A701

# RIM Installers Manual

by Jim Fox

University Computing Services
University of Washington

Revision date: January 9, 1989

This document describes the Rim program and the files of a Rim database. It is intended to assist programmers who are installing Rim or making modifications to it. You are assumed to be familiar with fortran-77 and with the tools that are necessary to edit, compile, load, and install products on your system. You are also assumed to be familiar with the *Rim Users Manual.*

copyright ©1988, 1989
RIM Installers Manual
UCS UCS Document A701
Revision Date January 9, 1989

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided also that the sections entitled "Distribution" and "Rim General Public License" are included exactly as in the original, and provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

# Contents

# List of Tables

Chapter 1

# Introduction

This document describes the internals of Rim, the Academic Computing Service's relation database program. It includes descriptions of the program, data, and files. You are assumed to have a working knowledge of Rim—as described in the Rim Reference Manual, and a knowledge of fortran 77.

Rim is written entirely in fortran 77. Of its few hundred subroutines, only 20 are 'system dependent' and have to be edited or rewritten for each operating system. A list of these appears in table **??**. The others should compile without modification. All source is upper case and has a maximum record length of 72 columns.

Common blocks and system parameters are included in subroutines via 'include' statements, having the form

| | |
|---|---|
| **INCLUDE** '*FILENAME*.DCK' | on the VAX/VMS version, and |
| **INCLUDE** (*FILENAME*) | on the IBM/CMS version, and |
| **INCLUDE** '*filename*.d' | on the UNIX version. |

so there is an inherent system dependency. However, include statements always begin in column 7 and the included file never includes another file.

Rim has a long history and was worked on by many programmers over a period of many years—there is, alas, little uniformity in style among the subroutines. There are also no variable naming conventions—except that parameters nearly always begin with 'Z' and variables do not.

Rim has stopped using the terms 'relation', 'attribute', and 'tuple' and now uses only 'table', 'column', and 'row' for all user messages and documentation. However, most of Rim's subroutines still use the old terms. You should consider the sets of terms to be synonymous.

The term 'page' is used to mean a fixed-length portion of a file. All I/O is performed at the page level. The term 'record' means a logical set of data. Generally a page will contain several records.

Rim uses CHARACTER variables and constants for messages and for internal communication but most text, and all character data on file, is represented by 7-bit ASCII codes in integer variables. Thus, for example, when the integer variable CH contains 97 it represents a lower case ‘a’. A variable containing a single ascii character is call an ‘ascii-char’.

To reduce memory requirements ascii characters are often packed and stored in arrays. Words of 32 or 36 bits can hold 4 characters; words of 60 or 64 bits can hold 8 characters. An integer array containing packed characters is called ‘ascii-text’.

All dates are converted to Julian integers according to the algorithm of Tantzen.[1]

Times are converted to ‘Julian’ integers according to the algorithm

$$\text{integer} = \text{hours} * 3600 + \text{minutes} * 60 + \text{seconds}.$$

[1]Collected Algorithms of the ACM, 199, R. Tantzen.

User specified display formats occupy one word. The contents of a format integer depend on its corresponding data type.

The integer and real format integer is

$$\text{format} = (R * 10000 + D * 100 + N$$

where

| | |
|---|---|
| **R** | is the items per line, |
| **D** | is the decimal places, and |
| **N** | is the field width. |

The text format integer is

$$\text{format} = N$$

where

| | |
|---|---|
| **N** | is the field width. |

The date format integer is

$$\begin{aligned}\text{format} = \quad & (L * 12^5 + DP * 12^4 + MP * 12^3 + ML * 12^2 + \\ & YP * 12 + YL) * 128 + CH\end{aligned}$$

where

| | |
|---|---|
| **L** | is the total length of the displayed date, |
| **DP** | is the starting position of the day field, |
| **MP** | is the starting position of the month field, |
| **ML** | is length of the month field, |

**YP** is the starting position of the year field,

**YL** is length of the year field, and

**CH** is an ascii filler character.

ML can be 2 or 3. If $ML = 3$ then the month is printed as a three character abbreviation, otherwise it is a two digit number. YL can be 2 or 4. CH is a filler which is put wherever there is nothing else.

The time format integer is

$$\text{format} = (L * 12^5 + SP * 12^4 + MP * 12^3 + HP * 12) * 128 + CH$$

where

**L** is the total length of the displayed time,

**SP** is the starting position of the seconds field,

**MP** is the starting position of the minutes field,

**HP** is the starting position of the hour field,

**CH** is an ascii filler character.

SP can be zero, indicating that the seconds are not to be displayed. CH is a filler which is put wherever there is nothing else.

The SYSPAR include file defines parameters which tailor Rim to specific machines and installations. Most of the ‘Z____’ parameters are defined in this file. A few of the most common are:

**ZCW** Number of ascii characters that can be packed into a word.

**ZC** Maximum number of characters in a Rim name (table name, column name, password, link name, ...). This is 16 for most Rims.

**Z** Number of words required for a name of ascii-text. Normally $Z = \frac{ZC-1}{ZCW} + 1$. Variables that hold names are dimensioned (Z).

**ALL9S** The maximum allowable integer. Missing values and other flags are greater than **ALL9S**.

Chapter 2

# The Program

Rim contains a ‘trace’ flag which may be set with the Rim command

$$\textbf{set trace} \left\langle \begin{array}{c} \# \\ \textbf{on} \\ \textbf{off} \end{array} \right\rangle \langle \textbf{to } \textit{file} \rangle$$

where the number (#) is the trace level. **On** sets the value to 1. **Off** sets it to zero.

Many subroutines will print checkpoint information when the trace level reaches a specified point. This information and the trigger level are described in table 2.1. Other trace information may be easily added to this list by local installers.

| Subroutine | level | displayed information |
|---|---|---|
| NXTCRD | 1 | input text |
| SELWHR | 2 | key search att: pointer, key start |
| RIOIN | 3 | unit, page, #words, status |
| RIOOUT | 3 | unit, page, #words, status, position |
| SELWHR | 4 | key testing: att#, pointer, key start |
| RMLOOK | 4 | status, new position, nblk |
| RMRES | 5 | '1' indptr, indcur |
| RMRES | 5 | '2' indptr, buffer loc |
| RMSAV | 5 | indptr, buffer loc, #words |
| GETDAT | 6 | index, mat, offset, block, length, nid |
| GETDAT | 6 | 'del' offset, block |
| PGEXEC | 8 | opcode, prog source line# |
| BLKCHG | 9 | ind, rows, cols, blocks(-,ind) |
| BLKDEF | 9 | ind, rows, cols |
| BLKCLR | 11 | ind, blocks(-,ind) |

Table 2.1: Tracing information and trigger levels for subroutines that check the **trace** flag. The date and time are also displayed for all trace entries. You must check the source to see exactly which variables are displayed by each trace point.

Chapter 3

# Files

This chapter describes the three files which make up a RIM database.

- File 1 contains the schema—table, column, and link definitions, and other database information.
- File 2 contains the actual data tuples.
- File 3 contains B-trees for keyed attributes.

Each is a direct access file which is dynamically extensible.

File names are implementation dependent but generally take the form

*filename*.**rimdb***n*, ($N$ = 1,3).

| item # | fortran name | length (words) | field contents |
|---|---|---|---|
| 1 | KDBHDR | Z | a string containing "**RIM DATABASE**" |
| 2 | | 1 | version of Rim creating this database |
| 3 | DBNAME | Z | name of the database. often just the filename |
| 4 | OWNER | Z | owner password |
| 5 | DBDATE | 1 | date the database was last modified |
| 6 | DBTIME | 1 | time the database was last modified |
| 7 | LF1REC | 1 | last file 1 record used |
| 8 | NRROW | 1 | next available relation pointer[1] |
| 9 | NAROW | 1 | next available attribute pointer[1] |
| 10 | NLROW | 1 | next available link pointer[1] |
| | | — | The remainder of the first page is filled with zeros. |

[1] $pointer = \frac{\text{record\#}-1}{\text{words/row}} + \text{row offset}$

Table 3.1: File 1 header

This file contains the schema. It consists of a header, followed by one or more relation pages, attribute pages, and link pages.

The header is shown in table 3.1. It contains some identification information and pointers for the other pages. The rest of file 1 pages are relation pages, attribute pages, or link pages.

Relation data always begins on page 2 and continues on other pages via forward pointers. The set of relation pages constitutes the "Relation table". The Relation table obviously has 'holes'—corresponding to attribute and link pages. The relation table holds dictionary data for all the relations in the database. Each page of the relation table has the following structure:

| item # | fortran name | length (words) | field contents |
|---|---|---|---|
| 1 | NEXTROW | 1 | pointer[1] to the next row in this table. A negative value indicates that this row has been deleted. A value of zero indicates that the end of the table has been reached. |
| 2 | NAME | Z | relation name |
| 3 | RDATE | 1 | date of last modification of this relation. |
| 4 | NCOL | 1 | number of columns of fixed length in words. |
| 5 | NATT | 1 | number of attributes in this relation. |
| 6 | NTUPLE | 1 | number of rows (tuples) in this relation. |
| 7 | RSTART | 1 | data file pointer[2] for the start of data for this relation. |
| 8 | REND | 1 | data file pointer[2] for the end of data for this relation. |
| 9 | RPW | Z | read password for this relation. |
| 10 | MPW | Z | modify password for this relation. |

[1] relation pointer $= \frac{\text{record\#}-1}{\text{words/row}} + \text{row offset}$

[1] data pointer $= \text{record\#} * \text{ZHTOI} + \text{row offset}$

Table 3.2: File 1 relation record

**word (1)** pointer to the first row in the next page of the relation table. This value will be 0 if this page is the last relation table page.

**words(2)–(ZF1)** records of the relation table. Each record of ZRELL words describes one relation. Relation records are described by table 3.2.

Attribute data always begins on page 3 and also continues on other pages via forward pointers. The set of attribute pages constitutes the "Attribute table". The Attribute table obviously has 'holes'—corresponding to relation and link pages. The attribute table holds dictionary data for all the attributes in the database. Each page of the attribute table has the following structure:

**word (1)** pointer to the first row row in the next page of the attribute table. This value will be 0 if this page is the last attribute table page.

**words (2)–(ZF1)** records of the attribute table. Each record of ZATTL words describes one attribute of one relation. The attribute records are described in table 3.3.

Link data always begins on page 4 and also continues on other pages via forward pointers. The set of link pages constitutes the "Link table". The Link table obviously has 'holes'—corresponding to relation and attribute pages.

As an example of a database header, consider the UCS VMS and VM/CMS implementations. We have: ZCW = 4, ZC = 16, and therefore Z = 4. Also, the page length of file 1 is 1024 words (ZF1).

Each row of the relation table occupies 19 (ZRELL = 7+3*Z) words so with 1024 (ZF1) words per page you can fit 53 (ZRELR = (ZF1-1)/ZRELL) rows of relation data on each page. This means that rows 1 through 53 would be on page 1, 54 through 107 on page 2 and so on. To make sure the relation data starts on the second page, rows 1 through 53 are skipped and the first relation row is 54. Therefore the initial value for NRROW is 54.

Each row of the attribute table occupies 14 (ZATTL = 6+2*Z) words so with 1024 words per page we can fit 73 (ZATTR = (ZF1-1)/ZATTL) rows of attribute data on each page. This means that rows 1 through 73 would be on page 1, 74 through 146 on page 2 and so on. To make sure the attribute data starts on the third page, rows 1 through 146 are skipped the first attribute row is 147. Therefore the initial value for NAROW is 147.

Each row of the link table occupies 21 (ZLNKL = 1+5*Z) words so with 1024 words per page we can fit 48 (ZLNKR = (ZF1-1)/ZLNKL) rows of link data on each page. This means that rows 1 through 48 would be on page 1, 49 through 96 on page 2 and so on. To make sure the link data starts on the

| item # | fortran name | length (words) | field contents |
|---|---|---|---|
| 1 | NEXTROW | 1 | pointer[1] to the next row in this table. A negative value indicates that this row has been deleted. A value of zero indicates that the end of the table has been reached. |
| 2 | ATTNAM | Z | attribute name |
| 3 | RELNAM | Z | name of the relation that contains this attribute. |
| 4 | ATTCOL | 1 | starting column for this attribute in the data row. (a word index) |
| 5 | ATTLEN | 1 | length of this attribute. For variable length attributes this value is 0. For fixed length attributes this value is $A *$ ZHALF$+B$, where A and B depend . on the attribute type as follows: |
| 6 | ATTYPE | 1 | attribute type |
| 7 | ATTKEY | 1 | 0 for non-key attributes; B-tree starting record number for key attributes. |
| 8 | ATTFOR | 1 | attribute format |

| type | length | A | B |
|---|---|---|---|
| text | fixed | nchar | nwords |
| scaler | fixed | 0 | nwords[1] |
| vector | fixed | rows | nwords[2] |
| matrix | fixed | rows | nwords[3] |
| text | var | 0 | 0 |
| scaler | var | 0 | 0 |
| vector | var | 0 | 0 |
| matrix | fix-var | rows | 0 |
| matrix | var-var | 0 | 0 |

[1] attribute pointer $= \frac{\text{record}\#-1}{\text{words/row}} +$ row offset

Table 3.3: File 1 attribute record

| item # | fortran name | length (words) | field contents |
|---|---|---|---|
| 1 | KDBHDR | Z | a string containing **"RIM DATABASE"** |
| 2 | | 1 | version of Rim creating this database |
| 3 | DBNAME | Z | name of the database. often just the filename |
| 4 | OWNER | Z | owner password |
| 5 | DBDATE | 1 | date the database was last modified |
| 6 | DBTIME | 1 | time the database was last modified |
| 8 | LF2REC | 1 | current available page for adding data |
| 9 | LF2WRD | 1 | next available word for adding data |
| | | — | Actual data begins after the header on this page. |

Table 3.4: File 2 header

fourth page, rows 1 through 144 are skipped the first link row is 145. Therefore the initial value for NLROW is 145.

This file contains the actual data for all relations in the database. The header is shown in table 3.4. It contains some identification information and pointers for the other pages. All words after the header record on the first page and all following pages contain actual data. When RIM stores a row of data on file 2 it never breaks a row—each row has all its data values on single pages of file 2. This places a limit on the maximum size of a row. It is two words less than the page size (ZF2).

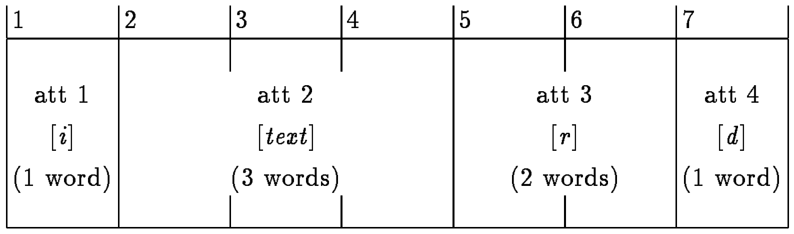
Each data row has its values plus two additional words which precede the actual data. These words contain the following:

**1** a pointer to the next row in the relation. If no more rows follow this row, this pointer is 0.

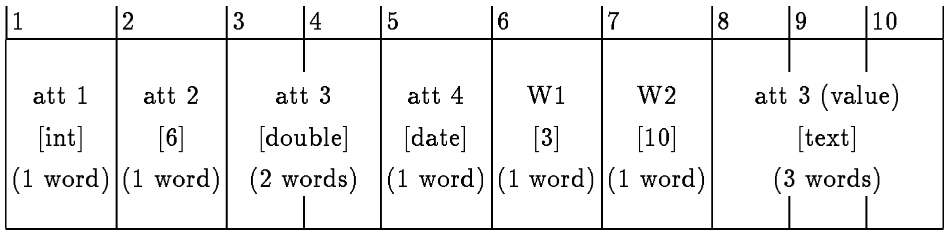
**2** length in words for the actual data row.

The format for each row consists of the actual data values in a one to one correspondence with the attributes as defined for the relation. All numeric values such as INTegers, REALs, and DOUBles are stored in a binary format which is standard for the machine. Text values are stored as ascii-text.

A tuple containing an integer (value *i*), a 10-character text item (value *text*), a double (value *r*), and a date (value *d*) occupies 7 words and looks like this:



Variable length attributes are the exception to this rule. They have a one-word pointer at their position in the tuple. This pointer contains the offset (from the start of the row) to the actual location of the variable attribute's data. At that offset a two word header contains length information. Following this is the actual value. The same roe, if the character string is variable length, occupies 10 words and looks like this:



The variable length header (words W1 and W2 in the example) contains

| Data type | W1 | W2 |
|---|---|---|
| text | # words | # chars |
| scaler | 1 | 0 |
| vector | # words | # cols |
| matrix | # words | #rows |

**I think**

This file contains the B-tree records and multiple occurrence tables for support of keyed attributes. The header is shown in table 3.5. It contains some identification information and pointers for the other pages. All actual B-tree data begins in other records.

The records on file 3, with the exception of the first record, are either B-tree node records or multiple occurrence table records. Each attribute in each relation which is a 'key' attribute has its own set of B-tree node records. All key attributes share space on the multiple occurrence table records.

The tree structure used in RIM is similar to other B-tree structures but has several unique features which make it ideal for use in RIM. For each key attribute, the value of **ATTKEY** in the attribute table of file 1 has the record number of the top or starting B-tree node record on file 3. Each node record has pointers which point to either other node records or to relation rows on file 2. Each node record is actually a two dimensional array with 3 rows and 42 columns. The length of records on file 3 must be a value which can be divided evenly by 3.

A B-tree node as an array with 3 rows and 42 columns has information for a maximum of 42 values i.e., the order of B-tree is 42. The first row has the

| item # | fortran name | length (words) | field contents |
|---|---|---|---|
| 1 | KDBHDR | Z | a string containing "**RIM DATABASE**" |
| 2 | | 1 | version of Rim creating this database |
| 3 | DBNAME | Z | name of the database. often just the filename |
| 4 | OWNER | Z | owner password |
| 5 | DBDATE | 1 | date the database was last modified |
| 6 | DBTIME | 1 | time the database was last modified |
| 7 | LENBF3 | 1 | length of each page in words. |
| 8 | LF3REC | 1 | next available page for adding a new B-tree record |
| 9 | MOTREC | 1 | current record used for adding multiple occurrence pointers |
| 10 | MOTADD | 1 | next available word in the MOTREC record for adding multiple occurrence pointers |
| 11 | LF3RCH | 1 | free-page record chain (not presently used) |
| 12 | LF3MCH | 1 | free-page record chain (not presently used) |
| | | — | The remainder of the first page is filled with zeros. |

Table 3.5: File 3 header

actual key values which are always stored in increasing order. This value is one word in length and contains the standard machine-dependent binary representation for integer and real attributes, and only the first word for double precision and text attributes. The second row has pointers which can represent one of three kinds of values. If this value is negative, its absolute value is the number of the next record as one traverses down a B-tree. If the value in this second row is positive, its meaning depends on whether the value in row three is zero or not. If the value in row 3 is zero, then row 2 has a packed number with the record number and word offset for the file 2 page with the row which corresponds with this key value. The value for the key attribute in this row of file 2 page matches the key value found in row 1 of the B-tree node. If the value in row 3 is not zero, then the value in row 2 has a packed number (with record number and word offset) which points to the end of the multiple occurrence table for this key value and row 3 has a packed number which points to the start of the multiple occurrence table for this key value.

The multiple occurrence table records are made up of linked lists of pointer pairs. Each pair has two pointers, the first of which points to the next pointer pair, and the second of which points to data on file 2. The pointers are packed values with record numbers and word offsets. The end of the link list is marked by zero in the **next** pointer field.

By having pointers to the start and end of the multiple occurrence list means that adding another occurrence of a key value does not involve the overhead of traversing the entire linked list. It is just as easy to add the fifteenth occurrence as it is the fifth. A record size of 126 allows for 63 pointer pairs on each multiple occurrence record.

When scanning a B-tree node the values are always in increasing order. This allows us to scan a node without looking 'two ways' at each value. The scan stop when we get the the special flag for the end of the list which is ZTEND. The node records are either nodes with all columns having node pointers or all columns having multiple occurrence and file 2 pointers. Nodes with multiple occurrence and file 2 pointers are sometimes called leaf nodes.

## Rime

The Rim file editor (Rime) is a useful tool for examining and correcting Rim databases. It allows you to examine the data files on a word or record basis. You can display in a convenient style the header and relation, attribute, and link tables of file 1, the header and data records of file 2, and the header and key records of file 3. You can change the value of any word in the database.

Rime is not a general user tool—its purpose is to investigate and correct Rim database problems. The Rime user is expected to have a comprehensive knowledge of Rim file structures. One of the operations you may perform with Rime, for example, is to un**remove** a table. You can do this by making the relation record's forward pointer positive. It is negative for a **remove**'d table.

Rime accepts the following commands. Many operate on a "current page", which is set by the two commands **file** and **page**. Most displays include integer, text, and data pointer interpretations. Rime's only error message is terse and uninformative.

**open** *filename* ⟨options⟩

opens a database identified by *filename*. This is comparable to the open command of Rim.

**header** #

displays the header record of file #. This does not affect the current file or current page.

**file** #

sets the current file to #.

**page** #

sets the current page to # in the current file. **file** must be set first.

$$\textbf{relations}\left\langle \begin{array}{c} \textit{name} \\ \textit{row\#} \\ . \end{array} \right\rangle$$

displays a relation record. You must be editing file 1. Omission of the parameter displays the next relation in the table. A particular relation can be selected with

*name* displays the named relation.

*row#* displays the relation at the specified address. *row#* is a relation pointer.

. displays the first relation in the table.

$$\textbf{attributes}\left\langle \begin{array}{c} \textit{name} \\ \textit{row\#} \\ . \end{array} \right\rangle$$

displays an attribute record. You must be editing file 1. Omission of the parameter displays the next attribute in the table. A particular attribute can be selected with

*name* displays the named attribute. The attribute is 'looked for' in the current relation.

*row#* displays the attribute at the specified address. *row#* is an attribute pointer.

. displays the first attribute in the table.

$$\textbf{links}\left\langle \begin{array}{c} \textit{name} \\ \textit{row\#} \\ . \end{array} \right\rangle$$

displays a link record. You must be editing file 1. Omission of the parameter displays the next link in the table. A particular link can be selected with *name* displays the named link.

*row#* displays the link at the specified address. *row#* is a link pointer.

. displays the first link in the table.

$$\textbf{tuple} \left\langle \begin{array}{c} \textit{word} \\ \textit{word page} \\ . \end{array} \right\rangle$$

displays a tuple. You must be editing file 2. Omission of the parameter displays the next tuple in the table. A particular tuple can be selected with

*word* displays the tuple starting at *word*.

*word page* displays the tuple starting at *word* on page *page*.

. displays the first tuple in the current relation.

*word count*

displays selected words from the current page. *word* is the starting word address in the page. *count* is the number of words to display. Words are displayed as integers, text, and pointers.

**store** *word value*

changes the contents of the current page at *word* to *value*. If *value* is not an integer then it is assumed to be text. If a text value is longer than ZCW characters then more than one word in the page will be affected.

**real** *value*

displays the value as a real. If $< \textit{value} > \leq 4096$ it is assumed to point to a word in the current page.

**double** *value*

displays the value as a double precision real. If $< value > \leq 4096$ it is assumed to point to a doubleword in the current page. Otherwise a **double** acts just like **real**.

**text** *value*

displays the value as ascii-text. If $< value > \leq 4096$ it is assumed to point to a word in the current page.

**date** *value*

displays the value as a date. If $< value > \leq 4096$ it is assumed to point to a word in the current page.

**time** *value*

displays the value as a time. If $< value > \leq 4096$ it is assumed to point to a word in the current page.

**itoh** *value*

displays the value as a datafile pointer (the ITOH function). If $< value > \leq 4096$ it is assumed to point to a word in the current page.

**htoi** *word page*

displays the pointer elements as an integer (the HTOI function).

**set** *options*

sets running parameters—just like Rim's **set** command.

**show** *options*

shows running parameters—just like Rim's **show** command.

**parameters**

displays some of the compilation parameters defined in SYSPAR. These are not the parameters of **set** and **show**.

**system** *command*

sends the *command* to the operating system for execution. *command* must be a single string.