mst_rdb

Edition 1.0.9, for version 1.0.9
9 February 2023

Richard J. Edgar, Diab Jerius, Terry Gaetz

Table of Contents

1 Introduction

The `mst_rdb` package allows the reading of database files in `/rdb` format. These files consist of optional comments beginning with pound signs (`#`), a header containing the field names of the columns, and rows of `ASCII` data delineated by tabs. There are two data types, strings and numeric. The `mst_rdb` library allows access to both types of data fields.

The reading of an `/rdb` table consists of the following steps:

- read the header from an already opened file; see Section A.1.1 [rdb_rd_hdr], page 3.
- map the header, selecting the fields you wish to read, and passing information about the location within a structure where you wish the information to be placed. There are two functions supplied to accomplish this task; see Section A.2.1 [rdb_map_cols_arst], page 3, and Section A.2.2 [rdb_map_cols_stst], page 6.
- Find out how many lines there are in the datafile, if desired; see Section A.3.1 [rdb_count], page 8.
- read the data, one line at a time, and process it; see Section A.6.1 [rdb_col_read_st], page 10.
- close the file and free allocated memory space; see Section A.7.1 [rdb_free_map], page 11, and Section A.7.2 [rdb_free_hdr], page 11.

There are several ways in C of specifying a complex data structure, and eventually this library will support a number of them. For now, there is one supported method for each of the above steps. The suffixes of the function names indicate what method is being used to specify the data structures; `ar` stands for array, `st` stands for `struct`, and `va` stands for `varargs`.

It is presumed that the user wishes to select certain columns to read, and has established an array of `struct`s, or a single `struct`, into which the data are to be placed. The header mapping routine connects the properties of this `struct` to those of the `/rdb` file, selecting desired columns by name, and informing the software of the data type (string or numeric) for each desired field. The user then passes the reading routine a pointer to a struct of the agreed type, and the reading routine obtains data from a single line of the file, and places the desired fields into the struct. If numeric, the data are put into a `double` in the `struct`. If string, space for the string is allocated dynamically, and a pointer to this string is put into a `char *` field of the `struct`. It is important for the user to remember to free the memory allocated for unused strings.

When the user is done with the data structures used in reading an `/rdb` file, they should be freed. Use the normal `free()` call on all strings returned by the column reading function, and then call first `rdb_free_map()` and then `rdb_free_hdr()` to free the memory associated with the column map and header data structures respectively.

Appendix A Functions

A.1 Reading /rdb File Headers

A.1.1 rdb_rd_hdr

Parse an /rdb file header prior to reading.

Synopsis

```
#include <mst_rdb/mst_rdb.h>

rdbHeader *rdb_rd_hdr(FILE *fin);
```

Parameters

`FILE *fin` the stream to parse

Description

Rdb_rd_hdr extracts column names from a stream connected to an rdb data file. It leaves the file pointer set to the first line after the header. Lines preceding the header which are empty or begin with '#' are ignored.

Returns

This routine returns a pointer to a dynamically allocated rdbHeader structure (filled in with appropriate values). It returns NULL if there are no header records. Upon error it prints a message to stderr and exits. When the user is finished with these data structures, their memory should be freed with the rdb_free_hdr() function.

Author

Diab Jerius

A.2 Mapping /rdb Columns

A.2.1 rdb_map_cols_arst

Create a map between user requested columns and those in an /rdb file. The 'arst' signifies that the function is called with [1] ARrays of column names and type arguments, and [2] expects subsequent reads to go into elements of a STructure.

Synopsis

```
#include <mst_rdb/mst_rdb.h>

DataColumnMap_st *rdb_map_cols_arst(
  rdbHeader const *hdr,
  unsigned long ncols,
  char *col[],
  RDB_Type const type[],
  size_t const data_offset[]
);
```

Parameters

`rdbHeader const *hdr`
: the rdb header

`unsigned long ncols`
: the number of columns to read

`char *col[]`
: an array of pointers to strings holding the names of the columns to map

`RDB_Type const type[]`
: an array of type specifiers; allowed values are RDB_String, RDB_Num (i.e. double)

  Possible values for a `RDB_Type const` are as follows: `RDB_String`, `RDB_Num`

`size_t const data_offset[]`
: offsets into struct for data values

Description

When reading an /rdb file, the user may wish data to be returned in an order different from that in the data file. This routine takes a set of column names and creates a map between it and the names read from an /rdb header. The map is used by other /rdb data input routines.

Returns

a pointer to a dynamically allocated DataColumnMap structure (filled in with appropriate values). It prints a message to stderr and exits if a user supplied column is not found in the /rdb hdr. When the user is finished with the data structures allocated here, their memory should be freed with the rdb_free_map() function.

Author

Diab Jerius, Richard J. Edgar

Note

It is generally cleaner to use rdb_map_cols_stst().

Example

```
/* mst_rdb: test main program for mst_rdb library
   9/18/95 rje */
```

```
#include <stdio.h>
#include <stddef.h>
#include <string.h>

#include <tracefct/tracefct.h>

#include <mst_rdb/mst_rdb.h>

int
main(int argc, char* argv[])
{
    FILE *infile;

    typedef struct {
char *mirror;
double x0;
double y0;
double z0;
double rho0;
double p;
double k;
    } Tilt;

    Tilt data;
    long nlines;

    rdbHeader *head;
    DataColumnMap_st *map;

    char *fields[] = {
       "x0",
       "y0",
       "z0",
       "mirror",
       "rho0",
       "p",
       "k"
    };
    unsigned long nfields = 7;

    RDB_Type types[] = {  RDB_Num, RDB_Num, RDB_Num, RDB_String,
RDB_Num, RDB_Num, RDB_Num };
    size_t offsets[7];

    tf_init(argv[0],0,-1);
    infile = fopen("tilt.rdb","r");

    offsets[0] = offsetof(Tilt, x0);
    offsets[1] = offsetof(Tilt, y0);
    offsets[2] = offsetof(Tilt, z0);
    offsets[3] = offsetof(Tilt, mirror);
    offsets[4] = offsetof(Tilt, rho0);
    offsets[5] = offsetof(Tilt, p);
    offsets[6] = offsetof(Tilt, k);

    head = rdb_rd_hdr(infile);

    map = rdb_map_cols_arst(head,nfields,fields,types,offsets);

    nlines = rdb_count(infile,head);
    printf("There are %lu lines in database.\n", (unsigned long) nlines);

    while(rdb_col_read_st(infile, head, map, &data))
    {
      /*
...
      */
    }
    rdb_free_hdr(head);
    fclose(infile);

    return EXIT_SUCCESS;
}
```

A.2.2 rdb_map_cols_stst

Create a map between user requested columns and those in an /rdb file. The 'stst' signifies that the function is called with [1] a STructure containing column names and type arguments, and [2] expects subsequent reads to go into elements of a STructure.

Synopsis

```
#include <mst_rdb/mst_rdb.h>

DataColumnMap_st *rdb_map_cols_stst(
  rdbHeader const *hdr,
  unsigned long ncols,
  RDBFieldStInfo const fields[]
);
```

Parameters

`rdbHeader const *hdr`
: the rdb header

`unsigned long ncols`
: the number of columns to read

`RDBFieldStInfo const fields[]`
: an array of RDBFieldStInfo structs containing names of the columns to map, data types (string or numeric), and offsets into the user-supplied target struct for each column to be read

Description

When reading an /rdb file, the user may wish data to be returned in an order different from that in the data file. This routine takes a set of column names and creates a map between it and the names read from an /rdb header. The map is used by other /rdb data input routines.

The RDBFieldStInfo struct is defined in <mst_rdb.h> as follows:

```
typedef struct
{
char *name;
RDB_Type type;
size_t offset;
} RDBFieldStInfo
```

There is a macro defined in <mst_rdb.h> to aid in filling up an array of these structures; it is called RDBentry(name, type, structid). Here is an example of how to use it:

```
typedef struct{
  double x;
  double y;
  char * string
} my_rdb_struct;
RDBFieldStInfo fields[]={
  RDBentry( x,      RDB_Num,      my_rdb_struct),
  RDBentry( y,      RDB_Num,      my_rdb_struct),
  RDBentry( string, RDB_String,   my_rdb_struct),
};
#define NFIELDS ( sizeof(fields) / sizeof(RDBFieldStInfo) )


infile = fopen("my_database.rdb","r");
head = rdb_rd_hdr(infile);
map = rdb_map_cols_stst(head, NFIELDS, fields );
```

Returns

a pointer to a dynamically allocated DataColumnMap_st structure (filled in with appropriate values). It prints a message to stderr and exits if a user supplied column is not found in the /rdb hdr. When the user is finished with the data structures allocated here, their memory should be freed with the rdb_free_map() function.

Author

Diab Jerius, Richard J. Edgar

A.3 Counting Lines in the /rdb File

A.3.1 rdb_count

Grab columns of data from an rdb data file.

Synopsis

```
#include <mst_rdb/mst_rdb.h>

long rdb_count(
  FILE *fin,
  rdbHeader *hdr
);
```

Parameters

`FILE *fin`	the stream to parse
`rdbHeader *hdr`	the rdb file description

Description

rdb_count counts the lines in an /rdb-format table file which has previously been opened, had its header read, and had columns mapped (using functions elsewhere in this library). On exit, the file is left ready to read the first data element.

Returns

The routine returns the number of data records in the file. If the file cannot be rewound, it returns -1.

Author

Diab Jerius

A.4 Testing for the existence of a column with a given name

A.4.1 rdb_is_column

Check whether /rdb header includes column "name".

Synopsis

```
#include <mst_rdb/mst_rdb.h>
```

```
int rdb_is_column(
  rdbHeader const *hdr,
  char const *name
);
```

Parameters

`rdbHeader const *hdr`
: the rdb file description

`char const *name`
: header name sought

Description

rdb_is_column compares the provided character string against the names of the /rdb columns.

Returns

The routine returns 1 if name is a column name, 0 otherwise.

Author

Terry Gaetz

A.5 Rewinding an /rdb File

A.5.1 rdb_rewind

rewind database to first data row

Synopsis

```
#include <mst_rdb/mst_rdb.h>

int rdb_rewind(
  FILE *fin,
  rdbHeader *hdr
);
```

Parameters

`FILE *fin` the stream to parse

`rdbHeader *hdr`
: the rdb file description

Description

rdb_rewind repositions the database such that the next read will return the first row in the database.

Returns

Upon success it returns 0. If the file cannot be rewound, it returns 1. Upon error it returns -1.

Author

Diab Jerius

A.6 Reading Data from the /rdb File

A.6.1 rdb_col_read_st

Grab columns of data from an /rdb data file.

Synopsis

```
#include <mst_rdb/mst_rdb.h>

int rdb_col_read_st(
  FILE *fin,
  rdbHeader *hdr,
  DataColumnMap_st const *map,
  void *data
);
```

Parameters

`FILE *fin` the stream to parse

`rdbHeader *hdr`
the rdb file description

`DataColumnMap_st const *map`
the columns to read

`void *data`
where to stow the data. must be pointer to an appropriate struct

Description

The rdb_col_read_st function extracts data from an /rdb file on a stream. It reads the next record on the stream, extracting data via the specified column mapping and fills in a user supplied structure.

An NULL string is returned if an RDB_String column contains no data, the user must still free the pointer to the NULL string. An empty RDB_Num column is flagged as an error if it is supposed to contain data. Empty lines are ignored.

When reading string arguments, this routine places into the target data structure a pointer to a dynamically allocated string area, into which the string is copied. When the user is finished using the string, this area should be freed.

Returns

The routine returns 1 if it successfully read data, 0 upon end of data. If the number of columns in the record is not the same as in the header, or if a (numeric) column contains garbage, a message is printed to stderr and the program is halted.

Author

Diab Jerius, Richard J. Edgar

A.7 Freeing Data structures

A.7.1 rdb_free_map

Frees the memory associated with an /rdb column map.

Synopsis

```
#include <mst_rdb/mst_rdb.h>

void rdb_free_map(DataColumnMap_st *map);
```

Parameters

`DataColumnMap_st *map`
    the column map to free

Description

rdb_free_map - Frees the memory associated with an /rdb column map.

Author

Richard J. Edgar

A.7.2 rdb_free_hdr

Frees the memory associated with an /rdb header.

Synopsis

```
#include <mst_rdb/mst_rdb.h>

void rdb_free_hdr(rdbHeader *hdr);
```

Parameters

`rdbHeader *hdr`
    the header to free

Description

rdb_free_hdr - Frees the memory associated with an /rdb header.

Author

Diab Jerius