a new member record*/
dbup        186   2                             strcpy(fp,savekey);
dbup        187   2                     else
dbup        188   2                             strcpy(fp,savenext);
dbup        189   2                     strcpy (bp, savekey);
dbup        190   2                     os.btd = NEWL;
dbup        191   2                     os.btb = NEW;
dbup        192   2                     putpts(fp,bp,&os);      /*store new member occr rec*/
dbup        193   2                     item_insert(cor_st->sbtfd,&cor_st->sbtroot,cor_unit,&os);
dbup        194   2                     if (*savenext != NULL)  /*fix backward pointer of next member*/
dbup        195   2                     {
dbup        196   3                             if (item_search(cor_st->sbtfd,cor_st->sbtroot,savenext,
dbup        197   3                             &os) != PRESENT ||
dbup        198   3                             getpts(fp,bp,&os) == ERROR ||
dbup        199   3                             strcmp(bp,savekey) != MATCH)
```

```
file            line level  source

dbup             200     3                                  return(kerror("set processing error",savenext));
dbup             201     3                              strcpy (bp,cor_unit);
dbup             202     3                              putpts(fp,bp,&os);
dbup             203     3                              return(OK);
dbup             204     3                          }
dbup             205     2                      }
dbup             206     1                  }
dbup             207     0
dbup             208     0
dbup             209     0
dbup             210     0
dbup             211     0
```

```
file          line level  source

dbup           213     0       /*Recdel deletes a record from the database record file. Any set occurrences
dbup           214     0         for this key are also deleted.*/
dbup           215     0
dbup           216     0
dbup           217     0       recdel()
dbup           218     0       {
dbup           219     1       #include "dbup_c_dec"
dbup           220     1               struct set_type *pst;
dbup           221     1       #ifdef DEBUG
dbup           222     1       printf("entering recdel\n");
dbup           223     1       #endif
dbup           224     1
dbup           225     1                                       /*delete the record occurrence*/
dbup           226     1               if (item_delete(rbtfd,&rbtroot,cor_unit,&os) != PRESENT)
dbup           227     1                       return(NOTPRESENT);
dbup           228     1               pst = st_root;          /*now delete all set occur*/
dbup           229     1               owntdel(pst);           /*delete owner occur*/
dbup           230     1               pst = st_root;
dbup           231     1               mebtdel(pst);           /*delete member occur*/
dbup           232     1               return(PRESENT);
dbup           233     1       }
dbup           234     0
dbup           235     0
dbup           236     0
dbup           237     0
dbup           238     0
```

```
file            line level  source

dbup             240     0      /*Owndel deletes an owner occurrence in a set. This causes the record in the
dbup             241     0        record file to be deleted and all members to be deleted. Also, any set that
dbup             242     0        contains this owner as a member is also visited and the owner-now-member
dbup             243     0        is deleted.*/
dbup             244     0
dbup             245     0
dbup             246     0      owndel()
dbup             247     0      {
dbup             248     1      #include "dbup_c_dec"
dbup             249     1              int more;
dbup             250     1      #ifdef DEBUG
dbup             251     1      printf("e owndel\n");
dbup             252     1      #endif
dbup             253     1
dbup             254     1              cor_rt = cor_st->owner;
dbup             255     1              if (item_delete(cor_st->sbtfd,&cor_st->sbtroot,cor_unit,&os) != PRESENT)
dbup             256     1                      return(NOTPRESENT);
dbup             257     1              else
dbup             258     1              {
dbup             259     2
dbup             260     2      #ifdef DEBUG
dbup             261     2      printf("after delete fo= %ld, bo= %d\n",os.btd,os.btb);
dbup             262     2      #endif
dbup             263     2
dbup             264     2                      savecor(unit,&st,&rt);          /*save currency pointers*/
dbup             265     2
dbup             266     2      #ifdef DEBUG
dbup             267     2      printf("after savecor- unit=%s, st= %d, rt= %d\n",unit,st,rt);
dbup             268     2      printf("should be -coru= %s, st= %d, rt= %d\n",cor_unit,cor_st,cor_rt);
dbup             269     2      #endif
dbup             270     2
dbup             271     2                      more = TRUE;
dbup             272     2                      while (more)
dbup             273     2                      {
dbup             274     3                              getpts(fp,bp,&os);      /*get key for next member*/
dbup             275     3      #ifdef DEBUG
dbup             276     3      printf("getpts- fp= %s, bp= %s\n",fp,bp);
dbup             277     3      #endif
dbup             278     3
dbup             279     3                              if (*fp == NULL)
dbup             280     3                                      more = FALSE;
dbup             281     3                              else
dbup             282     3                              {
dbup             283     4                                      strcpy(cor_unit,fp);    /*set currency for next member*/
dbup             284     4                                      mebdel(os.btd,os.btb);
dbup             285     4                              }
dbup             286     3                      }
dbup             287     2              }
dbup             288     1              rscor(unit, &st, &rt);          /*reset currency*/
```

```
file            line level  source

dbup             290     1      #ifdef DEBUG
dbup             291     1      printf("after rscor- coru= %s, corst= %d, corrt= %d\n",cor_unit,cor_st,cor_rt);
dbup             292     1      #endif
dbup             293     1
dbup             294     1              recdel();               /*delete the record from the record btree*/
dbup             295     1              rscor(unit,&st,&rt);
dbup             296     1              return(OK);
dbup             297     1      }
dbup             298     0
dbup             299     0
dbup             300     0
dbup             301     0
dbup             302     0
```

```
file            line level  source

dbup             304     0      /*mebdel deletes a member from a set occurrence.  If the prior member or owner
dbup             305     0        is not in the btree for the set, then the file offset and block offset passed
dbup             306     0        to the routine as parameters are checked. If the fos is equal to newl, the routine is
dbup             307     0        in the middle of a recursive call and this member will be deleted later. If
dbup             308     0        fos is not equal to newl, it then contains the file offset of the prior member
dbup             309     0        and is used to get the prior member's pointers.*/
dbup             310     0
dbup             311     0
dbup             312     0      mebdel(fos,bos)
dbup             313     0      long fos;
dbup             314     0      int bos;
dbup             315     0      {
dbup             316     1      #include "dbup_c_dec"
dbup             317     1
dbup             318     1
dbup             319     1      #ifdef DEBUG
dbup             320     1      printf ("e mebdel\n");
dbup             321     1      #endif
dbup             322     1
dbup             323     1              cor_rt = cor_st->member;
dbup             324     1              savecor(unit,&st,&rt);
dbup             325     1              if (item_delete(cor_st->sbtfd,&cor_st->sbtroot,cor_unit,&os) != PRESENT)
dbup             326     1                      return(NOTPRESENT);
dbup             327     1              saveos = os;
dbup             328     1              getpts (fp,bp,&os);
dbup             329     1              if (*bp == NULL || *fp == NULL)
dbup             330     1                      return(kerror("delete processing error", cor_unit));
dbup             331     1                                      /*get pointers for this member*/
dbup             332     1              if(item_search(cor_st->sbtfd,cor_st->sbtroot,bp,&os) != PRESENT)
dbup             333     1                      if (fos == NEWL)
dbup             334     1                              return(NOTPRESENT);  /*will get this member later*/
dbup             335     1                      else
dbup             336     1                      {
dbup             337     2                              os.btd = fos;
dbup             338     2                              os.btb = bos;
dbup             339     2                      }
dbup             340     1
dbup             341     1      #ifdef DEBUG
dbup             342     1      printf("back from i_search fo= %ld, bo= %d\n", os.btd,os.btb);
dbup             343     1      #endif
dbup             344     1
dbup             345     1              strcpy (savekey,bp);
dbup             346     1              strcpy (savenext,fp);
dbup             347     1
dbup             348     1      #ifdef DEBUG
dbup             349     1      printf("savekey= %s, savenext= %s\n",savekey,savenext);
dbup             350     1      #endif
dbup             351     1
dbup             352     1              getpts (fp,bp,&os);             /*get prior key's record*/
dbup             353     1              if (strcmp(savekey,savenext) == MATCH)
```

```
file          line level  source

dbup           354     1              {
dbup           355     2                                  /*last member for this owner*/
dbup           356     2                      if (*bp != NULL)
dbup           357     2                              return(kerror("delete processing error",bp));
dbup           358     2                      else
dbup           359     3                      {
dbup           360     3                              *fp = NULL;
dbup           361     3                              putpts(fp, bp, &os);    /*reset prior's pointers*/
dbup           362     2                      }
dbup           363     1              }
dbup           364     1              else
dbup           365     2              {
dbup           366     2                      strcpy(fp,savenext);    /*reset prior's pointers*/
dbup           367     2                      putpts(fp,bp,&os);
dbup           368     2                      if (item_search(cor_st->sbtfi,cor_st->sbtroot,savenext,&os)
dbup           369     2                          != PRESENT)
dbup           370     2                              return(kerror("missing set links", savenext));
dbup           371     2                      getpts(fp,bp,&os);      /*reset next member's pointers*/
dbup           372     2                      if (*bp != NULL)
dbup           373     2                              strcpy (bp,savekey);
dbup           374     2                      putpts(fp,bp,&os);
dbup           375     1              }
dbup           376     1              rscor (unit, &st, &rt);         /*now delete the record if present*/
dbup           377     1              recdel();
dbup           378     1              return(OK);
dbup           379     0      }
dbup           380     0
dbup           381     0
dbup           382     0
dbup           383     0
```

```
file            line level  source

dbup             385     0      /*Recfind outputs a record using the currency pointers to decide which key*/
dbup             386     0
dbup             387     0
dbup             388     0      recfind (ofp)
dbup             389     0      FILE *ofp;
dbup             390     0      {
dbup             391     1      #include "dbup_c_dec"
dbup             392     1              char c, *cptr;
dbup             393     1
dbup             394     1                                      /*get the record from db storage*/
dbup             395     1      #ifdef DEBUG
dbup             396     1      printf("entering recfind\n");
dbup             397     1      #endif
dbup             398     1              if (item_search(rbtfd,rbtroot,cor_unit,&os) != PRESENT)
dbup             399     1                      return(kerror(rbtfd,rbtroot,cor_unit,&os));
dbup             400     1                                      /*get record into io buffer*/
dbup             401     1              if (read_rec(os.btd, os.btb, iobuf, &cor_rt->rci) == ERROR)
dbup             402     1                      return (ERROR);
dbup             403     1              c = 'p';
dbup             404     1              cptr = iobuf;
dbup             405     1              while (c != END_OF_RECORD)
dbup             406     1              {
dbup             407     2                      c = *cptr++;
dbup             408     2                      putc (c,ofp);
dbup             409     2              }
dbup             410     1              return(OK);
dbup             411     1      }
dbup             412     0
dbup             413     0
dbup             414     0
dbup             415     0
dbup             416     0
```

```
file            line level  source

dbup             418     0      /*Ownfind finds the owner for a member key and outputs the record for that
dbup             419     0        owner.*/
dbup             420     0
dbup             421     0
dbup             422     0      ownfind (ofp)
dbup             423     0      FILE *ofp;
dbup             424     0      {
dbup             425     1      #include "dbup_c_dec"
dbup             426     1
dbup             427     1      #ifdef DEBUG
dbup             428     1      printf("e ownfind\n");
dbup             429     1      #endif
dbup             430     1
dbup             431     1
dbup             432     1
dbup             433     1              cor_rt = cor_st->owner;          /*find set owner occur*/
dbup             434     1              if (item_search(cor_st->sbtfd,cor_st->sbtroot,cor_unit,&os) != PRESENT)
dbup             435     1                      return(kerror("key absent from set type",cor_unit));
dbup             436     1              getpts(fp,bp,&os);
dbup             437     1              if (*bp == NULL)
dbup             438     1                      return(kerror("key is an owner",cor_unit));
dbup             439     1              while (*bp != NULL)
dbup             440     1              {                                /*follow links to owner*/
dbup             441     2                      if(item_search(cor_st->sbtfd,cor_st->sbtroot,fp,&os) != PRESENT)
dbup             442     2                              return(kerror("processing error-missing set link",cor_unit));
dbup             443     2                      strcpy(cor_unit,fp);
dbup             444     2                      getpts(fp,bp,&os);
dbup             445     2              }
dbup             446     1              recfind(ofp);           /*output owner record*/
dbup             447     1              return(OK);
dbup             448     1      }
dbup             449     0
dbup             450     0
dbup             451     0
dbup             452     0
dbup             453     0
```

```
file            line level  source

dbup             455     0      /*Mebfind finds the 1st member record for an owner*/
dbup             456     0
dbup             457     0
dbup             458     0      mebfind (ofp)
dbup             459     0      FILE *ofp;
dbup             460     0      {
dbup             461     1      #include "dbup_c_dec"
dbup             462     1
dbup             463     1      #ifdef DEBUG
dbup             464     1      printf("e mebfind \n");
dbup             465     1      #endif
dbup             466     1
dbup             467     1
dbup             468     1
dbup             469     1              cor_rt = cor_st->member;
dbup             470     1              if (item_search(cor_st->sbtfd,cor_st->sbtroot,cor_unit,&os) != PRESENT)
dbup             471     1                      return(kerror("key absent from set type",cor_unit));
dbup             472     1              getpts(fp,bp,&os);
dbup             473     1              if (*bp != NULL)
dbup             474     1                      return(kerror("key not an owner",cor_unit));
dbup             475     1              if (*fp == NULL)
dbup             476     1                      return(kerror("key has no members",cor_unit));
dbup             477     1              strcpy (cor_unit,fp);           /*reset currency to member's record*/
dbup             478     1              strcpy (cor_st->cos,cor_unit);
dbup             479     1              recfind(ofp);                   /*output record*/
dbup             480     1              return(OK);
dbup             481     1      }
dbup             482     0
dbup             483     0
dbup             484     0
dbup             485     0
dbup             486     0
```

```
file            line level  source

dbup             488     0      /*Nextfind outputs the record of the next key in the set link*/
dbup             489     0
dbup             490     0      nextfind (ofp)
dbup             491     0      FILE *ofp;
dbup             492     0      {
dbup             493     1      #include "dbup_c_dec"
dbup             494     1
dbup             495     1      #ifdef DEBUG
dbup             496     1      printf("e nextfind\n");
dbup             497     1      #endif
dbup             498     1
dbup             499     1
dbup             500     1              if ((*cor_st->cos) == NULL)
dbup             501     1                      return(error("undefined current of set type"));
dbup             502     1              strcpy(cor_unit,cor_st->cos);
dbup             503     1              if(item_search(cor_st->sbtfd,cor_st->sbtroot,cor_unit,&os) != PRESENT)
dbup             504     1                      return(kerror("key absent from set type",cor_unit));
dbup             505     1              getpts(fp,bp,&os);              /*find next key in set*/
dbup             506     1              if (*fp != NULL && *bp != NULL)
dbup             507     1              {
dbup             508     2                      strcpy(cor_unit,fp);
dbup             509     2                      if (item_search(cor_st->sbtfd,cor_st->sbtroot,cor_unit,&os) != PRESENT)
dbup             510     2                              return(kerror("missing set link",cor_unit));
dbup             511     2                      getpts(fp,bp,&os);      /*set cor record type*/
dbup             512     2                      if (*bp == NULL)
dbup             513     2                              return (error ("last member reached"));
dbup             514     2                      else
dbup             515     2                      {
dbup             516     3                              cor_rt = cor_st->member;
dbup             517     3                              strcpy(cor_st->cos, cor_unit);  /*reset cur set type*/
dbup             518     3                              recfind (ofp);                  /*output record*/
dbup             519     3                              return (OK);
dbup             520     3                      }
dbup             521     2              }
dbup             522     1              else
dbup             523     1                      return(error("owner key is current of set type"));
dbup             524     1      }
dbup             525     0
dbup             526     0
dbup             527     0
dbup             528     0
dbup             529     0
```

```
file            line level  source

dbup              531     0       /*Cngown changes a member in a set occurrence to a new owner in the same set*/
dbup              532     0
dbup              533     0
dbup              534     0       cngown(newokey)
dbup              535     0       char *newokey;
dbup              536     0       {
dbup              537     1       #include "dbup_c_dec"
dbup              538     1
dbup              539     1       #ifdef DEBUG
dbup              540     1       printf("e cngown\n");
dbup              541     1       #endif
dbup              542     1
dbup              543     1
dbup              544     1
dbup              545     1                                       /*validate new owner key and member key*/
dbup              546     1               if (valks(newokey,&os,&saveos) == ERROR)
dbup              547     1                       return(ERROR);
dbup              548     1               getpts(fp,bp,&os);              /*get member's links*/
dbup              549     1               if (*bp == NULL)
dbup              550     1                       return(kerror("key is owner in this set",cor_unit));
dbup              551     1               if (item_search(cor_st->sbtfd,cor_st->sbtroot,bp,&saveos2) != PRESENT)
dbup              552     1                       return(kerror("missing set links",fp));
dbup              553     1               if (strcmp(fp,bp) == MATCH)
dbup              554     1               {                               /*final member in this occur.*/
dbup              555     2                       getpts(fp,bp,&saveos2);
dbup              556     2                       *fp = NULL;                     /*reset owner's pointers*/
dbup              557     2                       putpts(fp,bp,&saveos2);
dbup              558     2               }
dbup              559     1               else
dbup              560     1               {
dbup              561     2                       strcpy(savenext,fp);            /*save member's links*/
dbup              562     2                       strcpy(savekey,bp);
dbup              563     2                       getpts(fp,bp,&saveos2);         /*reset prior member's links*/
dbup              564     2                       strcpy(fp,savenext);
dbup              565     2                       putpts(fp,bp,&saveos2);
dbup              566     2                       if(item_search(cor_st->sbtfd,cor_st->sbtroot,savenext,&saveos2) != PRESENT)
dbup              567     2                               return(kerror("missing set link",savenext));
dbup              568     2                       getpts(fp,bp,&saveos2);         /*reset following mem's links*/
dbup              569     2                       if (*bp != NULL)
dbup              570     2                               strcpy(bp,savekey);
dbup              571     2                       putpts(fp,bp,&saveos2);
dbup              572     2               }
dbup              573     1                                       /*now add member to new owner*/
dbup              574     1               getpts(fp,bp,&saveos);          /*reset new owner's links*/
dbup              575     1               strcpy(savenext,fp);
dbup              576     1               strcpy(fp,cor_unit);
dbup              577     1               putpts(fp,bp,&saveos);
dbup              578     1                                               /*reset next member's links*/
dbup              579     1               if (*savenext != NULL)
dbup              580     1               {
```

```
file            line level  source

dbup             581     2                      if(item_search(cor_st->sbtfd,cor_st->sbtroot,savenext,&saveos) != PRESENT)
dbup             582     2                              return(kerror("missing links",savenext));
dbup             583     2                      getpts(fp,bp,&saveos);
dbup             584     2                      strcpy(savekey,bp);
dbup             585     2                      strcpy(bp,cor_unit);
dbup             586     2                      putpts(fp,bp,&saveos);
dbup             587     2              }
dbup             588     1              else
dbup             589     1                      strcpy(savenext,newokey);       /*new owner has no members*/
dbup             590     1              strcpy(fp,savenext);            /*set member's new links*/
dbup             591     1              strcpy(bp,newokey);
dbup             592     1              putpts(fp,bp,&os);
dbup             593     1              return(OK);
dbup             594     1      }
dbup             595     0
dbup             596     0
dbup             597     0
dbup             598     0
dbup             599     0
```

```
file          line level  source

dbup           601     0       /*Cngall changes all the members in a set occurrence to a new owner within
dbup           602     0         the same set.*/
dbup           603     0
dbup           604     0
dbup           605     0       cngall(newokey)
dbup           606     0       char *newokey;
dbup           607     0       {
dbup           608     1       #include "dbup_c_dec"
dbup           609     1
dbup           610     1       #ifdef DEBUG
dbup           611     1       printf("e cngall\n");
dbup           612     1       #endif
dbup           613     1
dbup           614     1
dbup           615     1
dbup           616     1               if (valks(newokey,&os,&saveos) == ERROR)
dbup           617     1                       return(ERROR);
dbup           618     1               getpts(fp,bp,&os);              /*get member key*/
dbup           619     1               while (*fp != NULL)
dbup           620     1               {
dbup           621     2                       strcpy(cor_unit,fp);    /*move each member to newowner*/
dbup           622     2                       if (cngown(newokey) == ERROR)
dbup           623     2                               return(ERROR);
dbup           624     2                       getpts(fp,bp,&os);
dbup           625     2               }
dbup           626     1               strcpy(cor_unit,newokey);
dbup           627     1       }
dbup           628     0
dbup           629     0
dbup           630     0
dbup           631     0
dbup           632     0
```

```
file            line level  source

dbup             634     0      /*-------------------------------------------------------------------------*/
dbup             635     0
dbup             636     0              /*              Service routines                */
dbup             637     0
dbup             638     0
dbup             639     0
dbup             640     0
dbup             641     0
dbup             642     0      /*Setoadd searches the set type binary tree and if a match of owner and
dbup             643     0        cor record type is found, adds the new record key as an owner to the set*/
dbup             644     0
dbup             645     0
dbup             646     0      setoadd(pst)
dbup             647     0      struct set_type *pst;
dbup             648     0      {
dbup             649     1      #include "ext_dec"
dbup             650     1      #ifdef DEBUG
dbup             651     1      printf("e setoadd\n");
dbup             652     1      #endif
dbup             653     1              if(pst == NULL)
dbup             654     1                      return;
dbup             655     1              else
dbup             656     1              {
dbup             657     2                      if (pst->owner == cor_rt)
dbup             658     2                      {
dbup             659     3                              cor_st = pst;
dbup             660     3                              ownadd();
dbup             661     3                      }
dbup             662     2                      setoadd(pst->sleft);
dbup             663     2                      setoadd(pst->sright);
dbup             664     2              }
dbup             665     1      }
dbup             666     0
dbup             667     0
```

```
file            line level  source

dbup             669     0      /*Owntdel searches the set type binary tree and if a match of owner and
dbup             670     0        cor record type is found, deletes any owner record with cor unit as key.*/
dbup             671     0
dbup             672     0      owntdel(pst)
dbup             673     0      struct set_type *pst;
dbup             674     0      {
dbup             675     1      #include "ext_dec"
dbup             676     1      #ifdef DEBUG
dbup             677     1      printf("e owntdel\n");
dbup             678     1      #endif
dbup             679     1              if(pst == NULL)
dbup             680     1                      return;
dbup             681     1              else
dbup             682     1              {
dbup             683     2                      if (pst->owner == cor_rt)
dbup             684     2                      {
dbup             685     3                              cor_st = pst;
dbup             686     3                              owndel();
dbup             687     3                      }
dbup             688     2                      owntdel(pst->sleft);
dbup             689     2                      owntdel(pst->sright);
dbup             690     2              }
dbup             691     1      }
dbup             692     0
dbup             693     0
dbup             694     0
dbup             695     0
dbup             696     0
```

```
file            line level  source

dbup             698     0      /*Mebtdel searches the set type tree and if a match of member and cor_rt
dbup             699     0        is found, deletes the owner record of cor_unit.*/
dbup             700     0
dbup             701     0
dbup             702     0      mebtdel(pst)
dbup             703     0      struct set_type *pst;
dbup             704     0      {
dbup             705     1      #include "ext_dec"
dbup             706     1      #ifdef DEBUG
dbup             707     1      printf("e mebtdel\n");
dbup             708     1      #endif
dbup             709     1
dbup             710     1
dbup             711     1              if (pst == NULL)
dbup             712     1                      return;
dbup             713     1              else
dbup             714     1              {
dbup             715     2                      if (pst->member == cor_rt)
dbup             716     2                      {
dbup             717     3                              cor_st = pst;
dbup             718     3                              mebdel(NEWL,NEW);
dbup             719     3                      }
dbup             720     2                      mebtdel(pst->sleft);
dbup             721     2                      mebtdel(pst->sright);
dbup             722     2              }
dbup             723     1              return;
dbup             724     1      }
dbup             725     0
dbup             726     0
dbup             727     0
dbup             728     0
dbup             729     0
```

```
file          line level  source

dbup           731     0      /*Rscor resets the currency pointers*/
dbup           732     0
dbup           733     0
dbup           734     0      rscor(unit,st,rt)
dbup           735     0      char *unit;
dbup           736     0      struct rec_type **rt;
dbup           737     0      struct set_type **st;
dbup           738     0      {
dbup           739     1      #include "ext_dec"
dbup           740     1
dbup           741     1      #ifdef DEBUG
dbup           742     1      printf("e rscor\n");
dbup           743     1      #endif
dbup           744     1              strcpy(cor_unit,unit);
dbup           745     1              cor_st = *st;
dbup           746     1              cor_rt = *rt;
dbup           747     1              return;
dbup           748     1      }
dbup           749     0
dbup           750     0
dbup           751     0
dbup           752     0
dbup           753     0
```

```
file            line level  source

dbup             755     0      /*Savecor saves the currency pointers*/
dbup             756     0
dbup             757     0
dbup             758     0      savecor(unit,st,rt)
dbup             759     0      char *unit;
dbup             760     0      struct set_type **st;
dbup             761     0      struct rec_type **rt;
dbup             762     0      {
dbup             763     1      #include "ext_dec"
dbup             764     1
dbup             765     1      #ifdef DEBUG
dbup             766     1      printf("e savecor\n");
dbup             767     1      #endif
dbup             768     1
dbup             769     1              strcpy(unit,cor_unit);
dbup             770     1              *st = cor_st;
dbup             771     1              *rt = cor_rt;
dbup             772     1              return;
dbup             773     1      }
dbup             774     0
dbup             775     0
dbup             776     0
dbup             777     0
dbup             778     0
```

```
file            line level  source

dbup             780     0      /*Putpts builds a set type record containing the links for an owner or member in
dbup             781     0        a set occurrence. Once the record is built, it is output to the set file.*/
dbup             782     0
dbup             783     0
dbup             784     0      putpts(fp,bp,os)
dbup             785     0      char *fp,*bp;
dbup             786     0      struct ptrs *os;
dbup             787     0      {
dbup             788     1      #include "ext_dec"
dbup             789     1              char iobuf[MAX_RECORD+1];
dbup             790     1              char *pt;
dbup             791     1              char c;
dbup             792     1      #ifdef DEBUG
dbup             793     1      printf("e putpts\n");
dbup             794     1      #endif
dbup             795     1
dbup             796     1
dbup             797     1                                      /*build pointer record*/
dbup             798     1              pt = iobuf;
dbup             799     1              c = 'a';
dbup             800     1              while (c != NULL)
dbup             801     1                      c = *pt++ = *fp++;
dbup             802     1              while(pt <iobuf + MAX_KEY +1)
dbup             803     1                      *pt++ = '*';
dbup             804     1              c = 'a';
dbup             805     1              while (c != NULL)
dbup             806     1                      c = *pt++ = *bp++;
dbup             807     1              while(pt < iobuf + 2*(MAX_KEY + 1))
dbup             808     1                      *pt++ = '*';
dbup             809     1              *pt = END_OF_RECORD;
dbup             810     1              if (write_rec(&os->btd,&os->btb,iobuf,&cor_st->sci) == OK)
dbup             811     1                      return(OK);
dbup             812     1              else
dbup             813     1                      return(ERROR);
dbup             814     1      }
dbup             815     0
dbup             816     0
dbup             817     0
dbup             818     0
dbup             819     0
```

```
file            line level  source

dbup             821     0      /*Getpts retrieves the forward and rear pointers after reading a set type
dbup             822     0        record*/
dbup             823     0
dbup             824     0
dbup             825     0      getpts (fp,bp,os)
dbup             826     0      char *fp,*bp;
dbup             827     0      struct ptrs *os;
dbup             828     0      {
dbup             829     1      #include "ext_dec"
dbup             830     1              char iobuf[MAX_RECORD + 1];
dbup             831     1              char *pt;
dbup             832     1
dbup             833     1      #ifdef DEBUG
dbup             834     1      printf("e getpts\n");
dbup             835     1      #endif
dbup             836     1
dbup             837     1
dbup             838     1              if(read_rec(os->btd,os->btb,iobuf,&cor_st->sc1) == ERROR)
dbup             839     1                      return(ERROR);
dbup             840     1              pt = iobuf;
dbup             841     1              while((*fp++ = *pt++) != NULL)  /*get forward pointer*/
dbup             842     1                      ;
dbup             843     1              pt = iobuf + MAX_KEY + 1;
dbup             844     1              while((*bp++ = *pt++) != NULL)  /*get backward pointer*/
dbup             845     1                      ;
dbup             846     1              return(OK);
dbup             847     1      }
dbup             848     0
dbup             849     0
dbup             850     0
dbup             851     0
dbup             852     0
```

```
file            line level  source

dbup             854     0      /*Valkr validates the key for an add of owner or member into set type and
dbup             855     0        checks record against cor record type format.*/
dbup             856     0
dbup             857     0
dbup             858     0      valkr (os,iobuf)
dbup             859     0      struct ptrs *os;
dbup             860     0      char *iobuf;
dbup             861     0      {
dbup             862     1      #include "ext_dec"
dbup             863     1              char key[MAX_KEY + 1];
dbup             864     1
dbup             865     1      #ifdef DEBUG
dbup             866     1      printf("e valkr\n");
dbup             867     1      #endif
dbup             868     1
dbup             869     1
dbup             870     1              if(item_search(cor_st->sbtfd,cor_st->sbtroot,cor_unit,os) == PRESENT)
dbup             871     1                      return(kerror("key already present in set type",cor_unit));
dbup             872     1              if(item_search(rbtfd,rbtroot,cor_unit,os) != PRESENT)
dbup             873     1                      return(kerror("record not present in database",cor_unit));
dbup             874     1              if (read_rec(os->btd,os->btb,iobuf,&cor_rt->rc1) == ERROR)
dbup             875     1                      return(ERROR);
dbup             876     1              if (bldkey (iobuf,key) == ERROR || strcmp(key,cor_unit) != MATCH)
dbup             877     1                      return(kerror("record doesn't fit set type",key));
dbup             878     1              else
dbup             879     1                      return(OK);
dbup             880     1      }
dbup             881     0
dbup             882     0
dbup             883     0
dbup             884     0
dbup             885     0
```

```
file            line level  source

dbup             887     0      /*Valks validates the keys input for change owner and change all commands*/
dbup             888     0
dbup             889     0
dbup             890     0      valks(newokey,os,saveos)
dbup             891     0      char *newokey;
dbup             892     0      struct ptrs *os,*saveos;
dbup             893     0      {
dbup             894     1      #include "ext_dec"
dbup             895     1
dbup             896     1      #ifdef DEBUG
dbup             897     1      printf("e valks\n");
dbup             898     1      #endif
dbup             899     1
dbup             900     1              if (item_search(cor_st->sbtfd,cor_st->sbtroot,newokey,saveos) != PRESENT)
dbup             901     1                      return(kerror("new owner key absent from set",newokey));
dbup             902     1              if (item_search(cor_st->sbtfd,cor_st->sbtroot,cor_unit,os) != PRESENT)
dbup             903     1                      return(kerror("key absent from set",cor_unit));
dbup             904     1              return(OK);
dbup             905     1      }
dbup             906     0
dbup             907     0
dbup             908     0
dbup             909     0
dbup             910     0
```

```
file            line level  source

dbup             912     0      /*Bldkey uses the cor record type as a template and builds a key from the record
dbup             913     0        in the i/o buffer.*/
dbup             914     0
dbup             915     0
dbup             916     0      bldkey (iobuf,key)
dbup             917     0      char *iobuf,*key;
dbup             918     0      {
dbup             919     1      #include "ext_dec"
dbup             920     1              char *pfb;              /*pointer to fields of buffer as we move down*/
dbup             921     1              char *prk;              /*pointer to fields of key as we move down*/
dbup             922     1              int i,num;
dbup             923     1              int *nextkf;
dbup             924     1              char *start;            /*start of each key field*/
dbup             925     1
dbup             926     1      #ifdef DEBUG
dbup             927     1      printf("e bldkey\n");
dbup             928     1      #endif
dbup             929     1
dbup             930     1
dbup             931     1              nextkf = cor_rt->keyf;
dbup             932     1              prk = key;
dbup             933     1              for (i=0; i<cor_rt->noskeyf; i++, nextkf++)
dbup             934     1              {
dbup             935     2                      pfb = iobuf;
dbup             936     2                      num = 0;
dbup             937     2                      while(num != *nextkf -1)
dbup             938     2                      {               /*find right field*/
dbup             939     3                              if (*pfb == NULL)
dbup             940     3                                      return(error("illegal key field"));
dbup             941     3                              else if(*pfb++ == cor_rt->fdlim)
dbup             942     3                                      num += 1;
dbup             943     3                      }
dbup             944     2                      start = pfb;
dbup             945     2                      while (*pfb != NULL && *pfb != END_OF_RECORD &&
dbup             946     2                             *pfb != cor_rt->fdlim && prk < key+MAX_KEY)
dbup             947     2                              *prk++ = *pfb++;
dbup             948     2                      if (start == pfb)
dbup             949     2                              return(error("null key fields are not allowed"));
dbup             950     2                      if (prk == key + MAX_KEY)
dbup             951     2                              return(error("key exceeds max length"));
dbup             952     2              }
dbup             953     1              *prk = NULL;            /*end key string*/
dbup             954     1              if (valchar(key) == ERROR)              /*validate characters in key*/
dbup             955     1                      return(ERROR);
dbup             956     1              return(OK);
dbup             957     1      }
dbup             958     0
dbup             959     0
dbup             960     0
dbup             961     0
```

file	line	level	source
dbup	962	0	

```
file        line level  source

dbup         964     0      valchar (key)
dbup         965     0      char *key;              /*validate characters in key are printable*/
dbup         966     0      {                       /*ascii dependent*/
dbup         967     1
dbup         968     1      #ifdef DEBUG
dbup         969     1      printf ("e valchar\n");
dbup         970     1      #endif
dbup         971     1
dbup         972     1              while (*key != NULL)
dbup         973     1                      if (*key < ' ' || *key > '~').
dbup         974     1                              return(error("invalid character in key"));
dbup         975     1                      else
dbup         976     1                              key++;
dbup         977     1              return (OK);
dbup         978     1      }
dbup         979     0
dbup         980     0
dbup         981     0
dbup         982     0
dbup         983     0
```

```
file            line level  source

dbup             985     0      /*Nfields returns the number of fields in a string*/
dbup             986     0
dbup             987     0
dbup             988     0      nfields (string,fdlm)
dbup             989     0      char *string;
dbup             990     0      char fdlm;                          /*filield delimiter*/
dbup             991     0      {
dbup             992     1              int num;                    /*number of fields*/
dbup             993     1
dbup             994     1      #ifdef DEBUG
dbup             995     1      printf("e nfields, fdlm= %c\n",fdlm);
dbup             996     1      #endif
dbup             997     1              num = 0;
dbup             998     1
dbup             999     1              for (;*string != NULL; string++)
dbup            1000     1                      if (*string == fdlm || *string == END_OF_RECORD)
dbup            1001     1                              num += 1;
dbup            1002     1              return (num);
dbup            1003     1      }
```