RBIO USER'S GUIDE

CHUCK DEBAUN

OSS/UAS

COMPUTING DIVISION

FERMI NATIONAL ACCELERATOR LABORATORY

REVISED: April 1996

ABSTRACT

RBIO (Raw Bufferred I/O) Library provides a FORTRAN and C language callable, medium independent I/O interface across UNIX platforms.

Table of Contents

Revision Record

Disclaimer

1. Introduction

- **1.1. Rbio Features**
- **1.2. Rbio Non−features**

/ul

2. Utilization

- **2.1. Compilation and Linkage**
- **2.2. Execution**

3. Definitions

3.1. Job File Numbers

- **3.2. The iostat Variable**

- **3.3. File definition**
- **3.4. FORTRAN Error and End−of−file Returns**
- **3.5. FORTRAN Subroutine Calls**
- **3.6. C function references**
- **3.7. Symbolic Names**
- **3.8. Tape Parameters**
  - **3.8.1. Search Criteria**
  - **3.8.2. Extending the Tape**
  - **3.8.3. Tape Data Buffering**
- **3.9. Replication of Standard I/O Devices (RBFOPEN only)**
- **3.10. Line Buffering**

4. General Purpose Routines

- **4.1. rbparm (Read Parameter File)**
- **4.2. RBFOPEN (General Purpose Open − verify vsn, if tape)**
- **4.3. RBFCLOSE (General Purpose Close)**
- **4.4. RBREAD (General Purpose Read)**
- **4.5. RBWRITE (General Purpose Write)**
- **4.6. RBFNAME (Get Path to File)**
- **4.7. RBJFNST (Get State of JFN)**
- **4.8. RBFTYPE (Get Device Type of JFN)**
- **4.9. RBSET (Set Various RBIO Internal Flags)**
- **4.10. RBLDAS (Append String to String)**
- **4.11. RBLDAN (Append Integer to String)**

- **4.12. RBGETVAL (Get Parameter/Environment value)**
- **4.13. RBSETVAL (Set Parameter Value)**

5. File System Only Routine

- **5.1. RBSEEK (Seek on File – Not Tape)**

6. Tape Related Routines

- **6.1. RBMOUNT (Verify Tape vsn)**
- **6.2. RBOPEN (Open File – Tape Only)**
- **6.3. RBCLOSE (Close dataset – tape only)**
- **6.4. RBUMOUNT (Unload tape vol)**
- **6.5. RBREWIND (Rewind Tape volume)**
- **6.6. RBLKLEN (Get Block Size)**
- **6.7. RBRECLEN (Get Record Length)**
- **6.8. RBVSN (Get VSN of Current Tape)**
- **6.9. RBDSNAME (Get Tape File Name)**
- **6.10. RBFORMAT (Return Format of File)**
- **6.11. RBDSEQN (Return Dataset Sequence number)**

7. Examples

- **7.1. File Definition String**
- **7.2. RBLDAS and RBLDAN**
- **7.3. File Copy Program**

8. Error Codes

9. Parameter Files

- **9.1. Parameter File Statements**

- **9.2. Use of the Internal Flag**
- **9.3. Parameter File Example**

10. Questions and Answers about the rbio Library

Revision Record

July 1991 Original draft.

December 1992 Second Revision.

April 1996 Third Revision. Reflects the addition of C language support and dependence upon ftt.

Disclaimer

This document and associated documents and programs, and the material and data contained therein, were developed under the sponsorship of an agency of the United States government, under D.O.E. Contract Number EY−76−C−02−3000 or revision thereof. Neither the United States Government nor the Universities Research Association, Inc. nor Fermilab, nor any of their employees, nor their respective contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for accuracy, completeness or usefulness of any information, apparatus, product or process disclosed, or represents that its use would not infringe privately−owned rights. Mention of any specific commercial product, process, or service by trade name, trademark, manufacturer, supplier, or otherwise, shall not, nor is it intended to, imply fitness for any particular use, or constitute or imply endorsement, recommendation, approval or disapproval by the United States Government or URA or Fermilab. A royalty−free, non−exclusive right to use and disseminate same for any purpose whatsoever is expressly reserved to the U.S. and the U.R.A. Any further distribution of this software or documentation, parts thereof, or other software or documentation based substantially on this software or parts thereof will acknowledge its source as Fermilab, and include verbatim the entire contents of this Disclaimer, including this sentence.

1. Introduction

The **rbio** library allows the FORTRAN or C program to perform buffered I/O on a file object, which may be a regular UNIX file or a tape dataset. If the file object is a tape dataset, label processing will be done transparently to the calling program. Label processing will be performed even if the request is for an unlabeled tape to ensure that a labeled tape is not mounted. The calling program can open a *symbolic name*. possibly defined as an environment variable, and let the library obtain the value of the name. Parameter files that conditionally define symbolic names can be used.

1.1. Rbio Features

- Label processing. **Rbio** creates, and processes **ANSI X3.27** (ISO 1001) standard labels. This provides some level of data protection and allows access to data on tape by name, as well as by position. This also assists data portability because other systems which recognise **ANSI X3.27** (ISO 1001) standard labels can process the tapes in a similar manner.

- **Rbio** reads and writes unlabeled and Volume Header Only tapes.

- **Rbio** supports long names on tape by using a subset of the **DEC VMS** extension to the **ANSI X3.27** labeling standard.

- **Rbio** allows the use of *symbolic variables* which are to be evaluated during execution. These variables may be defined in the **UNIX** environment, or conditionally within a *parameter file*.

- **Rbio** provides device independent access to files and file−like objects.

- **Rbio** provides FORTRAN callable string routines to ease the use of other rbio routines.

- **Rbio** supports C language interfaces. These are referrenced in a manner similar to the FORTRAN interfaces.

- As a portability aid, **rbio** uses **ftt** as its low level tape interface. Should the tape driver interface semantics change, only relinking with the latest version of the **ftt** library would be required.

- To avoid confusion, **rbio** has a *stubs* library containing *dummy* routines for other libraries which "may" be linked in. For instance, **rbio** internally calls routines from the **OCS** library, these routines are only available if **OCS** is installed and the library is linked into the executable. The *stubs* library avoids this issue by containing *dummy* routines which would otherwise be called from the **OSC** library and return values recognized by **rbio** as *not valid*. The values returned are such that **rbio** recogizes that **OCS** is not available.

1.2. Rbio Non−features

There are many things which **rbio** is *not*. Some of these "Non−features" are listed below:

- **Rbio** is not part of any operating system. Because of this, any protection afforded by the label processing of **rbio** is dependent upon the use of label processing, supplied by **rbio** or some other facility, by *others*.

- **Rbio** does not *block* data records on output, or *unblock* data records on input.

- **Rbio** does not recognize, or support **IBM "standard" (EBCDIC)** tape labels.

- **Rbio** does not support **ISO 4341** *short header tape* format. It does recognise it, however.
- **Rbio** does not write **ISO 4341** *unlabeled tape* format. It can read it.
- **Rbio** does not support **ANSI x3.27**(ISO 1001) tape *permissions* or passwords.
- **Rbio** does not reset any drive specific parameters.
- **Rbio** does not perform tape drive *allocations*, nor does it *interface* to any mechanism to perform allocations.
- **Rbio** does not require the use of a tape allocation scheme. See **CAUTION**
- **Rbio** does not prompt operators to mount tapes.
- **Rbio** does not support partitioned tapes.
- **Rbio** does not translate data.

2. Utilization

The utilization of the **rbio** library involves both the compilation and linkage phase as well as the execution phase. These will be treated separately

2.1. Compilation and Linkage

To compile and link with **rbio** versions *v9_0* and higher requires the following:

```
set up ftt
set up rbio, usually setup rbio is sufficient.
if desired, set up OCS.
compile and link your FORTRAN code using

     f77 ... $RBIO_LIB ... $RBIO_STUBS
or   f77 ... $RBIO_LIB $OCSLIB ... $RBIO_STUBS

In AIX the following is recommended:

     f77 -gextname ... $RBIO_LIB ... $RBIO_STUBS
or   f77 -gextname ... $RBIO_LIB $OCSLIB ... $RBIO_STUBS

The C language construct is remarkably similar

     cc ... $RBIO_LIB ... $RBIO_STUBS
or   cc ... $RBIO_LIB $OCSLIB ... $RBIO_STUBS
```

2.2. Execution

The following shell level commands are useful, if using tapes:

```
setup ftt
setup ocs

setenv TAPE_MGR cps_tape
                        Tell rbio to use cps tape drive naming conventions.

setenv TAPE_LABELS VMS
                                          To create VMS labels

setenv TAPE_LABELS ANSI
                                          To inhibit production of VMS tapes
```

use **ocs, cps** or some other tape allocation mechanism to allocate an appropriate tape drive and send mount request to operator. Most allocation schemes permit something like

```
setenv MYTAPE `ocs_tape ...`
```

which will assign the path associated with the allocated drive to the environment variable MYTAPE. MYTAPE can then be manipulated either at the script(shell) level or within the C or FORTRAN code to become a full tape *file−definition* The **rbio** routine **rbgetval** can be used to retrieve the value from within FORTRAN or C.

CAUTION

Although **RBIO** no longer requires the use of a tape drive allocation mechanism, it is the only portable method, in a multi−user environment to prevent corruption of tape data.

3. Definitions

The following defines the terms used in the remainder of the document.

3.1. Job File Numbers

When a file is opened by **rbopen** or **rbfopen**, it is associated with a job file number (JFN). All subsequent use of this file will use the JFN. A JFN is an integer in the range 0−99, inclusive.

- JFNs are not FORTRAN logical unit numbers.
- JFNs are not C file descriptor numbers.
- JFNs are not recognized by the standard C or FORTRAN I/O statements, and unit numbers and file descriptors are not recognized by these routines.

CAUTION

Although the same number can be a unit number and a **JFN**, doing so will make the program more confusing than need be. It is strongly recommended that this practice be avoided. The number of files that one process can have open is limited by the system. Usually, the limit is 20 files. This limit applies to the count of all unit numbers, file descriptors and **JFN**s. In addition, the package uses two files internally when it is necessary to request the next volume in sequence.

3.2. The iostat Variable

All of the routines provide a condition code to show the status of the requested I/O operation. The content of the variable at the time of the routine call is ignored. If the operation was successful, a *zero* value will be assigned to this variable. If a read encountered an end−of−file, a *minus one* will be assigned to this variable. If an error occurs, a *positive integer* indicating the error will be assigned to this variable.

3.3. File definition

The *file−definition* string fully defines the file, or dataset, to be processed. In its simplest form, it is merely the name of a normal UNIX file system file. It may, however, contain additional information which will affect the processing of the data. In these cases, the first portion of the file definition string will be a name, or full pathname, of a normal UNIX file system file which may be associated with a tape device. The remainder of the string is a colon(":") delimited list of parameters as follows:

LDR
: Line Delimited Record, or *line buffered*. This is the format commonly written to a terminal, or printer.

UFORT
: FORTRAN unformatted record format. This is the format written by a FORTRAN unformatted write, and expected by a FORTRAN unformatted READ. It is supported in RBIO to provide the capability of having record structures on disk(within normal UNIX file system files)

any of the tape parameters.
: See Tape parameter discussion.

3.4. FORTRAN Error and End−of−file Returns

All FORTRAN alternate returns are optional. If the calling program does not provide alternate return statement numbers, the subroutine will always return control to the following statement. The presence or absence of alternate returns has no effect on the setting of the iostat variable.

If the subroutine is called with one alternate return, control will be transfered to the named

statement if an error occurs.

If rbread is called with a second alternate return, control will be transfered to the named statement if an end−of−file is encountered.

In other cases where a subroutine can have a second alternate return, the meaning of the second return is explained in the text of the subroutine description.

CAUTION

FORTRAN alternate returns are provided for historical reasons, usage is not recommended.

This may become a portability issue. Some FORTRAN compilers may not support alternate returns in the usual manner. This could lead to an **rbio** failure which would be difficult to trace. So far all FORTRAN compilers have behaved in the expected manner, however, it can't be guaranteed.

3.5. FORTRAN Subroutine Calls

The subroutines in the **rbio** library can be called with arguments which are variables or constants. The subroutine calls are designed to accept the standard arguments as produced by the FORTRAN compiler. Avoid the use of **%VAL** or **%REF** in argument passing to the subroutines in the library. Arguments which are passed in a non−standard manner will not be recognized correctly by the library subroutines.

Avoid the use of **%LOC** in argument passing to these subroutines. Where character arrays are called for, the subroutines also require length information which is passed implicitly by the **CALL** statement. A datum referenced by **%LOC** does not contain this information. If the FORTRAN compiler provides pointers, do not use them as parameters for the same reason.

3.6. C function references

In general, the arguments to the C function entry points are *pointers*. Details are included with the individual function descriptions.

3.7. Symbolic Names

Symbolic names allow the user to specify *file−definitions* at run−time, instead of having to hardcode them into the program. If the first character in *file−definition* is a **$**, routines that use *file−definition*s interpret the rest of *file−definition* as a symbolic name.

If one or more parameter files have been read, the subroutine will search the symbolic names defined in these files. If no parameter files have been read, or if the symbolic name is not found among the names defined in the file(s), the name is treated as an environment variable. If the name is not defined in the environment, an error occurs.

The value of the symbolic variable **MUST** be a *file−definition*. It may not be, or contain, another symbolic name.

3.8. Tape Parameters

If the value of the *file definition* contains a colon(":"), and does not define a line buffered file or a FORTRAN unformatted file, everything after the first colon is considered to be tape related parameters. These are given as a colon separated list of specifications, with each specification of the form *code*=*value*. The following are the valid tape specification codes; unless otherwise noted, they are recognized by **rbfopen** and **rbopen** and ignored by **rbmount**: >p>

Volser
: The volume serial number of the tape to be used. If there will be more than one volume, this must be a comma−separated list of volume serial numbers. This field is required for any tape. It is recognized by **rbmount** and **rbopen**. It is ignored by **rbopen**

Type
: The label type. (Only the first character is significant) This field defaults to **A**nsi, and may contain one of the following:

  Ansi
  : **ANSI X3.27(ISO 1001)** labels

  Unlabeled
  : Unlabeled

  Nolabels
  : Unlabeled

  Vol_hdr_only
  : Contains only an ANSI x3.27 VOL1 Header. Files are delimited as if tape was unlabeled.

Dataset
: The dataset name. This field is required when opening the data set for writing on labeled tape. It is ignored for unlabeled or volume−hdr−only tapes.

Sequence
: The dataset sequence number. This will be matched against the sequence number written in the data set header, if tape is ANSI labeled. It is the dataset ordinal for non− ANSI tapes. It is an error if the dataset does not exist and the tape is to be read. If a dataset does not exist with the requested sequence number and the tape is to be written, the file will be added at the end of the tape. If a new data set is written at the end of the tape, its sequence number will be one greater than that of the previously existing last data set, regardless of the number given in this field. Consequently, the phrase **S=10000** guarantees that the data set will be added to the end of the tape (as long as this is permissible; see Extending the Tape). This field is required when opening the tape for writing. If reading, defaults to "next" file.

Generation

The dataset generation number. Defaults to 1.

Number
: The version number. If the dataset is to be written, and the version number is not specified, defaults to 0001.

Format
: The format code of the records on the tape. This must be one of **F** (fixed) or **U** (unblocked). Ignored if the data set is opened for reading, because the information will be extracted from the data set header.

Block
: If the format is **F**, this is the block length. If the format is **U**, this is the block length of the largest record. Ignored if the data set is opened for reading and tape is ANSI labeled.

  Required if the data set is opened for writing or tape is not ANSI labeled.

Record
: Specifies the record length for fixed (**F**) format data sets.

  Required if the data set is opened for writing and is blocked; ignored otherwise.

Keep
: The retention period in days to be given to the new data set. Ignored unless the data set is opened for writing. Does not affect the expiration of any data sets currently on the tape. The default retention period is zero days; the data set will be overwritable as soon as it is written.

Xtend
: The only valid value for this is NO. This prevents a new data set from being written if no existing data set header is matched (see Extending the Tape). Ignored unless the data set is opened for writing.

Only the first character of the code is significant; the following forms are identical:

```
D=GAMMA.DAT
Dataset=GAMMA.DAT
Dumb name=GAMMA.DAT
```

However, the second form will be helpful to people who have to work with the program, while the third form may be expected to be a hindrance.

3.8.1. Search Criteria

When the data set ID, sequence number, generation number or version number are given in the symbolic name, they will be matched against the data set header, if tape is ANSI labeled, to find the correct data set, if opened for reading.

If the data set is opened for writing only the sequence number is used to select the file. It is an error if the file exists and has not expired (only ANSI labeled tapes have expiration dates encoded).

If the data set is open for reading and no search criteria are given, the "next" data set on the tape will be matched.

3.8.2. Extending the Tape

If the tape is opened for writing and no data set is found which matches the search criteria, a new data set will be created at the end of the tape, if permissible.

This can be prevented by giving X=NO as a specifier.

If the tape is not to be extended and no data set at the specified location, an error is returned.

Attempting to overwrite a unexpired dataset will always yield an error.

3.8.3. Tape Data Buffering

When a JFN identifying a tape is read, an entire block is read internally from the device. No more than the requested number of bytes will be copied into the named data space. The remaining bytes are discarded.

When a JFN identifying a tape is written, it is an error to write more bytes than the declared maximum block size. In all other cases, the data presented to the write will be written in one tape block.

This means that, even for fixed block length data sets, it is possible to have uneven block lengths. It is the responsibility of the calling program to prevent this.

The **rblklen** subroutine provides a means of obtaining the block size defined for a data set opened with the given JFN.

3.9. Replication of Standard I/O Devices (RBFOPEN only)

The following *file−definitions* (which are valid in symbolic names) are defined:

*I
: The standard input will be opened. This is the same device that is attached to unit 5. This can only be opened for reading.

*O
: The standard output will be opened. This is the same device that is attached to unit 6. This can only be opened for writing. Modifiers will be ignored.

*E

The standard error output will be opened. This is the same device that is attached to unit 0. This can only be opened for writing. Modifiers will be ignored.

Descriptors for standard I/O devices opened in this manner are obtained with *dup(2)*. These devices may be opened in line buffered mode.

If FORTRAN conventional I/O (with a unit number) or C conventional I/O and raw buffered I/O (with a JFN) are used on the same device, the results are undefined. In all probability, doing this with standard error would interleave output in the order it was written; however, there is surely some way to invalidate this assumption.

3.10. Line Buffering

When a JFN opened in line buffered mode is read, the JFN is read through the next newline, or until end−of−file or an error is encountered. No more than the given number of bytes will be copied into the named data space. The remaining bytes are discarded. The new−line will never be copied into the data space.

When a JFN opened in line buffered mode is written, the contents of the data space are written to the JFN, followed by a newline.

4. General Purpose Routines

The routines in this chapter may be used with any device. They may also be used with the device specific routines where device specific behavior is needed.

4.1. rbparm (Read Parameter File)

FORTRAN: CALL RBPARM(*filename, iostat, *err−lbl*)

C: rbparm(*char* filename, int* iostat*);

Read symbolic name definitions from the named file. When a symbolic name is defined, it is in force until redefined, or until the end of the program.

4.2. RBFOPEN (General Purpose Open − verify vsn, if tape)

FORTRAN: CALL RBFOPEN(*jfn, file−definition, mode, iostat, *err−lbl*)

C: rbfopen(*int* jfn, char* file−definition, char* mode int* iostat*);

Open the *file−definition* with the given *mode* and associate it with the given *JFN*. For tapes this is equivalent to a **rbmount** reference followed by **rbopen**.

If the value of *file−definition* begins with **$**, it is interpreted as a *symbolic name*, and the value associated with the name is accessed to obtain the description of the file to open. It is an error to give a symbolic name for which there is no value defined.

If a colon is encountered in the (possibly resolved) file definition, it is interpreted to indicate that file parameters follow. Tapes are opened utilizing this method.

The *mode* is a character constant or variable whose contents are of the form *access/modifier*. access may be any of:

read
: Open the file definition for reading only. The only valid modifier with **read** is **exist**, which is implied.

write
: Open the file for writing only. In the absence of any modifier, **truncate** is implied.

update
: Open the file for reading and writing. Unless the **create** modifier is used, **exist** is implied.

append
: Open the file for writing only, but append new data to existing data. Unless the **create** modifier is used, **exist** is implied.

The modifier may be any of:

/exist
: Require the file to exist. The file will be opened as is.

/create
: Create the file if it does not exist; open the file as is otherwise.

/truncate
: Create the file if it does not exist; the contents of the file otherwise. After the call, the file will always be empty. This modifier is only available for **write**.

/new
: Require the file to **not exist**; it will be created. This modifier is only available for **write**.

Only the first character of access or modifier is significant. The following forms are identical:

```
w/t
write/truncate
```

The second form will improve the clarity of the code.

When a file is created, the permissions are determined by the process umask; see *umask(1)*.

4.3. RBFCLOSE (General Purpose Close)

FORTRAN: CALL RBFCLOSE(*jfn, iostat, *err–lbl*)

C: rbfclose(*int* jfn int* iostat*);

Close the named *JFN*. This is the full service close, and works for any JFN. If the *JFN* references a tape device, this has the effect of an **RBCLOSE** followed by an **RBUMOUNT**.

4.4. RBREAD (General Purpose Read)

CALL RBREAD(*jfn, buf, nbytes, rbytes, iostat, *err–lbl, *eof–lbl*)

C: rbread(*int* jfn, char* buf, int* nbytes, int* rbytes, int* iostat*);

Read at most *nbytes* bytes from the file named by the given *JFN* to the data area *buf*. Set the variable *rbytes* to the number of bytes actually read. If an error occurred or end of file was encountered, this value is undefined.

4.5. RBWRITE (General Purpose Write)

FORTRAN: CALL RBWRITE(*jfn, buf, nbytes, wbytes, iostat, *err–lbl*)

C: rbwrite(*int* jfn, char* buf, int* nbytes, int* wbytes, int* iostat*);

Write *nbytes* bytes from the data area *buf* to the file named by the given *JFN*. Set the variable *wbytes* to the number of bytes actually written. If an error occurred, this value is undefined.

4.6. RBFNAME (Get Path to File)

FORTRAN: CALL RBFNAME(*jfn, name, iostat *err–lbl*)

C: rbfname(*int* jfn, char* name, int* iostat*);

Load the file or device name associated with jfn into the character string name, which must be large enough to contain the file name. If name is too small, an error occurs.

4.7. RBJFNST (Get State of JFN)

FORTRAN: CALL RBJFNST(*jfn, state, iostat, *err–lbl*)

C: rbjfnst(*int* jfn, char* state, int* iostat*);

Loads the integer state with a code describing the state of the given JFN. The value of state will be one of:

0
: The JFN is disabled for further I/O. This will have happened because of a previous error.

1
: The JFN does not reference a labeled tape device.

2
: The JFN is mounted for reading; no dataset is open.

3
: The JFN is mounted for writing; no dataset is open.

4
: The JFN is mounted for reading; a dataset is open.

5
: The JFN is mounted for reading; end−of−file has been encountered on the open dataset.

6
: The JFN is mounted for writing; a dataset is open.

4.8. RBFTYPE (Get Device Type of JFN)

FORTRAN: CALL RBFTYPE(*jfn, typebuf, iostat, *err−lbl*)

C: rbftype(*int* jfn, char* typebuf, int* iostat*);

Load a string indicating the type of file associated with *jfn* into *typebuf*. *typebuf* must be a character string of at least one byte. One of the following values will be returned:

F
: Regular file

C
: Character special file

B
: Block special file

P
: Pipe

S
: Socket

4.9. RBSET (Set Various RBIO Internal Flags)

FORTRAN: CALL RBSET(*jfn, string, iosp*)

C: rbset(*int* jfn, char* string, int* iosp*);

Set various internal rbio flags. Determine subsequent behavior. *String* values:

"DEBUG"
: Turn on internal **rbio** debugging. *jfn* ignored.

"ANSI"
: Set **ANSI** tape indicator. Turns off **VMS** indicator. Does not process/create **VMS** extended labels. This can be overridden by contents of tape VOL1 label.

"VMS"
: Set **VMS** indicator. Processes/creates tape with **VMS** extended **ANSI** labels.

"UFORT"
: Set FORTRAN unformatted flag. Read/write as FORTRAN unformatted.

"UNLOAD"
: Unload tape from drive at **RBUMOUNT**.

"NOUNLOAD"
: Do *not* unload tape from drive at **RBUMOUNT**. Leaves tape rewound in drive.

4.10. RBLDAS (Append String to String)

FORTRAN: CALL RBLDAS(*string1, string2, string3, iostat, *err−lbl*)

C: rbldas(*char* string1, char* strings, char* string3, int* iostat*);

After the last non−blank character in *string1*, append the string **:***string2***=***string3*. The trailing blanks of *string2* and *string3* are omitted. It is an error if insufficient space remains at the end of *string1* to perform the operation.

If *string1* is not initialized, the results are undefined.

See below for the use of this subroutine.

4.11. RBLDAN (Append Integer to String)

FORTRAN: CALL RBLDAN(*string1, string2, num, iostat, *err−lbl*)

C: rbldan(*char* string1, char* string2, int* num, int* iostat*);

After the last non−blank character in *string1*, append the string **:***string2***=***num*. The trailing blanks of *string2* are omitted, and the value of the integer *num* is converted to a character string. It is an error if insufficient space remains at the end of *string1* to perform the operation.

If *string1* is not initialized, the results are undefined.

The subroutines RBLDAS and RBLDAN provide the ability to build an argument to RFBOPEN in the FORTRAN program.

The following program fragment shows an example of this method. See also.

```
C       These are the variables that will be used
        CHARACTER*100 FNAME
        CHARACTER*20 DSNAME
        INTEGER SEQNO
        INTEGER BLKSIZE

C       The program fills in DSNAME, SEQNO and BLKSIZE

C       The symbol TAPEDEV contains the name of the device to open
        CALL RBGETVAL('TAPEDEV', FNAME, IOSTAT, *311)
C       Will fail if TAPEDEV is not defined
        CALL RBLDAS(FNAME, 'D', DSNAME, IOSTAT, *312)
C       That put the data set name in the string
        CALL RBLDAN(FNAME, 'S', SEQNO, IOSTAT, *312)
C       That put the sequence number in the string
        CALL RBLDAN(FNAME, 'B', BLKSIZE, IOSTAT, *312)
C       That put the block size in the string
        CALL RBLDAS(FNAME, 'F', 'F', IOSTAT, *312)
C       That indicated fixed record size
C       The program will not have gotten here if any of the calls failed
        CALL RBFOPEN(25, FNAME, 'write', IOSTAT, *310)
```

See also.

4.12. RBGETVAL (Get Parameter/Environment value)

FORTRAN: CALL RBGETVAL(*name, value, iostat, *err−lbl*)

C: rbgetval(*char* name, char* value, int* iostat*);

Obtain the value associated with the symbol *name*, and copy the value into *value*. *name* and *value* are both character strings.

Do not preceed the symbol *name* with a *$*.

The symbol *name* can have been set or defined in a parameter file, or in the environment.

It is an error if *name* is not defined,

if *name* is longer than 20 characters,

if *value* is too small to fit the retrieved value.

4.13. RBSETVAL (Set Parameter Value)

FORTRAN: CALL RBSETVAL(*name, value, iostat, *err–lbl*)

C: rbsetval(*char* name, char* value, int* iostat*);

Associate the *value* with *name*. *name* and *value* are both character strings.

Do not preceed the symbol *name* with a *$*.

Symbols may be used in *value* by prefixing them with **$**, just as in a parameter file. The named symbol must exist, and the value of the symbol will be substituted for the symbol in the definition. For example, in the program fragment:

```
PSTR = "$CWD/OFILE"
CALL RBSETVAL("OUTPUT", PSTR, IOSTAT, *330)
```

the instance of **$CWD** will be replaced by its value in the definition of **OUTPUT**.

It is an error if:

- *name* is defined and marked readonly;
- *name* is longer than 20 characters;
- *value* is longer than 128 characters;
- The expansion of value by symbol subtitution is longer than 400 characters;
- A symbol in *value* is undefined.

5. File System Only Routine

5.1. RBSEEK (Seek on File – Not Tape)

FORTRAN: CALL RBSEEK(*jfn, location, iostat, *err–lbl*)

C: rbseek(*int* jfn, int* location, int* iostat*);

Reposition the current location of the file to be *location* bytes from the start of the file. *location*

must be an INTEGER*4. It is an error to attempt to reposition the file with a negative value of *location*, or attempt to reposition a device that cannot seek.

Note that RBSEEK cannot be used with files which are actually data sets on a tape.

If a file is repositioned beyond the current end of file and then written to, a hole will be left between the old end of file and the start of the later write. While this is not a threat to the integrity of the data, it can have bothersome and confusing side effects. In particular, UNIX utilities such as cp(1) will replace the missing bytes with nulls on the copied file, causing the size of the new file to bloat.

6. Tape Related Routines

6.1. RBMOUNT (Verify Tape vsn)

FORTRAN: CALL RBMOUNT(*jfn, file−definition, mode, iostat, *err−lbl*)

C: rbmount(*int* jfn, char* file−definition, int* iostat*);

Open the *file−definition* with the given *mode* and associate it with the given *JFN*. Do not locate a dataset to be opened.

File−definition will be processed as described in **RBFOPEN, but it must ultimately reference a tape device.**

mode is interpreted as described in **RBFOPEN**, but only **read** and **write** are valid. *No* modifiers are permitted.

6.2. RBOPEN (Open File − Labeled Tape Only)

FORTRAN: CALL RBOPEN(*jfn, file−definition, iostat, *err−lbl, *err2−lbl*)

C: rbopen(*int* jfn, char* file−definition, int* iostat*);

Search forward from the current position on the device associated with the given *JFN* until the requested dataset is located.

jfn must reference a tape device. It must have either been created with **RBMOUNT**, or last closed using **RBCLOSE**.

Control will be directed to *err2−lbl*, if given, if an error occurs which does not render the *JFN* unusable for further I/O operations.

6.3. RBCLOSE (Close dataset – tape only)

FORTRAN: CALL RBCLOSE(*jfn, iostat, *err–lbl*)

C: rbclose(*int* jfn, int* iostat*);

Close the current dataset on the given *JFN*, which must reference a tape device. The device is held open. If the *JFN* was mounted for reading, the tape is positioned to process the next dataset. If the *JFN* was mounted for writing, the tape is positioned beyond the trailer for the dataset just written.

6.4. RBUMOUNT (Unload tape vol)

FORTRAN: CALL RBUMOUNT(*jfn, iostat, *err–lbl*)

C: rbumount(*int* jfn, int* iostat*);

Close the named *JFN*, which must reference a tape device. Unload tape, if "UNLOAD" specified to **RBSET**.

6.5. RBREWIND (Rewind Tape volume)

CALL RBREWIND(*jfn, iostat, *err–lbl*)

C: rbrewind(*int* jfn, int* iostat*);

Close and rewind the tape, and reposition the tape for reading. The *JFN* must reference a tape device which is mounted. The tape is left in same condition as it would be after a mount. Requires **rbopen** call to continue processing.

6.6. RBLKLEN (Get Block Size)

FORTRAN: CALL RBLKLEN(*jfn, nbytes, iostat, *err–lbl, *noblk–lbl*)

C: rblklen(*int* jfn, int* nbytes, int* iostat*);

Set the variable *nbytes* to the block size associated with the file named by the given *JFN*. The *JFN* may be open for reading or writing.

If the *JFN* is valid but describes a file with no block size, *nbytes* will be set to *zero*, *iostat* will be set to the error code for this condition and control will be transferred to the statement identified by *noblk–lbl*, if given.

On any other error, the value of *nbytes* will be undefined, the value of *iostat* will be set to the appropriate error code and control will be transferred to the statement identified by *err–lbl*, if

given.

6.7. RBRECLEN (Get Record Length)

FORTRAN: CALL RBRECLEN(*jfn, nbytes, iostat, *err−lbl, *noblk−lbl*)

C: rbreclen(*int* jfn, int* nbytes, int* iostat*);

Set the variable *nbytes* to the record size associated with the dataset associated with the given *JFN*. The *JFN* may be open for reading or writing.

Only data sets with a *fixed* record format have a record size. Data sets having variable length blocks have a record size of zero. When called for a JFN naming an unblocked data set, *nbytes* will be set to *zero*, but the subroutine will return normally.

On any error, the value of *nbytes* will be undefined, the value of *iostat* will be set to the appropriate error code and control will be transferred to the statement identified by *err−lbl*, if given.

6.8. RBVSN (Get VSN of Current Tape)

FORTRAN: CALL RBVSN(*jfn, name, iostat*)

C: rbvsn(*int* jfn, char* name, int* iostat*);

Return the Volume ID of the currently mounted tape associated with jfn in *name*. If *name* is not large enough to contain the entire Volume ID, an error occurs.

6.9. RBDSNAME (Get Tape File Name)

FORTRAN: CALL RBDSNAME(*jfn, name, iostat, *err−lbl, *nods−lbl*)

C: rbdsname(*int* jfn, char* name, int* iostat*);

Return the data set name associated with *jfn* in the character string *name*, which must be at least 17 bytes long.

If *jfn* is not associated with a labeled tape device, the value of *iostat* will be set to the appropriate error code and control will be transferred to the statement identified by *nods−lbl*, if given.

6.10. RBFORMAT (Return Format of File)

FORTRAN: CALL RBFORMAT(*jfn, fmt, iosp*)

C: rbformat(*int* jfn, int* fmt, int* iosp*)

Return internal **rbio** format value in *fmt*, if tape. *Zero* otherwise.

6.11. RBDSEQN (Return Dataset Sequence number)

FORTRAN CALL RBDSEQN(*jfn, seqn, iosp*)

C: rbdseqn(*int* jfn, int* seqn, int* iosp*);

Return dataset sequence in variable *seqn*, if tape. *Zero* otherwise.

7. Examples

The following examples, from the most simplistic program fragment to a full program, indicate techniques which have been found to be useful.

7.1. File Definition String

This program fragment shows, in a very simplistic sense, the content of the file−definition string. Here the assumption is that a tape allocation command has been issued and the response, typed at the terminal was /abcd/dev/10019_nsv. Also the file to be read is DAT.FILE on tape ZB9999

```
CHARACTER*1000  CSTRING
INTEGER IOST, LREAD
CALL RBFOPEN(3, '/abcd/dev/10019_nsv:VSN=ZB9999:D=DAT.FILE','R',IOST)
IF(IOST .NE. 0) STOP
CALL RBREAD(3, CSTRING, 1000, LREAD, IOST)
CALL RBFCLOSE(3)
END
```

This obviously do−nothing program segment would do the following:

RBFOPEN would verify that the device pointed to by /abcd/dev/10019_nsv had tape ZB9999 in it. The tape label type would be verified and if labeled, the volume id would also be verified. Because the default label Type is ANSI, if the tape is not ANSI, the tape would not match. The tape would have been positioned to read file DAT.FILE. The operation is successful if IOST is zero.

RBREAD would have read no more than 1000 characters from DAT.FILE into CSTRING. The number of characters actually read would be placed in *LREAD*

RBFCLOSE would have disassociated JFN 3 from the tape. The tape would be rewound and unloaded.

7.2. RBLDAS and RBLDAN

This example is exactly that same in execution as the first one. It uses **RBLDAS** to build the file−definition string.

```
        CHARACTER*80 OPENSTR
        OPENSTR = '/abcd/dev/10019_nsv'
        CALL RBLDAS( OPENSTR, 'VSN','ZB9999',IOST)
        CALL RBLDAS( OPENSTR, 'DSN', 'DAT.FILE', IOST)
        CALL RBFOPEN(3, OPENSTR, 'r', IOST)
```

The RBREAD and RBFCLOSE calls haven't changed.

An immediate extension is to place the tape allocation in a shell script with stdout redirected so that FORTRAN could read in the pathname. Alternatively, one might use command substitution to assign the pathname to an environmental variable and use the rbio routine **RBGETVAL** to retrieve the value. It is advisable to check the value returned to be certain that it represents a fully qualified pathname. Hint: the first character is '/'.

There are situations in which it is necessary to compute numeric values for fields in the file−definition. For this reason there is a numeric analog of **RBLDAS** named **RBLDAN**. The following might be informative:

```
        INT = 31
        CALL RBLDAN( OPENSTR, 'S', INT, IOST)
```

If this were placed immediately before the **RBFOPEN** in the preceeding example the result would be

```
        OPENSTR = '/abcd/dev/10019_nsv:VSN=ZB9999:DSN=DAT.FILE:S=31'
```

If **RBFOPEN** were called with this string, the result would be *File not found*, if DAT.FILE was not the 31st file on tape ZB9999

7.3. File Copy Program

The following program shows a simple use of these functions, transferring data from an input file to an output file. Because both files are opened with symbolic names, they may represent ordinary files, or tape devices.

```
        SUBROUTINE CAT(IOK)
        INTEGER*4 BUFLN, IOC
        CHARACTER*50 FNAME
C       Unless everything goes perfect, the operation will be
C       considered to have failed
C
```

```
C       Hope no other part of the program uses JFNs 10 and 11
        CALL RBFOPEN(10, "$DATASRC", "read", IOK)
        IF(IOK .NE. 0) GOTO 300
        CALL RBFOPEN(11, "$DATADST", "write/trunc", IOK)
        IF(IOK .NE. 0) GOTO 301
C       Show the user the input file name, unless it is too big
        CALL RBFNAME(10, FNAME, IOC)
        IF(IOC .NE. 0) GO TO 99
        WRITE(6, 1000)FNAME
1000   FORMAT(1H , 'Reading from ', A50)
C       Find out if the input file is blocked, and if so,
C       how big a block
99      CALL RBLKLEN(10, BUFLN, IOC)
        IF (BUFLN.GT.512) GOTO 302
C       Cannot handle blocking of greater than 512 bytes
        IF (BUFLN.GT.0) GOTO 100
C       If not blocked, pick an attractive value
        BUFLN = 512
C       It is the caller's responsibility to ensure that the
C       output file conforms to the input file, if necessary
  100   CALL FCOPY(10, 11, BUFLN, IOK)
        IF (IOK.EQ.0) GOTO 303
        CALL RBFCLOSE(10, IOK)
C       Even if the close of the JFN fails, its hard
C       to imagine a problem when the file is open
C       for reading
        CALL RBFCLOSE(11, IOK)
        IF(IOK .NE. 0) GO TO 304
        RETURN

C       This is the error section; blurt messages to
C       standard error.  Since the subroutines them-
C       selves produce error messages, most of this
C       is window dressing
 300   WRITE(0, 3000) IOK
  3000   FORMAT('Unable to open input file; error ', I2)
        RETURN
  301   WRITE(0, 3001) IOK
 3001   FORMAT('Unable to open output file; error ', I2)
        GOTO 331
C       This message is rather important to the program
  302   WRITE(0, 3002) BUFLN
 3002   FORMAT('Input file, which is blocked at ', I5,
       * ' bytes, is too large')
        GOTO 330

C       In UNIX, files are closed when the process exits
C       This code goes out of its way to be neat
  303   WRITE(0, 3003)
 3003   FORMAT('Error during copy of file')
  330   CALL RBFCLOSE(11, IOC)
  331   CALL RBFCLOSE(10, IOC)
        RETURN
  304   WRITE(0, 3004) IOK
 3004   FORMAT('Failed to close output file; error ', I2)
        IOK = 0
        RETURN
        END

        SUBROUTINE FCOPY(JFNI, JFNO, NB, IOGOOD)
```

```
        INTEGER*4 RIO, DUMMY, IOS
C       Want BUF to be as big as the largest block could
C       possibly be
        CHARACTER*512 BUF
  200   CALL RBREAD(JFNI, BUF, NB, RIO, IOS)
        IF(IOS .EQ. -1) GO TO 205
        IF(IOS .GT. 0) GO TO 310
        CALL RBWRITE(JFNO, BUF, RIO, DUMMY, IOS)
        IF(IOS .NE. 0) GO TO 311
C       Generally, it will not be necessary to test how
C       many bytes were actually written, since anything
C       less than the proper amount will usually manifest
C       as an error
        GOTO 200
  205   IOGOOD = 1
        RETURN
  310   WRITE(0, 3010) JFNI, IOS
 3010   FORMAT('Error ', I2,' reading from JFN ', I2)
        RETURN
  311   WRITE(0, 3011) JFNO, IOS
 3011   FORMAT('Error ', I2,' writing to JFN ', I2)
        RETURN
        END
```

8. Error Codes

These values are returned in the iostat variable when errors occur.

1. File is already open. The JFN named is already open.

2. named file from being opened for the given access.

3. Named file is a directory. The named file is a directory.

4. Insufficient system resources. A system resource, such as the number of concurrently open files, has been entirely consumed.

5. File does not exist. The named file does not exist, and the access modifier of the open requires its existence.

6. Unknown error. A system call within the library failed with a value of errno which was not foreseen when the library was written.

7. Symbol does not exist. No environment variable was found matching the named symbol.

8. Cannot reopen device for writing. The device cannot be reopened for writing. This will happen if a tape is mounted read−only.

9. Error in use of JFN. The JFN is not open, or a previous error has made the operation invalid on the JFN.

10. File is too big. The write would cause the file to exceed the system limit; see ulimit(2).

11. Definition error. A syntax error has been found in the label definitions of a symbolic name.

12. File has not expired. The data set was opened for writing, and the expiration date has not passed.

13. Illegal open mode. The mode specified is unknown to rbopen, or the combination of the access mode and modifier is invalid.

14. Wrong volume serial number. The volser on the tape did not match the volser given in the label definitions.

15. No labels. The data set was opened for reading and no labels were found, or the JFN given to rbdsname is not associated with a labeled tape.

16. Bad labels. The tape is labeled, but a label was not found where it was required to be.

17. Data set does not exist. The data set was to be opened for reading, or the data set was to be open for writing and the tape was not to be extended, and the specified data set was not found.

18. No more core. The subsystem could not create buffers for I/O when opening the JFN, or could not create symbol storage or interpretation space.

19. End of medium. The system encountered a hard end of volume indication that could not be processed. On a write, this is because the blocking is less than the device−specific minimum necessary to be able to adjust backwards and write EOV trailers. On a read, this is because data is written after the end−of−tape marker; this data cannot be read.

20. No data set name or sequence. The data set is to be opened for writing, and the data set name or sequence number has not been specified (both must be given).

21. No more volumes to mount. A tape mount is necessary, but all of the volume serial numbers that were given in the definition have been used.

22. Not record structured. The rblklen or rbreclen subroutine has been called with a JFN that does not reference either a labeled tape or a variable record formatted file. This error does not preclude continued use of the JFN.

23. Parameter file error. An error was encountered by rbparm, and the instructions provided by the parameter file were to signal that type of error to the caller.

24. Cannot seek. The JFN is associated with a tape drive, terminal, or other device that does not support repositioning within the file.

25. Buffer too short for string. The buffer is too short to store the name that would be loaded, or the interpretation of a symbol exceeded the allowed buffer length.

26. String too long. A name has been given to rbgetval or rbsetval that is longer than 20 characters, or a value has been given to rbsetval that is longer than 63 characters.

27. Symbol is readonly. The symbol to be defined by rbsetval is defined and marked as readonly.

28. Cannot open no−rewind device. The label processing functions cannot determine the no−rewind form of the given device. May be caused by incorrect value in environment variable TAPE_MGR. For example, if using cps controlled devices, TAPE_MGR must have the value cps_tape.

29. Interpretation depth error. Too many levels of interpretation were required to resolve a symbolic name. The functions use this limit to trap circular definitions.

30. No block size specified. No block size has been specified when attempting to open a dataset on a labeled tape with RBOPEN or RBFOPEN.

31. Not an integer. A string which is expected to represent an integer does not do so.

32. No dataset is open. The JFN is valid, but no dataset has been opened on it.

33. Not a tape JFN. The JFN is valid, but does not reference a tape.

34. Cannot move backwards on sequential device. The specified sequence number would position the JFN before its current position, requiring the device to seek backwards.

35. Option not valid in this subroutine. An option has been given which is not valid in this call or context, although it is valid in another context. This differentiates from Definition error, which indicates that the specifier is not valid in any context.

36. Cannot read tape label block. An attempt to read a label record from a tape failed.

37. Cannot write tape label block. An attempt to write a label record to a tape failed.

38. Cannot reposition device. An attempt to reposition the tape failed.

39. Cannot close file or device. An error occurred while closing the file associated with the JFN.

40. Bad argument to system call. A system call failed due to a bad argument.

41. Feature is unsupported. An attempt was made to use a facility which is not supported in this version of the library.

42. Cannot request mount of subsequent volume. The attempt to request the next volume failed because the named device to which the request was to be written could not be accessed.

43. Request to mount subsequent volume denied. The operator indicated that the requested volume was not available.

44. Permanent Input/Output Error. An error was reported by the device during an input or output error. No other information is available.

9. Parameter Files

Parameter files provide the program with a facility by which they can set up I/O definitions without having to hardcode them in the program. Conditional evaluation of statements is provided to allow the parameter file interpreter to select from several definitions using environment conditions as criteria.

9.1. Parameter File Statements

The following statements can be included in parameter files:

comment text
: text is ignored.

ignore text
: text is ignored. Works like comment, but allows commentary to be differentiated from statements that should not be executed in this instance.

set symbol value
: Set symbol to value. If symbol is already defined, the new definition will override the old one, unless symbol is readonly, which causes a definition error.

  Where the forms $n or ${n} are encountered in value, the latter will be expanded by evaluating n as a symbol and replacing it with its definition. It is a definition error if n is undefined.

define symbol value
: Set symbol to value. If symbol is already defined as a symbolic name (not an environment variable), a definition error will occur and the value will not be changed. Expansion of value occurs as explained for set.

readonly symbol
: Set symbol readonly. If symbol is defined in the environment, the definition will be copied to the symbol table. If symbol is not defined at all, a definition error will occur.

iff

Perform the following statement if the internal flag is set.

ifnf
: Perform the following statement if the internal flag is not set.

ifdef symbol
: Perform the following statement if symbol is defined.

ifndef symbol
: Perform the following statement if symbol is not defined.

else
: Perform the following statement if the preceding if−family statement did not cause the statement following it to be performed. In the absence of any control instructions, an else statement will be paired with the lowest level if statement. Use the begin and end statements to control the pairing of else statements.

begin
: Begin a compound statement block. When placed after an if or else statement, the entire block is conditionally performed. It is an error to use a compound statement block in any other place.

end
: End a compound statement block.

break
: Break out of a compound statement block. This is only valid within a compound statement block, and is used where a condition is encountered which makes it undesirable to perform the rest of the block. When used in blocks in an if−else statement set, breaking out of the block after an if does not cause the statement(s) in the else clause to be performed.

negate
: Toggle the internal flag. See the discussion of the internal flag, below.

eq *value1 value2*
: Set the internal flag if *value1* is equal to *value2*.

neq *value1 value2*
: Set the internal flag if *value1* is not equal to *value2*.

access *type filename* Test the accessibility and/or *type* of the file. The internal flag will be set if all of the tests called for in type are true. Available values for *type* are:

C	True if *filename* is a character special device.
D	True if *filename* is a directory.

F True if *filename* is a regular file.

P True if *filename* is a FIFO.

R True if *filename* exists and is readable.

S True if *filename* exists and is not empty.

W True if *filename* exists and is writeable.

X True if *filename* exists and is executable.

More than one test can be specified in *type*; the specifiers should be separated by commas. Every test must be true for the internal flag to be set.

require *exitcode $symbol*

If *symbol* is not defined, terminate execution of the entire program. The program will exit with the value of *exitcode*.

abort *[exitcode [message]]*

Terminate execution of the entire program. If *exitcode* is given, the program will exit with this value instead of 1. If *message* is given, it will be written to the standard error output before the program terminates.

stop *[error]* Terminate reading the parameter file. If the option error is given, the call to **rbparm** will return the error *Error signalled in parameter file*; otherwise, it will return normally.

say *message*

Write *message* to the standard error output. Symbol names may be included in *message* by prefacing them with a $; these will be resolved when *message* is printed.

unset *symbol*

Remove the definition of *symbol* as a symbolic name. Environment variables cannot be removed.

on error *action*

If an error (other than a definition or syntax error) occurs, perform *action*. The default is to continue after errors. *action* can be any one of:

stop

Has the same effect as a stop error statement, but is only performed if an error is encountered.

abort exitcode

Causes the program to exit when an error is encountered, returning exitcode.

on deferr *action*

If a definition error occurs, perform *action*. *action* can be any action defined for on error.

The default is to continue after definition errors. The set, define and readonly statements can cause definition errors.

9.2. Use of the Internal Flag

The internal flag is used for comparison and parameter file interpretation control. The eq, neq, access and negate statements set or alter the flag, and the iff and ifnf statements test the flag. The eq, neq and access statements set the flag if the corresponding condition is met, but never clear the flag if the condition is not met. This allows several of these statements to be used in series, implementing an OR operation on the results of the tests. To construct a logical AND operation, invert all the tests and the use of the result. For example, use neq instead of eq and ifnf instead of iff. The negate statement complements the internal flag. This can be used to construct somewhat more advanced logical combinations. It will be necessary to set flag variables to construct very complex logical relationships. Note that all statements except comment, ignore, eq, neq, access and negate clear the internal flag. The iff and ifnf statements clear the flag after it has been tested. This feature resets the flag for subsequent operations.

9.3. Parameter File Example

The following parameter file shows examples of some of the features described above. The symbols TARGETS and RUNDATA are to be opened for input, and output is to the file named by the symbol OUTFILE. comment Blank lines are allowed comment Do not continue after errors (except definition errors) on error stop comment PROJLIB is a symbol other symbols will depend on comment so it has to be there require 1 $PROJLIB readonly PROJLIB comment If the input symbols are not defined in the environment, comment define them now ifndef TARGETS begin set TARGETS $PROJLIB/targets.dat set OUTFILE $PROJLIB/results.dat end ifndef RUNDATA set RUNDATA $PROJLIB/wfile comment If not already done, define OUTFILE define OUTFILE output.dat This file is used by passing its name to rbparm. When the file is successfully processed, control is passed to the next FORTRAN statement.

10. Questions and Answers about the rbio Library

- *How do allocation routines relate to rbio?*

  Most UNIX−like operating systems have historically treated tape devices as SHARABLE, this leads to data corruption. Allocation routines exist in order to ensure that a UNIX session has EXCLUSIVE access to a tape device. Most allocation routines output, on stdout, the pathname to the allocated device. This is the pathname to be passed to rbio routines RBMOUNT, RBFOPEN, or RBOPEN.

- *How do I make rbio recognize VMS tape labels?*

  If the tape already has VMS extended ANSI tape labels, you need do nothing. Rbio will recognize this and act accordingly.

  The creation of a VMS labeled tape is controlled by the environment variable

TAPE_LABELS. If TAPE_LABELS is VMS , rbio will create VMS compatible tape labels when the file is written.

- *Doesn't rbio perform buffering?*

  Rbio (Raw Buffered Input/Output) is buffered, but does not perform blocking. Thus, a RBREAD call for a portion of a record discards the remainder. A RBWRITE call writes the record, as supplied, to the tape.

  Because the tape device may report an error if an attempt is made to read a partial block, the entire record is read. The buffer exists to protect the caller from possible memory overwrite on input, and rbio from a possible overwrite on output.

- – *Why is there a RBFOPEN and a RBOPEN?, Wouldn't one do?*

  RBFOPEN is the device independent open routine. It is a combination, for tape, of RBMOUNT and RBOPEN.

- *Why is there a RBFCLOSE and a RBCLOSE?*

  As with RBFOPEN/RBOPEN, RBFCLOSE is the device independent close routine. RBCLOSE terminates a tape dataset, while RBFCLOSE terminates processing of the tape. RBFCLOSE is a combination, for tape, of RBCLOSE and RBUMOUNT. However, combinations of RBCLOSE followed by either RBFCLOSE or RBUMOUNT are often seen in tape processing and do no harm.

- *What is the difference between RBMOUNT and RBUMOUNT?*

  The primary purpose of RBMOUNT is to verify to identify the label type of the tape and, if labeled, the volume ID of the tape.

  The primary purpose of RBUMOUNT is to terminate output on tape and, optionally(see rbset),unload the tape.

- *What is the difference between RBFNAME and RBDSNAME?*

  RBFNAME returns the pathname to the tape device.

  RBDSNAME returns the filename of the file on the tape.

- *Is there any way to find out which volume is currently mounted?*

  Yes. RBVSN will return the volume id of the currently mounted tape.

- *What happens if my program does not close the JFN for an open tape dataset?*
  - The tape will remain in the tape drive, and may have its data modified.

- The allocation routine may not be able to deallocate the device interfering with the use of the tape drive.
- If the dataset is being read, the tape will not be rewound.
- If the dataset is being written, it will have no file mark at the end and no trailers. Most other systems will have a problem reading this.

- *What happens if the program closes the dataset but does not call RBUMOUNT for the jfn?*
  - The tape will remain in the drive, and may have its data modified.
  - The allocation routine may not be able to deallocate the device interfering with the use of the tape drive.
  - If the device is being written, the trailers will not be terminated by a double end−of−file.
  - The tape might not be rewound.
- *Can I call rbopen on a JFN for a dataset that was opened but never closed?*

  No. The error that will occur is File is already open.
- *Can I open a dataset with rbmount and rbopen, and use rbfclose to close it?*

  Yes. It is also possible to open a dataset with rbfopen and close it with rbclose and rbumount, or close it with rbclose and get another dataset with rbopen.
- *If rbumount is called with a JFN having an open dataset, will it work?*

  Yes. In this case, rbumount will implicitly perform the rbclose function.
- *I want to assemble the parameters to rbopen in the FORTRAN program. Can I just catentate FORTRAN strings to put together the device name and dataset parameters?*

  Not reliably, because FORTRAN string catentation includes the trailing blanks on the end of each string. This is the purpose of rbldas and rbldan.
- *What is the purpose of specifying a record length when opening a dataset for writing?*

  The target system on which the tape will be read may need this information. The rbio library writes record lengths in the header, but never uses them.
- *I want to write a lot of data to a tape, but I do not want more than one volume to be created. How can I cause the software to return an error when the tape is full?*

Only supply one volume serial number with the V= specifier.