CNWRA 91-005

RDFREE: A FORTRAN UTILITY FOR FORMAT FREE INPUT

USER'S GUIDE

Prepared for
Nuclear Regulatory Commission
Contract NRC-02-88-005

Prepared by
R. W. Janetzke
B. Sagar

Center for Nuclear Waste Regulatory Analyses
San Antonio, Texas

June 1991

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF EXAMPLES

RDFREE: A FORTRAN UTILITY FOR FORMAT FREE INPUT

USER'S GUIDE

ABSTRACT

The RDFREE utility is designed for creating a user friendly data input mechanism for FORTRAN programs. It consists of a suite of subroutines written in ANSI standard FORTRAN 77. These subroutines are designed for reading alphanumeric data without the usual format constraints that are inherent to FORTRAN. That is, the RDFREE utility can be used to read data without specifying any format type (e.g., I, E, F, A, or G formats) and generally (with some exceptions) without constraints of sequence. The primary motivation for developing RDFREE at the Center for Nuclear Waste Regulatory Analyses (CNWRA) was to aid in the development of user friendly computer codes for performance assessment of high-level nuclear waste repository. The basis of RDFREE is to associate alphanumeric data with appropriate keywords. The keywords and the data associated with each keyword are selected by the developer of the FORTRAN application program. The keywords may be partitioned into sets, each set being valid for a different part of the applications program.

The users of RDFREE should be aware that while RDFREE is self-contained and that knowledge of its inner workings is not mandatory, never-the-less its successful use will require some knowledge of programming in FORTRAN. The interface to RDFREE is via a subroutine, a function, and a set of common blocks. That is, the use of RDFREE requires that a set of common blocks be set up in the applications program and calls to a subroutine and a function be made at appropriate places.

The relaxed format of RDFREE eliminates many data entry errors while the keyword structure permits the input records to be given in any order. This results in an easy-to-read data set. Keywords can be associated with English language modifiers, as well as numeric and string data types. Comment functions are available to clarify the meaning of the input data.

1. INTRODUCTION

The RDFREE computer program modules described in this report were developed to provide a user-friendly input mechanism for FORTRAN application programs such as those currently under development at the Center for Nuclear Waste Regulatory Analyses (CNWRA) and the Nuclear Regulatory Commission (NRC). The modules comprise a suite of subroutines and functions which can be called from the application program. The features distinguishing RDFREE from standard FORTRAN input include a user defined list of keywords, a repeat control parameter, and a mixed mode free form input of both keywords and data fields. For example, an array of data items can be assigned a given value by including a repeat factor equal to the size of the array before the data value.

Even though the user does not need to be concerned with the exact mechanisms of reading, parsing, and storing input data in RDFREE, as an aid to understanding, some details are provided through out the document. The RDFREE code is quite similar to a computer language compiler and has many of the intricacies usually found in them.

After providing some definitions in Section 2, the RDFREE utility is described in Section 3. Guidance for using RDFREE is provided in Sections 4 to 12. The list of FORTRAN names reserved for RDFREE is given in Appendix A. A demonstration program using RDFREE is included as Appendix B. A formal description of the RDFREE grammar is given in Appendix D. The RDFREE modules are written in ANSI standard FORTRAN 77 language. A complete listing of the source code is provided in the Appendix C.

2. DEFINITIONS

The collection of various terms used throughout this report and their definitions are provided in the following.

keyword	An alphanumeric string of from 1-64 characters of which the first character is alphabetic. Only the first 6 characters are recognized by RDFREE and are significant for data interpretation. All keywords must be included in the master keyword list. An entire logical record (see definition below) is associated with a keyword.

COORDINATES may be a keyword with which grid coordinates of calculation points are associated. Note that only the COORDI (the first six characters) is recognized as the keyword by RDFREE. This means that for RDFREE all characters after COORDI (in this case NATES) are 'noise' characters. Noise characters help to provide English-like constructs for the input stream. Thus, even when COORDI is the keyword, writing it as COORDINATES does not cause an error.

Note that noise characters are not allowed in keywords made up of less than six characters. For example, if NAME is defined as a keyword, typing NAMES will cause an error, that is, only characters after the first six are ignored.

Example: Keyword.

modifier	A secondary keyword that helps to either modify or make more specific the meaning of the primary keyword. A modifier follows the primary (the first one) in a logical record (see definition below). All the rules

applicable to the construction of the keywords are also applicable to the construction of modifiers. Also, the modifiers must be included in the master keyword list.

> X can be modifier for the keyword COORDINATES. This modifier may be used to indicatc :hat the logical record of numerical values associated with the primary keyword (COORDI) are the x-coordinate values of the calculational points.
>
> *Example:* Modifier.

line The set of characters in columns 1-80. Data associated with a keyword may span any number of lines.

logical record Data associated with a primary keyword including the keyword itself, all modifiers and all of the alphanumeric data. The end of a record is indicated by a special character (or symbol) which has been chosen to be a vertical bar (¦) in RDFREE. In other words, a logical record is the set of characters from the first letter of the first keyword following an end of record to the next end of record symbol. It may include multiple lines, of which some may be blank.

> The logical record associated with COORDINATES may be:
>
> ```
> COORDINATES X  1, 2, 3 ¦
> ```
>
> *Example:* Logical Record.

end record symbol A symbol to indicate the end of a logical record associated with a primary keyword. This symbol in RDFREE is selected to be a vertical bar (¦).

master keyword list

An array containing all the keywords including modifiers. All keywords and modifiers must be strings with 6 characters or less, defined by the programmer to be the set of valid keywords to be recognized during the processing of a logical record. It may be a different list for each call to RDFREE.

noise word An alphanumeric string which does not appear in the master keyword list.

> If COORDI is the keyword, the logical record may be written as:
>
> ```
> COORDInates of calculational points are 1, 2, 3 ¦
> ```
>
> in which, 'nates of calculational points are' noise words. These are not included in the master keyword list. Note how the noise words may help to structure the input in a proper English construct.
>
> *Example:* Noise Word.

token A character or group of characters representing a basic syntactic element.

> A decimal point (.) or an equal sign (=) may be a token. These symbols must be discovered in the logical record for proper interpretation of the numeric data.
>
> *Example:* Token.

separator Symbols to indicate separation of a set of characters from other contiguous sets of characters. Five separator symbols are used in RDFREE. These are: blank ( ), comma (,), colon (:), equal (=), and tab (09h). These symbols are totally equivalent in RDFREE and can be used interchangeably. The reason for defining five rather than one symbol as a separator is the convenience these provide in constructing English-like constructs for input lines.

> A comma (,) and an equal (=) are defined as separators in RDFREE. These can be used to separate two numerical values, e.g.,
>
> ```
> COORDI = 12.3, 16.8 ¦
> ```
>
> *Example:* Separator.

numerical data Data that has numerical character, either integer or real. In using RDFREE, no attention need be made about the integer or real nature of the numerical data. Numbers can be typed in any form and RDFREE will change these into integers or real numbers at the time these are assigned to variables. The decimal, E, and D formats are acceptable for RDFREE input.

```
COORDINATES X =  1.e-1, 1.0, 2, 4 |
```

The asterisk (*) symbol is used to input multiple numeric data. e.g.,

```
HEIGHTS = 5*3.2 |
```

This means that there are five numerical data each equal to 3.2.

Example: Numerical Data.

alphabetic data Data that is made up of a string of alphabetic characters including numerals. In RDFREE, to distinguish alphabetic data from keywords which are also alphabetic, the alphabetic data is bounded by a single quotation mark (') in the beginning and a single quotation mark (') in the end. Names of files or names of radionuclides can be entered this way.

```
OUTPUT:  write on file 'MYOUTPUT' in a FORMATTED mode|
```

In this example, OUTPUT is the primary keyword, FORMATTED is a modifier, (:) is a separator, MYOUTPUT is alphabetic data input, and the rest are noise words.

Example: Alphabetic Data.

operators Two symbols are used as operators in RDFREE. These are the asterisk (*) and the single quotes ('). The purpose of * is to indicate multiple numerical and string values and that of the ' is to indicate that the string of characters (alpha and numeric), till the end quote (') is encountered, form an alphabetic data value.

comment	String of characters (alpha and numeric) that are not to be interpreted. The backslash symbol (\) is used in RDFREE to indicate comments, i.e., all character strings followi.ıg a \ symbol are taken as comments.

```
\coordinates are input in the next record
```

Example: Comment.

3. DESCRIPTION OF THE *RDFREE* UTILITY

The RDFREE library is a set of modules designed to enable free form input for FORTRAN programs. Library modules RDFREE and EXIST are the only ones which need to be called in the application program. The input data is allowed to be inserted without any regard to form (real or integer) and column location, thereby freeing the user from the task of counting columns in order to position the data. RDFREE also allows the use of keywords, which can be used as flags or identifiers for data which may follow. Errors are reduced on keyword entry by only requiring the correct spelling of the first 6 characters, while permitting the entire keyword to be given for readability. Also, RDFREE does not require that all data be in one input line. Any input may extend over multiple lines, however the end of line will be treated as a separator.

All alphanumeric strings are not recognized as keywords. A master keyword list is provided by the calling program to indicate which strings are to be considered valid. This allows for noise words to be present in the input to enhance readability, but ignored on processing so that only valid keywords are returned to the calling program. The master keyword list must be provided in upper case, since the input is converted to upper case before a comparison is performed in RDFREE.

The terms 'keyword' and 'modifier' can be used interchangeably. However, by convention the first keyword of a record is not considered a modifier, even though it has the same syntactical significance.

For inclusion in the application program, the programmer calls RDFREE with two parameters in the call list. The second parameter of the call list is an array which has the master list of keywords and the first parameter is the size of this array. At the call, RDFREE reads the next input record (which may be made up of many lines) until the end of logical record is encountered. Each record is analyzed character by character to generate tokens which are then parsed. The tokens are assigned one of the following types: keyword/modifier, numeric data, string data, or operator. The keywords are stored in the order they were encountered in the input and a counter is returned to indicate the number of keywords found. The numeric values are stored in a double precision array with pointers to indicate which ones belong to which keywords. The string values are stored in a character array with the required pointers.

A second module which may be used by the programmer is the logical function EXIST. By supplying a keyword, it returns a value of `.TRUE.` if the keyword was present in the input logical record. The common blocks contain sufficient information such that the programmer could make this check in a separate routine of his own design, but it is included as a convenience since the RDFREE routine also requires it.

RDFREE requires the definition of three FORTRAN I/O logical units in common variables. The input unit is used to read the free format input records. The output unit is used to echo the input records as they are read. The error unit is used to receive any error messages generated by the RDFREE system. The variable **`rderr`** is used to indicate the error number. Errors less than 0 indicate an end of file on input. Errors greater than 0 should not be ignored since they usually are the result of an ambiguous sequence of characters in the input. RDFREE cannot proceed correctly in this case and the errors should be considered fatal.

4. GENERAL HINTS FOR CONSTRUCTING *RDFREE* INPUT

A decision to use the RDFREE utility to input data in an application program means that all data (numeric and alphabetic) will be associated with keywords. It should be noted that the RDFREE mode of data input can be freely mixed with any other mode of data input, e.g., fixed format, although such mixing is liable to cause confusion. To use RDFREE mode of data input, the user must first settle on a list of keywords/modifiers and the associated numeric and alpha data that is to be input. Each keyword must be unique, i.e., it must always be associated with the same data fields. Obviously, the most representative names that are easy to remember should be selected as keywords. It is generally useful to minimize the number of primary keywords. This can be done by defining modifiers to associate more data with each keyword.

Each call to RDFREE will read one logical record. That means that a keyword, all modifiers and all numeric and alpha data in one record are read. At this point, the user can check the presence of specific keyword/modifiers by using the EXIST function. Based on which keyword/modifier is present in the logical record, the user can then assign the data to the FORTRAN variables in his/her program. It is obvious that the program should call RDFREE as many times as the number of logical records to be read. Usually the first logical record is associated with the keyword TITLE which generally defines the title of the problem. In order to indicate the end of the logical records, a keyword END or QUIT (or other appropriate one) can be defined. Once this keyword is encountered, the user can terminate calling RDFREE.

The following rules are followed in creating RDFREE compatible input files.

1. Each logical record begins with a keyword and ends with the terminating character (¦).

2. Blank lines are allowed and can be used to improve readability.

3. A comment may begin in any column with a back slash (\) symbol, but always ends at the end of line (column 80). However, any number of comment lines can be used. The terminator symbol (¦) need not be used for comment records.

4. An end of logical record terminating character (¦) is not recognized in a comment. That is, a logical record will not be terminated unless the terminating character (¦) appears to the left of a comment if both (¦) and (\) are used on the same line.

5. The maximum number of characters for a keyword is 64 of which only the first six are significant.

6. The minimum number of characters for a keyword is one.

7. TITLE is the only keyword that is presupposed in RDFREE; all others are defined by the user. Only string variables are associated with the TITLE keyword, i.e., no numerical record is read with this keyword.

8. If TITLE is to be used, then TITLE must be the first record. It is limited to one line (80 characters) and the terminator (|) symbol need not be used to indicate the TITLE's end.

9. The keyword input can be provided in either lower or upper case. Keywords are converted to uppercase in RDFREE. The master keyword list, however, must be in uppercase.

10. Only five separator characters are allowed [space ( ), equal (=), comma (,), tab (09h), and colon (:)], in addition to carriage return (end of line).

11. The total number of string data values (alphabetical data) associated with all keywords of a given logical record is limited by the parameter **MAXSVL**.

12. The total number of numeric data values associated with all keywords of a given logical record is limited by the parameter **MAXDVL**.

13. The * provides a repeat function, where the data value immediately following the * is repeated as indicated by the integer which immediately precedes the *.

14. The ' character indicates beginning of alphabetic data. The end of the alphabetic data is denoted by a matching ' character.

5. TYPES OF INPUT RECORDS

Four types of input records are recognized by the RDFREE utility: 1) title, 2) data, 3) comment, and 4) blank. In general, various record types may be used in any order with no unusual consequences except for the title record which is discussed in detail below.

5.1 TITLE RECORD

A title record is one which has the keyword **TITLE** in columns 1-5. Column 6 is ignored, and the rest of the line is associated as a string value with the variable **`TITLE`**. The string is stored in a common block for use by the user as appropriate. In general, the title line is the first line of the input file, but blank and comment lines do not interfere with its use if they appear before the title line. The fixed definition and position of the keyword is a departure from the usual RDFREE grammar where all keywords are user defined and can be placed anywhere in the input with few restrictions. The special handling of the keyword **TITLE** was implemented to be backward compatible with existing practices used in the modelling community. Namely that of using a title record as the first record of an ASCII file. This has resulted in a few subtle restrictions on its use.

```
TITLE Case 1, Scenario 2, Data set 3.
```

Example: Title Line.

The master keyword list must not include **TITLE**. Also, no keyword in the master keyword list should have its first five characters as TITLE. Under normal circumstances if the keyword **TITLE** does not appear in columns 1-5, it is ignored as a noise word. The **TITLE** keyword may be used more than once in a file if the preceding data lines are properly terminated with a vertical bar (¦), however there is no unique flag available to the calling program to indicate that a new title line has been encountered. It is possible to check the **`TITLE`** variable to see if it has changed. The modifier count, however will still be 0 in this case. (Note: A null data record will also return a zero for modifier count.) If the **TITLE** line interrupts a logical record in progress, the modifier count of the line is reset to 0 and subroutine **`rdfree`** will return immediately, and will assume that the next line after the title line is the start of a new logical record.

For those rare cases where **TITLE** is desired as a keyword it should not occupy columns 1-5 of any line unless it is the title line. To do so will cause the modifier counter to be set to 1, similar to the modifier count reset mentioned above. The discussion in this paragraph indicates that the **TITLE** keyword can have two meanings, 1) the normal meaning when it is in columns 1-5, and 2) a keyword/modifier meaning when it appears in columns 2-80. The first meaning will permit multiple occurrences of a title line as before, but will also cause **TITLE** to appear as the only keyword/modifier of the record. The second meaning will allow the keyword **TITLE** to appear anywhere in the keyword/modifier input record.

5.2 DATA RECORDS

Data records are any lines that have a keyword/modifier on them or a data value following a keyword/modifier. Blank and comment lines may be interspersed with data lines. Comments may also occupy the last field of a data record. Under normal conditions all data records must follow the title record.

```
RUN|
```

This one line logical record has a keyword of RUN and no modifiers.

```
GROUP1 PATH 1 2 3
       VELOCITY 1.E-4 1.E-5 1.E-6 |
```

This 2-line logical record has a keyword of GROUP1 and two modifiers, PATH and VELOCITY. The modifier PATH has 3 numeric values and the modifier VELOCI has 3 numeric values.

```
GROUP2 PATH
       1 \ x-axis
       2 \ y-axis
       3 \ z-axis
       |
```

This five line logical record has a keyword of GROUP2 and a modifier PATH. The PATH modifier has three numeric values on three separate lines with a comment field on each.

```
INPUT file='XYZ.DAT' |
```

This one line logical record has a keyword of INPUT, a noise word of 'file', a separator '=' and a string data value XYZ.DAT.

```
GROUP3 X_ARRAY 3*0.0 |
```

This one line logical record has a keyword of GROUP3 and a modifier X_ARRA. The modifier has 3 numeric values all of value 0.0.

```
HALF-LIFE = 2.E10 years |
```

This one line logical record has a keyword of HALF-L, a numeric data value of 2.E10, and a noise word 'years'.

Example: Data Lines.

5.3 COMMENT LINES

Comment lines begin with a backslash (\) in any column, but end at column 80. All characters are ignored in a comment line. Comments in a data line must come after all the data, that is, the entire record after the (\) is considered a comment.

```
\ This is a comment line.
```

Example: Comment Line.

5.4 BLANK LINES

Blank lines have no title, keyword, or data information. They may be all blanks or have a comment field following the leading blanks.

```

```

Example: A Blank Line.

```
                               \ A blank line with trailing comment field.
```

Example: A Blank Line with Comment.

6. INTEGRATION GUIDE USING FORTRAN FILES

The RDFREE utility may be integrated into either standard FORTRAN source files or preFOR (Janetzke and Sagar, 1991) command files. This section describes an integration procedure for the RDFREE utility using a standard FORTRAN source file. The list below details the procedure.

1. Check the external reference table for naming conflicts with the existing application. The application should not use any of the subroutine names or common block names that are reserved for RDFREE. Appendix A lists the reserved names for RDFREE.

2. Check the name of the parameters and variables in the RDFREE common blocks and the names of subroutine arguments for conflicts with existing variable names in the same module (See Appendix A).

3. Copy the RDFREE common blocks from file RDFCOM.FOR to the declaration area of all of the application routines which will call RDFREE.

4. Adjust the size of the arrays required by RDFREE. See Section 10 in this document called "SETTING RDFREE ARRAY SIZES".

5. Add a section of FORTRAN code to the modules which call RDFREE to check for the existence of the keywords and which sets the FORTRAN variables to the values returned by RDFREE.

6. Append .he RDFREE FORTRAN source from file RDFREE.FOR to the application source file.

7. Compile and link the resulting FORTRAN file.

Also see the sections which detail the use of subroutine RDFREE and function EXIST.

7. INTEGRATION GUIDE USING *preFOR* FILES

The preFOR is a utility -- a pre-processor for Fortran files developed at the CNWRA (Janetzke and Sagar, 1991). Certain preFOR commands (which are not accepted by standard FORTRAN compilers) can be embedded into a source code to make the development of the application program easier and more flexible. RDFREE may also be integrated into a program written for use with the preFOR utility. This section describes a procedure for integrating the RDFREE utility with a program written with preFOR commands. The list below details the procedure.

1. Check the external reference table for naming conflicts with the existing application. The application must not use any of the subroutine names or common block names that are reserved for RDFREE. Appendix A lists the reserved names for RDFREE.

2. Check the name of the parameters and variables in the RDFREE common blocks and the names of subroutine arguments for conflicts with existing variable names in the same module. (See Appendix A)

3. Insert the file RDFCOM.PRE into the fragment definition area of the application file.

4. Insert the following line into each subroutine which calls RDFREE.

```
*INSERT RDFCOM
```

Example: Using RDFREE common blocks with preFOR commands.

5. Adjust the size of the arrays required by RDFREE. See Section 10 in this document called "SETTING RDFREE ARRAY SIZES".

6. Append the file RDFREE.PRE to the application preFOR file.

7. Add appropriate calls to RDFREE and EXIST in the application code where required.

8. Run preFOR with the newly formed preFOR file as input.

9. Compile and link the resulting FORTRAN file.

Also see the sections which detail the use of subroutine RDFREE and function EXIST.

8. PROGRAMMER'S GUIDE FOR USING SUBROUTINE *RDFREE*

subroutine rdfree (lenmst, master)

NAME: **rdfree**

PURPOSE:

This module is part of a suite of routines used to analyze a free-form input stream in order to generate a consistent array of modifier names and associated data values.

ARGUMENTS:

input:

lenmst = INTEGER number of master keywords.

master = CHARACTER*6 array of master keywords.

Both arguments to RDFREE are input arguments and must be defined and supplied by the programmer. The first argument is an integer which indicates the size of the master keyword list. It must be greater than 0, but no upper limit is imposed by the RDFREE utility. The second argument is a character array containing all of the keywords which are valid for the next read of the input logical unit. It may be a different list for each invocation of the subroutine. Each element of the array has a character length of 6 which may include blanks, e.g., for keywords less than six characters long. All keywords in the master list must be in uppercase since RDFREE converts all input keywords (which can be in lower or upper case) to uppercase before analysis. An example of master keyword definitions via a **DATA** statement is given in program DEMO (See Appendix A).

```
INTEGER lenmst
CHARACTER*6 master(4)
DATA lenmst /4/
DATA master / 'KEYWD1', 'KEYWD2', 'RUN', 'HALF-L'/
.
.
.
CALL RDFREE (lenmst,master)
.
.
.
```

Example: Subroutine RDFREE Usage.

9. PROGRAMMER'S GUIDE FOR USING SUBROUTINE *EXIST*

`function exist (c,lenmst,master,elepos)`

NAME: **`exist`**

PURPOSE:

This subprogram is part of a suite of routines used with the data structures required by RDFREE and is used by RDFREE itself. The application program can also use it as a function to check for a specific modifier in the keyword/modifier **`mdfyr`** list returned from RDFREE. This routine searches the array **`master`** for an element equal to **`c`**. It then returns the element number in **`elepos`**. NOTE: **`c`** must be defined in the application program as CHARACTER*n, where n is greater than or equal to 6.

ARGUMENTS:

input:

`c`	=	CHARACTER*6 string to be matched to element of **`master`** keyword list.
`lenmst`	=	INTEGER length of the **`master`** keyword array.
`master`	=	CHARACTER*6 array of keywords to be searched.

output:

`elepos`	=	INTEGER element number of first matched string, 0 if string is not found to match any in **`master`**.
`exist`	=	LOGICAL if **`.TRUE.`** then string was found, if **`.FALSE.`** the string was not found.

Function **EXIST** is used to determine if a given keyword was present in the input logical record. Control is not returned from RDFREE until a logical end of record (vertical bar |) is found in the input. Then **EXIST** may be used to determine if any or all of the keywords were found in the logical record. The actual data values associated with a given keyword can be found by manipulating the values in the RDFCOM.FOR common blocks.

The first three arguments to **`EXIST`** are input arguments. **`c`** is the keyword for which an indication is desired as to its presence in the input record. If **`c`** is found in the list of keywords, **`EXIST`** will have a value of **`.TRUE.`** on return. Variables **`lenmst`** and **`master`** are used as the name of the second and third arguments in the program listing because **`EXIST`** is used internally to the RDFREE utility for that purpose. The user should not use these names when invoking the function, but rather the **`mcnt`** and **`mdfyr`** variable names which are provided in common.

The fourth argument is an output argument. It provides the position of the keyword/modifier represented by **c** relative to the beginning of the logical record. In this way the keyword will be in position 1 and the modifiers will be in positions 2 through n.

The function **EXIST** is of type logical. It will have a value of **.TRUE.** if the keyword in the first argument was found in the search list provided in the third argument. It will be **.FALSE.** otherwise.

```
CHARACTER*6 c
INTEGER     elepos
LOGICAL     lexist
.
.
.
c = 'GROUP1'
lexist = exist (c,mcnt,mdfyr,elepos)
```

Example: Function EXIST Usage.

10. SETTING *RDFREE* ARRAY SIZES

Use the following to set up and dimension the work space in the calling modules for the RDFREE routines. These parameters are provided in the same file as the RDFREE common blocks, RDFCOM.FOR or RDFCOM.PRE. They are not included in the common block definitions because parameters cannot reside in a common block.

MAXMOD is the dimension of the modifier string array **mdfyr**. This should be set to the maximum number of modifiers which will occur in a single logical record.

MAXDVL is the dimension of the double precision data value array **dvalue**. This should be set to the maximum number of numeric data values which will occur in a single logical record for all keywords.

MAXSVL is the dimension of the string data value array **svalue**. This should be set to the maximum number of string data values which will occur in a single logical record for all keywords.

MAXTOK is the dimension of the work array for **RDTOK**, it should be a minimum of (**MAXMOD + MAXDVL + 2*MAXSVL** + the number of noise words). Some cases may require a larger value to hold some transparent tokens in more involved data assignments.

11. PROGRAMMER'S GUIDE FOR USING THE *RDFREE* COMMON BLOCKS

In the discussion so far, little knowledge of the internal workings of the RDFREE utility is necessary to make use of the routines. However, most of the programmers attention in integrating the utility is given to the RDFREE common blocks. This is where the input data and all of the pointers are stored, as well as controlling parameters for the physical operation of the I/O procedures. This section describes all of the variables of the RDFREE common blocks. The variables are grouped according to their type, as required by the standard FORTRAN compilers.

11.1 COMMON /RDFINT/

```
INTEGER eunit
INTEGER iunit
INTEGER ounit

INTEGER firsti(maxmod)
INTEGER lasti(maxmod)
INTEGER mcnt
INTEGER rderr
INTEGER sordi(maxmod)
COMMON /rdfint/ eunit, iunit, ounit,
                mcnt, rderr, sordi, firsti, lasti
```

The following integer variables are required as input to the RDFREE utility.

eunit	=	INTEGER FORTRAN error output unit number. The error messages sent to this unit should not be ignored. Any error should be considered fatal to the correct transfer of data from the input stream into program variables.
iunit	=	INTEGER FORTRAN input unit number. This is the unit from which the free form input is read.
ounit	=	INTEGER FORTRAN output unit number. This is the unit which receives the echo of the input lines. These may be ignored on some systems by setting **ounit** to 0.

The following integer variables are output from the RDFREE utility.

firsti	=	INTEGER array whose elements point to the element of

		'svalue' or 'dvalue' which is the first data value for the corresponding modifier in 'mdfyr'.
`lasti`	=	INTEGER array whose elements point to the element of 'svalue' or 'dvalue' which is the last data value for the corresponding modifier in 'mdfyr'.
`mcnt`	=	INTEGER number of returned modifier names.
`sordi`	=	INTEGER array indicating svalue, dvalue, or none for corresponding modifier 'mdfyr'. sordi(m) = 0 if keyword name only. sordi(m) = 1 if values are stored in the svalue array. sordi(m) = 2 if values are stored in the dvalue array.
`rderr`	=	INTEGER read error status from RDTOK, which reads the free format input on unit 'iunit'. -1 = end of file on input 0 = normal return 1 = no keyword found in first position 2 = the character being handled by the token generator cannot be added to the current token due to rules violation.

```
LOGICAL LVAL
.
.
.
IF (SORDI(1) = 0) LVAL1 = .FALSE.
```

This FORTRAN fragment uses the logical variable 'lval1' to indicate that the first keyword in the 'mdfyr' array does not have any data values associated with it. Remember that the inex of the 'mdfyr' array has a one-to-one correspondence with the index of the 'sordi' array.

Example: Determining the Presence of Data Values.

11.2 COMMON /RDFCHA/

```
CHARACTER*6  mdfyr(maxmod)
CHARACTER*64 svalue(maxsvl)
CHARACTER*80 title
CHARACTER*64 token(maxtok)
COMMON /rdfcha/ mdfyr, svalue, title, token
```

The following character variable is required as input to the RDFREE utility.

token	=	CHARACTER*64 work array for token strings. This provides temporary storage while analyzing an input logical record.

The following character variables are output from the RDFREE utility.

mdfyr	=	CHARACTER*6 array of modifier names which were found in the input logical record.
svalue	=	CHARACTER*64 array of string data values to be associated with names in the 'mdfyr' array.
title	=	CHARACTER*80 string for the title.

```
CHARACTER*64 MYCHAR
.
.
.
MYCHAR = SVALUE(FIRSTI(2))
```

This FORTRAN code retrieves a string data value associated with the second element of the 'mdfyr' array which is the first modifier of the logical record. The actual position of the data in the 'svalue' array is stored in the 'firsti' array. Remember that the index of the 'firsti' and 'lasti' arrays has a one-to-one correspondence with the index of the 'mdfyr' array.

Example: Transferring a String Value from the 'svalue' Array to a Local Variable.

11.3 COMMON /RDFDP/

```
DOUBLE PRECISION dvalue(maxdvl)

COMMON /rdfdp/ dvalue
```

This is the numeric data array containing the values from the input logical record.

dvalue = DOUBLE PRECISION array of floating point values.

```
INTEGER MY_INT
.
.
.
MY_INT = IDINT(DVALUE(FIRSTI(2)))
```

This FORTRAN code retrieves an integer data value associated with the second element of the 'mdfyr' array which is the first modifier of the logical record. It is suggested that the intrinsic FORTRAN function IDINT be used when retrieving integers to eliminate any roundoff problems which may occur in the mixed mode assignment. The actual position of the data in the 'dvalue' array is stored in the 'firsti' array. Remember that the index of the 'firsti' and 'lasti' arrays has a one-to-one correspondence with the index of the 'mdfyr' array.

Example: Transferring an Integer Value from the 'dvalue' Array to a Local Variable.

11.4 SUMMARY

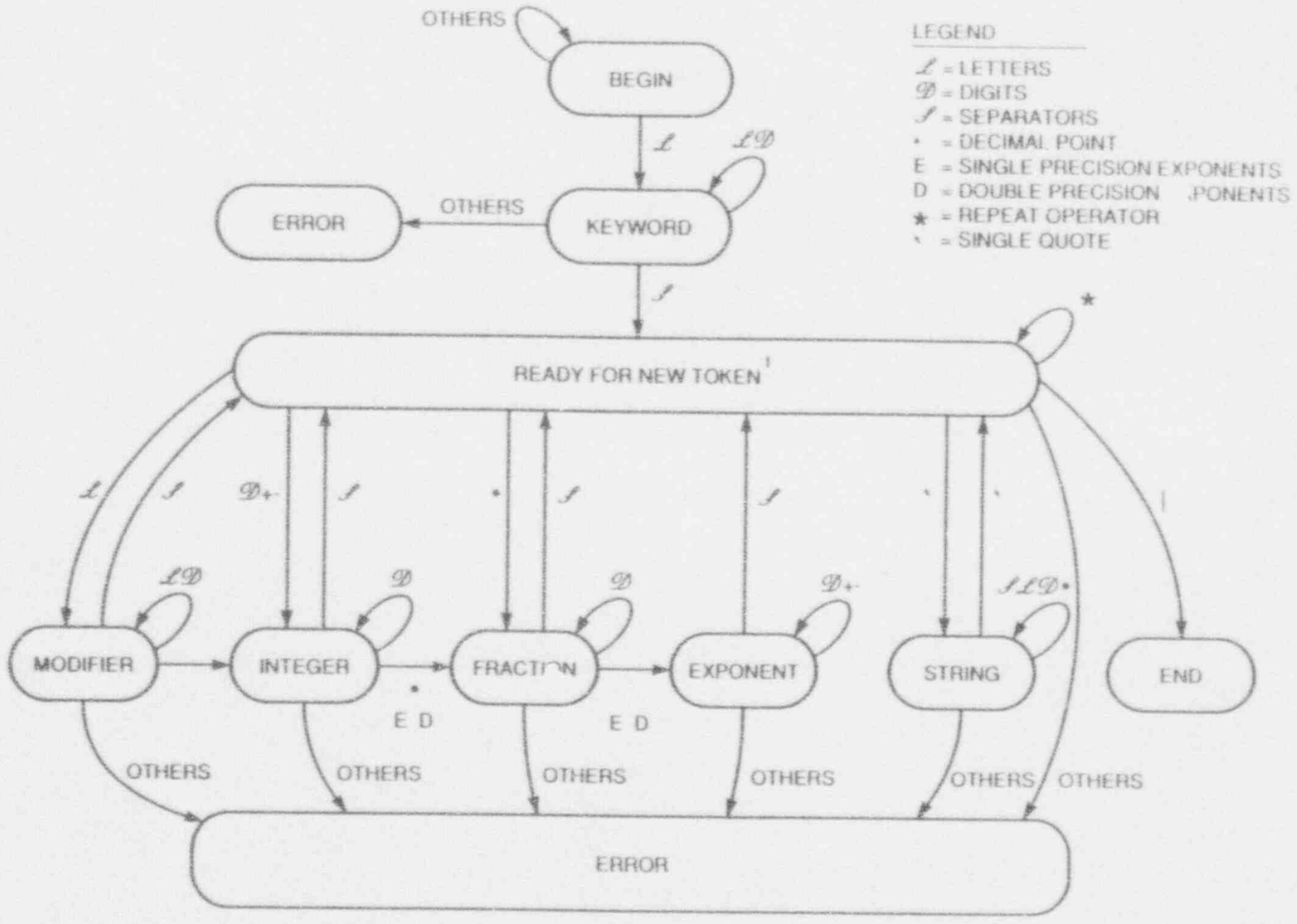
Pointer arrays 'firsti', 'lasti', and 'sordi' are returned to facilitate the extraction of the desired values from the two data arrays filled by RDFREE. An entry for each keyword found in the input line is made into each pointer array. The 'sordi' entry indicates which type of data is associated with the keyword (numeric, string, or neither). The 'firsti' entry gives the index of the start position of the data in the respective numeric or string array. And the 'lasti' entry gives the index of the stop position. Note all numeric data for all keywords for a given input line are returned in one array **`dvalue`**, so pointers must be used to locate the proper values. The procedures for extracting string values from 'svalue' are identical. For an example which uses all of the variables, see program DEMO in Appendix A.

12. GENERAL DESCRIPTION OF THE *RDFREE* PROCESSING

The programmer must define the I/O units in common **/RDFINT/** before calling RDFREE. When RDFREE is called, it calls RDTOK immediately upon entry. RDTOK then reads the next input record and analyzes it character by character. The characters in a comment field are not analyzed. RDTOK is a token generator that groups the input characters into tokens and stores the tokens in the 'token' array. A token can be a keyword, number, quoted string, asterisk, or noise word. This grouping is controlled by a set of state variables as shown in Figure 1. Many errors can be detected by testing each character to see if it is of a type permitted by the current state of the analysis. This testing is done by a set of logical functions which return a **.TRUE.** or **.FALSE.** indicating if a given character is a member of a predefined set of expected characters for a given state.

The tokens are then returned to RDFREE in character form, maintaining the order in which they were analyzed. The order of the tokens is then analyzed for correct grammar, with numeric tokens being converted to double precision and repeat factors converted to integer. Keywords are put in the 'mdfyr' array, numerics are stored in the 'dvalue' array, and strings are stored in the 'svalue' array. String tokens are differentiated from the alphanumeric keywords by a single quote as the first and last characters. Control is returned to the calling routine when the last line of a logical record is read and all tokens have been analyzed.

The logical relationship among the RDFREE modules is given in the calling tree of Figure 2.



1 TOKEN = NEW SET OF SYMBOLS ON THE INPUT LINE

Figure 1. RDFREE State Diagram for Token Generation

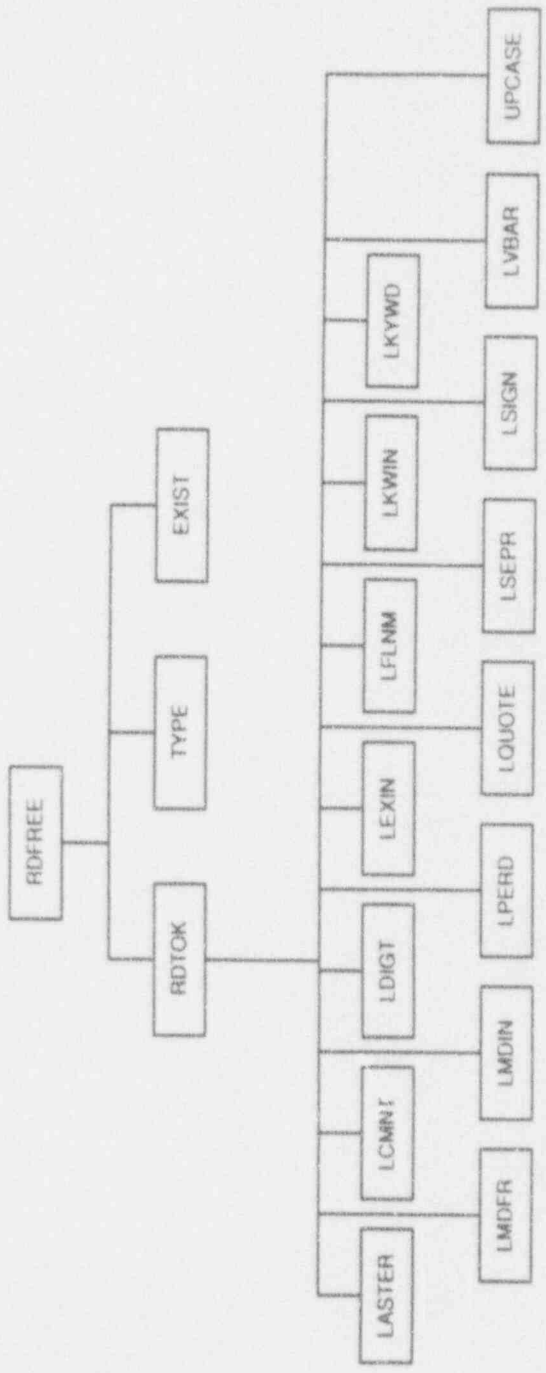


Figure 2. RDFREE Calling Tree

13. CONTACTS

For any problems related to the use of the RDFREE utility or to provide any suggestions for its improvement, contact Dr. Budhi Sagar at (512)522-5252 or Mr. Ron Janetzke at (512)522-3318.

Appendix A

RDFREE RESERVED NAMES

EXTERNAL REFERENCES

SUBPROGRAMS

```
exist           lmdin
laster          lperd
lcmnt           lquote
ldigt           lsepr
lexin           lsign
lflnm           lvbar
lkwin           rdfree
lkywd           rdtok
lmdfr           type
upcase
```

COMMON BLOCKS

```
rdfint
rdfcha
rdfdp
```

LOCAL REFERENCES

PARAMETERS

```
maxmod
maxdvl
maxsvl
maxtok
```

VARIABLES

```
euint
iunit
ounit
mcnt
rderr
sordi
firsti
lasti
mdfyr
svalue
title
token
dvalue
```

Appendix B

DEMO PROGRAM LISTING

```
CCCCCC
C     NAME: demo
C
C     PURPOSE:
C       This program demonstrates the suite of routines that perform free
C       format, keyword oriented data input from a multi-line logical record.
C       This input method can be used for large or small input data files and,
C       with limited functionality, for keyboard input. Multi-line input is
C       allowed with a vertical bar terminating character (|) required in all
C       cases.  Comments can be initiated anywhere with a backslash (\), but
C       continue only to the end of the current line.  A vertical bar in a
C       comment field will not signal an end of logical record function,
C       however comments may appear after a vertical bar.
C
C       Modifiers are allowed for the keywords.  Both the keyword and
C       modifiers may be up to 64 characters but only the first 6 are
C       considered.  The keyword is just like the modifiers in that it must
C       be defined in order to be recognized, but differs in that it is
C       always the first recognizable token in a logical record.  The
C       modifiers may begin anywhere including column one (1) of succeeding
C       lines.
C
C       Only five separator characters are allowed, space, equal, comma,
C       colon, and tab.
C
C       Keywords and modifiers may have from 0 to MAXDVL total data values
C       for a given input logical record. (A logical record is everything
C       from the first letter of the keyword to the terminating character.)
C
C       Two operator characters are allowed, the * and the '.  The * provides
C       a repeat function, where the data value immediately following the
C       * is repeated as indicated by the integer which immediately precedes
C       the *.
C
C       The ' operator indicates that the item following is a string to be
C       associated with the modifier preceding it.  The string must also be
C       terminated with a matching '.  This mechanism does allow for the
C       input of file names for various operating systems.
C
C       Both the keywords and modifiers can be associated with a set of
C       string values or a set of data values.  The string must be less than 65
C       characters, and the data will be returned in a DOUBLE PRECISION array.
C
C       The TITLE keyword is treated as a special case in that the title is
C       returned in a unique string variable of length 80, rather than in the
C       usual data tables used in all other cases.  When the TITLE keyword is
C       used in this way the keyword must occupy columns 1-5.
C
C       REFERENCES:
C              rdfree
C              exist
C
C       CHANGES:
C               1.00     05-03-91       Ron Janetzke
C                        Original version.
CCCCCC
CCCCCC
```

```
C      RDFREE PARAMETERS
C              Use the following parameters to set up and dimension the
C              work space for the RDFREE routines.
C
C              maxdvl = INTEGER maximum number of numeric values allowed.
C              maxmod = INTEGER maximum number of modifiers allowed.
C              maxsvl = INTEGER maximum number of string values allowed.
C              maxtok = INTEGER maximum number of tokens allowed.  The
C                       dimension for 'token' which is the work array for
C                       subroutine RDTOK.  It should be a minimum of
C                       (maxmod + maxdvl + 2*maxsvl + (the number of noise
C                       words in a line)).  Each noise word is stored as
C                       a token in the work array until it is rejected by
C                       later processing.  Some cases may require a larger
C                       value for 'maxtok' in order to hold some transparent
C                       tokens used in more involved data assignments.
CCCCCC
      INTEGER maxdvl
      INTEGER maxmod
      INTEGER maxsvl
      INTEGER maxtok
C
      PARAMETER (maxdvl=30)
      PARAMETER (maxmod=30)
      PARAMETER (maxsvl=30)
      PARAMETER (maxtok=100)
CCCCCC
C      Declare external reference.
CCCCCC
      LOGICAL exist
CCCCCC
C      /RDFINT/
C                  to RDFREE:
C                  eunit = INTEGER FORTRAN error output unit number.  The error
C                          messages sent to this unit should not be ignored.
C                          Any error should be considered fatal for the correct
C                          transfer of data from the input stream into
C                          program variables.
C                  iunit = INTEGER FORTRAN input unit number.
C                  ounit = INTEGER FORTRAN output unit number.
C
C                  from RDFREE:
C                  firsti = INTEGER array pointing to the first element of svalue
C                           or dvalue for corresponding modifier (mdfyr).
C                  lasti  = INTEGER array pointing to the last element of svalue
C                           of dvalue for corresponding modifier (mdfyr).
C                  mcnt   = INTEGER number of returned modifier names.
C                  sordi  = INTEGER array indicating svalue, dvalue, or none for
C                           corresponding modifier (mdfyr).
C                           sordi(m) = 0 if keyword name only.
C                          sordi(m) = 1 if values are stored in the svalue array.
C                          sordi(m) = 2 if values are stored in the dvalue array.
C                  rderr  = INTEGER read error status from RDTOK.
CCCCCC
      INTEGER eunit
      INTEGER iunit
      INTEGER ounit
```

```
C
      INTEGER firsti(maxmod)
      INTEGER lasti(maxmod)
      INTEGER mcnt
      INTEGER rderr
      INTEGER sordi(maxmod)
      COMMON /rdfint/ eunit, iunit, ounit,
     &                mcnt, rderr, sordi, firsti, lasti
CCCCCC
C     /RDFCHA/
C               to RDFREE:
C               token  = CHARACTER*64 work array for token strings.
C
C               from RDFREE:
C               mdfyr  = CHARACTER*6 array of modifier names.
C               svalue = CHARACTER*64 array of string values.
C               title  = CHARACTER*80 array for the title.
CCCCCC
      CHARACTER*6  mdfyr(maxmod)
      CHARACTER*64 svalue(maxsvl)
      CHARACTER*80 title
      CHARACTER*64 token(maxtok)
      COMMON /rdfcha/ mdfyr, svalue, title, token
CCCCCC
C     /RDFDP/
C               from RDFREE:
C               dvalue = double precision array of floating point values.
CCCCCC
      DOUBLE PRECISION dvalue(maxdvl)
      COMMON /rdfdp/ dvalue
CCCCCC
C     i       is a loop control variable.
C     j       is a loop control variable.
C     modptr  is the pointer to the modifier in the 'mdfyr' array.
C     num     is the number of data values associated with the modifier.
CCCCCC
      INTEGER i
      INTEGER j
      INTEGER modptr
      INTEGER num
CCCCCC
C     'lexist' is a flag to indicate the existence of a particular modifier
C     in the input line.
CCCCCC
      LOGICAL lexist
CCCCCC
C     adummy   is a dummy variable to read a blank line.
C     c        is the keyboard input buffer.
CCCCCC
      CHARACTER*1  adummy
      CHARACTER*64 c
CCCCCC
C     'lenmst' is the size of the 'master' keyword array.
C
C     'master' is the master list of all possible keywords to be considered.
C     If any other alphabetic token is found, it is ignored.  It must be
C     defined in upper case, since the token analyzer converts to upper
```

```
C     case before storing in the 'mdfyr' array.
C
C     NOTE: The element LEG'S uses a single qucte as a valid character in a
C           keyword.
CCCCCC
      INTEGER lenmst
      PARAMETER (lenmst=19)
      CHARACTER*6  master(lenmst)
      DATA master /   'TITLE',
     &                'END',
     &                'RUN',
     &                'VELOCI',
     &                'GROUP1',
     &                'GROUP2',
     &                'EXTERN',
     &                'PARAME',
     &                'LEG''S',
     &                'JUCNTI',
     &                'PATH',
     &                'NETWOR',
     &                'ELEMEN',
     &                'HALF-L',
     &                'MIGRAT',
     &                'SOURCE',
     &                'DISCHA',
     &                'DECAY',
     &                'EVERY'/
CCCCCC
C     Start here.
CCCCCC
      title=' '
      eunit = 5
      iunit = 5
      ounit = 5
CCCCCC
C     Give introduction.
CCCCCC
      write(ounit,1001)
 1001 FORMAT(/x,'================================================='/
     &        x,'==                  RDFREE  DEMO              =='/
     &        x,'================================================='//
     &        x,'This is a demonstration program for the RDFREE suite'/
     &        x,'of FORTRAN subroutines.   The user can enter keywords'/
     &        x,'and modifiers from the master list and all may have '/
     &        x,'data values associated with them in a free format '/
     &        x,'style.   A prompt is then given to select one of the'/
     &        x,'keywords or modifiers just entered in the multi-line'/
     &        x,'data record, and the program will display the data'/
     &        x,'values associated with it, if any.'//
     &        x,'             - Press return for more -')
      read (iunit,'(a1)') adummy
      write(ounit,1002)
 1002 FORMAT(
     &        x,'For example, after the master list is displayed,',
     &        x,'input the following two lines:'//
     &        x,'GROUP1 PATH 1,'/
     &        x,'        NETWORK 1 2 3|'//
     &         x,'Then at the query prompt enter > NETWORK'//
     &         x,'In this way you can experiment with different ',
     &         x,'separators, different data value '/
     &         x,'types (i.e. integer, floating point, and strings), ',
     &         x,'and the repeat data value'/
     &         x,'function.'//
     &         x,'NOTE: Under normal conditions the keyword TITLE must',
     &         x,' occupy columns 1-5.'/
     &         7x,'The title string may then follow it on one line.'/)
CCCCCC
C     Analyze input lines continuously until the END command is given.
CCCCCC
  100 CONTINUE
      write (ounit,2001)
 2001 FORMAT (x,' The MASTER LIST is'/
     & x,'____________________________________________________________')
      write (ounit,2002) master
 2002 FORMAT (2(x,'|',10(x,a6)/))
      write (ounit,2003)
 2003 FORMAT (
     & x,'|__________________________________________________________'/
     & x,'Enter data line(s) below in free form (don't forget the |)'/
     & x,'All input will be echoed after the RETURN. [CTRL-Z to quit].')
CCCCCC
C     Call 'rdfree' input routine which will automatically read the first
C     record of the input data file.
CCCCCC
      rderr = 0
      call rdfree (lenmst, master)
      if (rderr .lt. 0 ) go to 999
      if (rderr .gt. 0) then
         write (ounit,3001)
 3001 FORMAT (x,'All RDFREE errors are fatal, because they are usually'/
     &         x,'the result of an ambiguous sequence of characters.'/)
         go to 100
      endif
CCCCCC
C     Test 'mcnt' > 0.
CCCCCC
      if (mcnt .lt. 1) then
         write (ounit,*) 'No valid keywords found.'
         go to 100
      end if
CCCCCC
C     Display TITLE if known.
CCCCCC
  300 continue
      if (title .ne. ' ') write (ounit,3002) title(1:72)
 3002 FORMAT (/x,'TITLE=',a)
CCCCCC
C     Prompt for query word.
CCCCCC
      write (ounit,3003)
 3003 FORMAT (7x,'Enter query item in UPPERCASE (CTRL-Z to quit)')
CCCCCC
C     Allow the user to query the modifier list at random to see if a
C     particular modifier was present, and if so, what data values were
```

```
C     associated with it.
CCCCCC
  200 CONTINUE
      read (iunit, '(a)',end=888) c
      lexist = exist (c, mcnt, mdfyr, modptr)
CCCCCC
C     Display modifier data values if any.
CCCCCC
      if (.not. lexist) then
         write (ounit,*)
     &   'None of the keywords for this line match your query.'
C
      else if (sordi(modptr) .eq. 0) then
              write (ounit,4001) mdfyr(modptr),modptr
 4001 FORMAT  (' The item''s id is ',a6,', and is in position',i3,
     &         '.  It has no data values.')
C
      else if (sordi(modptr) .eq. 1) then
              num = lasti(modptr)-firsti(modptr) + 1
              write (ounit,4002) mdfyr(modptr),modptr,num
 4002 FORMAT  (' The item''s id is ',a6,', and is in position',i3,
     &         '.  It has',i3,' string data values.')
              DO 249 j=firsti(modptr), lasti(MODPTR)
                 write (ounit,'(x,a)') svalue(j)
  249         CONTINUE
C
      else if (sordi(modptr) .eq. 2) then
              num = lasti(modptr)-firsti(modptr) + 1
              write (ounit,4003) mdfyr(modptr),modptr,num
 4003 FORMAT  (' The item''s id is ',a6,', and is in position',i3,
     &         '.  It has',i3,' numeric data values.')
              do 259 j=firsti(modptr), lasti(modptr)
                 write (ounit, '(6x,1pd24.16)') dvalue(j)
  259         CONTINUE
      end if
C          Check for the END command keyword.
      if (mdfyr(1) .eq. 'END') go to 999
      GO TO 300
C
  888 continue
      if (rderr .ne. 0) go to 999
      GO TO 100

  999 continue
      stop
      end
```

Appendix C

RDFREE UTILITY LISTING

```
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
        subroutine rdfree (lenmst, master)
cccccc
c       NAME: rdfree
c
c       PURPOSE: This module is part of a suite of routines used to analyze
c                a free form input stream in order to generate a consistent
c                array of modifier names and associated data values.
c
c       ARGUMENTS:
c               input:
c               lenmst = INTEGER maximum number of master keywords.
c               master = CHARACTER*6 array of master keywords.
c
c       REFERENCES:
c               exist
c               rdtok
c               type
c               rdfcom
c
c       CHANGES:
c               1.00    08-28-90        Ron Janetzke
c                       Original text.
c               1.01    09-21-90        Ron Janetzke
c                       RDERR added to the argument lists of RDFREE and
c                       RDTOK.
c               1.02    02-12-91        Ron Janetzke
c                       Reflect input on iunit to ounit.
c                       Stop instead of 'go to 999' on all errors.
c               2.00    04-26-91        Ron Janetzke
c                       Pass parameters by common rather than argument list.
cccccc
        INTEGER lenmst
        CHARACTER*6  master(lenmst)
cccccc
c     RDFREE PARAMETERS
c             Use the following parameters to set up and dimension the
c             work space for the RDFREE routines.
c
c             maxdvl = INTEGER maximum number of numeric values allowed.
c             maxmod = INTEGER maximum number of modifiers allowed.
c             maxsvl = INTEGER maximum number of string values allowed.
c             maxtok = INTEGER maximum number of tokens allowed.  The
c                      dimension for 'token' which is the work array for
c                      subroutine RDTOK.  It should be a minimum of
c                      (maxmod + maxdvl + 2*maxsvl + (the number of noise
c                      words in a line)).  Each noise word is stored as
c                      a token in the work array until it is rejected by
c                      later processing.  Some cases may require a larger
c                      value for 'maxtok' in order to hold some transparent
c                      tokens used in more involved data assignments.
cccccc
      INTEGER maxdvl
      INTEGER maxmod
      INTEGER maxsvl
      INTEGER maxtok
```

```
C
      PARAMETER (maxdvl=30)
      PARAMETER (maxmod=30)
      PARAMETER (maxsvl=30)
      PARAMETER (maxtok=100)
CCCCCC
C     Declare external reference.
CCCCCC
      LOGICAL exist
CCCCCC
C     /RDFINT/
C                to RDFREE:
C                eunit = INTEGER FORTRAN error output unit number.  The error
C                        messages sent to this unit should not be ignored.
C                        Any error should be considered fatal for the correct
C                        transfer of data from the input stream into
C                        program variables.
C                iunit = INTEGER FORTRAN input unit number.
C                ounit = INTEGER FORTRAN output unit number.
C
C                from RDFREE:
C                firsti = INTEGER array pointing to the first element of svalue
C                         or dvalue for corresponding modifier (mdfyr).
C                lasti  = INTEGER array pointing to the last element of svalue
C                         of dvalue for corresponding modifier (mdfyr).
C                mcnt   = INTEGER number of returned modifier names.
C                sordi  = INTEGER array indicating svalue, dvalue, or none for
C                         corresponding modifier (mdfyr).
C                         sordi(m) = 0 if keyword name only.
C                        sordi(m) = 1 if values are stored in the svalue array.
C                        sordi(m) = 2 if values are stored in the dvalue array.
C                rderr  = INTEGER read error status from RDTOK.
CCCCCC
      INTEGER eunit
      INTEGER iunit
      INTEGER ounit
C
      INTEGER firsti(maxmod)
      INTEGER lasti(maxmod)
      INTEGER mcnt
      INTEGER rderr
      INTEGER sordi(maxmod)
      COMMON /rdfint/ eunit, iunit, ounit,
     &                mcnt, rderr, sordi, firsti, lasti
CCCCCC
C     /RDFCHA/
C                to RDFREE:
C                token  = CHARACTER*64 work array for token strings.
C
C                from RDFREE:
C                mdfyr  = CHARACTER*6 array of modifier names.
C                svalue = CHARACTER*64 array of string values.
C                title  = CHARACTER*80 array for the title.
CCCCCC
      CHARACTER*6  mdfyr(maxmod)
      CHARACTER*64 svalue(maxsvl)
      CHARACTER*80 title
      CHARACTER*64 token(maxtok)
      COMMON /rdfcha/ mdfyr, svalue, title, token
CCCCCC
C     /RDFDP/
C               from RDFREE:
C               dvalue = double precision array of floating point values.
CCCCCC
      DOUBLE PRECISION dvalue(maxdvl)
      COMMON /rdfdp/ dvalue
CCCCCC
C       dcnt   = data value counter
C       eunit  = error I/O unit number
C       i      = do loop control variable
C       iunit  = input unit number
C       j      = do loop control variable
C       keyptr = element number of keyword in master keyword array
C       lastin = the last integer which wa stored in the data array
C       mcnt   = modifier counter
C       mod1   = first data element associated with the current modifier
C       mod2   = last  data element associated with the current modifier
C       ounit  = output I/O unit number
C       rderr  = read error number
C       repcnt = repeat counter for the '*' operator
C       repptr = pointer to the data element which is to be repeated. The
C                value to be repeated has already been stored in one of
C                the 2 data arrays when 'repptr' is referenced.
C       scnt   = string value counter
C       tokcnt = token counter
CCCCCC
        INTEGER dcnt
        INTEGER i
        INTEGER j
        INTEGER keyptr
        INTEGER lastin
        INTEGER mod1
        INTEGER mod2
        INTEGER repcnt
        INTEGER repptr
        INTEGER scnt
        INTEGER tokcnt
CCCCCC
C       repval          = temporary storage for the number to be repeated.
CCCCCC
        DOUBLE PRECISION repval
CCCCCC
C       lastc            = previous token
C       c               = current token
C       ctype           = current token type
C       mdtyp           = current modifier type
C       type            = external function name
C       ASTER           = string parameter
C       STRING          = string parameter
C       UNKNO           = string parameter
C       ALPHA           = string parameter
C       NUMER           = string parameter
C       INTEG           = string parameter
C       FLOAT           = string parameter
```

```
cccccc
        CHARACTER*64 lastc
        CHARACTER*64 c
        CHARACTER*6  ctype
        CHARACTER*6  mdtyp
        CHARACTER*6  type
        CHARACTER*6  ASTER
        CHARACTER*6  STRING
        CHARACTER*6  UNKNO
        CHARACTER*6  ALPHA
        CHARACTER*6  NUMER
        CHARACTER*6  INTEG
        CHARACTER*6  FLOAT
        CHARACTER*6  NULL
cccccc
c       true   = logical parameter
c       false  = logical parameter
c       repeat = flag to indicate when a repeated data value is to be
c                processed.
cccccc
        LOGICAL true
        LOGICAL false
        LOGICAL repeat
cccccc
        parameter (ASTER = '*')
        parameter (STRING= 'string')
        parameter (UNKNO = 'unkno')
        parameter (ALPHA = 'alpha')
        parameter (NUMER = 'numer')
        parameter (INTEG = 'integ')
        parameter (FLOAT = 'float')

        parameter (true = .TRUE.)
        parameter (false = .FALSE.)
cccccc
c       start here.
cccccc

c       Read the first set of records making up the first data line.

        call rdtok (iunit, ounit, eunit, maxtok, token, tokcnt,title,
     &              rderr)

c       Initialize control variables.

        repeat = false
        repval = 0
        mdtyp = NULL
        lastc = NULL

        do 99 i=1,maxmod
           sordi(i) = 0
           firsti(i) = 0
           lasti(i) = 0
   99   continue

        mcnt   = 0
```

```
      dcnt   = 0
      scnt   = 0

C     Analyze each token.

      do 199 i=1,tokcnt
         c = token(i)

C        Find token type.

         ctype = type(c)

         if (ctype .eq. ALPHA) then
            if (mod1 .gt. 0) then

C           The previous modifier was in the process of counting its
C           associated data values.  Complete bookkeeping on the data
C           value pointers of the previous modifier and then proceed.

            firsti(mcnt) = mod1
            lasti(mcnt) = mod2
         end if

C        The token is alphabetic and to be used as a keyword/modifier.
C        But first check for sufficient modifier name array space.

         if (mcnt .eq. maxmod) then
            write (eunit,*) 'RDFREE argument MAXMOD exceeded for ',
     &                      'keyword storage.'
            stop
         end if

C        Do bookkeeping for adding current token ('c') to modifier
C        names only if it passes the test against the master keyword list.

         if (exist(c,lenmst,master,keyptr)) then

C        token is in the master list, so add it to the
C        modifier array.

         mcnt = mcnt + 1
         mdfyr(mcnt) = c

C        Initialize bookkeeping data associated with this pointer.

         sordi(mcnt) = 0
         mdtyp = NULL
         mod1 = 0
         mod2 = 0
      else

C        Do not update the 'lastc' variable and select the
C        next token.

         go to 199
      end if
      else if (ctype .eq. STRING) then
cccccc
c     If the modifier already has a numeric assigned to it
c     then send an error message and stop.
cccccc
         if (mdtyp .eq. NUMER) then
            write (eunit,*) 'String type appears in a ',
     &                      'numeric data type field for ',
     &                      'modifier ',mdfyr(mcnt)
            stop
         end if

c        This token is a string value.  But first check
c        for sufficient string array space.

         if (scnt .eq. maxsvl) then
            write (eunit, *) 'RDFREE argument MAXSVL ',
     &                       'exceeded for',
     &                       ' string values.'
            stop
         end if

c        Do bookkeeping for the string value.  Start at string
c        index 2 in order to skip the leading single quote which
c        was saved as part of the token by RDTOK.
         scnt = scnt + 1
         svalue(scnt) = c(2:)
         mdtyp = STRING

         if (mod1 .eq. 0) then

c           Do bookkeeping for the string value array for the first
c           string of the current keyword/modifier.

            mod1 = scnt
            mod2 = scnt
            sordi(mcnt) = 1
         else

c           Do bookkeeping for the second and following string
c           values of the current keyword/modifier.

            mod2 = scnt
         end if

         if (repeat .and. repcnt .gt. 1) then

c           Process a repeated value.  But first check for sufficient
c           storage in the string array.

            if (scnt+repcnt .gt. MAXSVL) then
               write (eunit,*) 'A repeat integer of ', repcnt,
     &                         'caused RDFREE argument MAXSVL ',
     &                         'for the string value array ',
     &                         'to overflow.'
               stop
            else
```

```
              repptr = scnt

C             No need to start at j=1 because 'c' has been stored
C             in 'svalue' above.
              do 149 j=2,repcnt
                scnt = scnt + 1
                svalue(scnt) = svalue(repptr)
                mod2 = scnt

  149         continue

C             repeat is now complete, so clear control variables.

              repeat = false
              repcnt = 0
           end if
        end if

     else if (ctype .eq. INTEG) then
CCCCCC
C          If the current token type is INTEG there are several states
C          possible.  It may be any of the following:
C          - a numeric item for a numeric keyword
C          - the repeat factor for a numeric item
C          - the repeat factor for a string item
C          - a numeric item to be repeated
CCCCCC
           if (mdtyp .eq. NULL .and. .not. repeat) then
              if (i+1                 .le. tokcnt .and.
     &            token(i+1)          .eq. ASTER  .and.
     &            i+2                 .le. tokcnt .and.
     &           (type(token(i+2))    .eq. INTEG  .or.
     &            type(token(i+2))    .eq. FLOAT  .or.
     &            type(token(i+2))    .eq. STRING)) then
                 read (c,701) lastin
  701            format (bn, i24)
              else
                 if (dcnt .eq. maxdvl) then
                    write (eunit,*) 'RDFREE argument MAXDVL ',
     &                              'exceeded for data values.'
                    stop
                  end if
               mdtyp = NUMER
               dcnt = dcnt + 1
               read (c,701) lastin
               dvalue(dcnt) = lastin
               mod1 = dcnt
               mod2 = dcnt
               sordi(mcnt) = 2

               end if
            else if (mdtyp .eq. NULL .and. repeat) then

C             Process a repeated value.  But first check for sufficient
C             storage in the data array.

              if (dcnt+repcnt .gt. MAXDVL) then
```

```
                 write (eunit, *) 'RDFREE argument MAXDVL ',
     &                            'exceeded for',
     &                            ' data values.'
                 stop
              else
                 mdtyp = NUMER
                 dcnt = dcnt + 1
                 read (c,701) lastin
                 dvalue(dcnt) = lastin
                 mod1 = dcnt
                 mod2 = dcnt
                 sordi(mcnt) = 2
                 repval = dvalue(dcnt)
                 if (repcnt .gt. 1) then
C                   No need to start at j=1 because 'c' has been stored
C                   in 'dvalu' above.
                    do 139 j=2,repcnt
                       dcnt = dcnt + 1
                       dvalue(dcnt) = repval
                       mod2 = dcnt
  139                continue
                  end if
                  repeat = false
                  repcnt = 0
                  lastin = 0
               end if

            else if (mdtyp .eq. NUMER .and. .not. repeat) then
                if (i+1 .le. tokcnt .and.
     &             token(i+1) .eq. ASTER) then
                   if (i+2 .le. tokcnt .and.
     &                type(token(i+2)) .eq. INTEG .or.
     &                type(token(i+2)) .eq. FLOAT) then
                      read (c,701) lastin
                   else
                      write (eunit,*) 'Mixed mode ',
     &                                'assignment while processing ',
     &                                'a field for ',
     &                                 mdfyr(mcnt)
                      stop
                   end if
                else
                   if (dcnt .eq. maxdvl) then
                     write (eunit,*) 'RDFREE argument MAXDVL ',
     &                               'exceeded for data values.'
                     stop
                   end if
                   dcnt = dcnt + 1
                   read (c,701) lastin
                   dvalue(dcnt) = lastin
                   mod2 = dcnt
                end if

             else if (mdtyp .eq. NUMER .and. repeat) then

C                 Process a repeated value.  But first check for sufficient
C                 storage in the data array.
```

```
                 if (dcnt+repcnt .gt. MAXDVL) then
                    write (eunit, *) 'RDFREE argument MAXDVL ',
     &                               'exceeded for',
     &                               ' data values.'
                    stop
                 else
                    dcnt = dcnt + 1
                    read (c,701) lastin
                    dvalue(dcnt) = lastin
                    mod2 = dcnt
                    repval = dvalue(dcnt)
                    if (repcnt .gt. 1) then
C                   No need to start at j=1 because 'c' has been stored
C                   in 'dvalu' above.
                       do 141 j=2,repcnt
                         dcnt = dcnt + 1
                         dvalue(dcnt) = repval
                         mod2 = dcnt
  141                  continue
                    end if
                    repeat = false
                    repcnt = 0
                    lastin = 0
                 end if

              else if (mdtyp .eq. STRING) then
                 if (i+1 .le. tokcnt .and.
     &               token(i+1) .eq. ASTER) then
                   if (i+2 .le. tokcnt .and.
     &                  type(token(i+2)) .eq. STRING) then
                      read (c,701) lastin
                   else
                      write (eunit,*) 'Mixed mode or incomplete ',
     &                                'assignment while processing',
     &                                'a repeated field for ',
     &                                 mdfyr(mcnt)
                      stop
                   end if
                 else
                      write (eunit,*) 'Mixed mode ',
     &                                'assignment while processing ',
     &                                'a field for ',
     &                                 mdfyr(mcnt)
                   stop
                 end if
             end if

          else if (ctype .eq. FLOAT) then
             if (mdtyp .ne. STRING) then
C                This token is a floating point numeric.  But first check
C                for sufficient data array space.

                 mdtyp = NUMER
                 if (dcnt .eq. maxdvl) then
                    write (eunit, *) 'RDFREE argument MAXDVL ',
     &                               'exceeded for',
     &                               ' data values.'
```

```
                     stop
                  end if

C                 Do bookkeeping for the numeric data value.

                  dcnt = dcnt + 1
                  read (c,702) dvalue(dcnt)
  702             format (bn,d24.16)

               if (mod1 .eq. 0) then

C                 Do bookkeeping for the data value array for the first
C                 number of the current keyword/modifier.

                  mod1 = dcnt
                  mod2 = dcnt
                  sordi(mcnt) = 2
               else

C                 Do bookkeeping for the second and following data values
C                 of the current keyword/modifier.

                  mod2 = dcnt
               end if

               if (repeat .and. repcnt .gt. 1) then

C                 Process a repeated value.  But first check for sufficient
C                 storage in the data array.

                  if (dcnt+repcnt .gt. MAXDVL) then
                     write (eunit, *) 'RDFREE argument MAXDVL ',
     &                                'exceeded for',
     &                                ' data values.'
                     stop
                  else
                     repval = dvalue(dcnt)

C                    No need to start at j=1 because 'c' has been stored
C                    in 'svalue' above.
                     do 159 j=2,repcnt
                       dcnt = dcnt + 1
                       dvalue(dcnt) = repval
                       mod2 = dcnt
  159                continue
                     repeat = false
                     repcnt = 0
                  end i
               end if
            else
               write (eunit,*) 'Attempt to assign a numeric ',
     &                         'value to a string type keyword.'
               stop
            end if

         else if (c .eq. ASTER) then
```

```
C                This token is an asterisk.  But first check the last token
C                for type = INTEGER.

                 if (type(lastc) .ne. INTEG) then
                    write (eunit,*) 'Error non-INTEGER repeat factor.'
                    stop
                 else
C                   Set up controls for processing a repeated data value.

                    repcnt = lastin
                    repeat = true
                 end if
           end if
           lastc = c
  199 continue

        if (mod1 .gt. 0) then

C          Both pointers have been used for bookkeeping, so return them.

           firsti(mcnt) = mod1
           lasti(mcnt) = mod2
        end if

  999   continue
        return
        end
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
        function type (C)
CCCCCC
C       NAME: type
C
C       PURPOSE: This module is part of a suite of routines used to analyze
C                a free form input stream in order to generate a consistent
C                array of modifier names and associated data values.  It is
C                used by RDFREE to classify a token as alphabetic, numeric,
C                string, or an * operator.
C
C       ARGUMENTS:
C                input:
C                c = CHARACTER*24 token string.
C
C                output:
C                type = CHARACTER*6 token classification.
C
C       REFERENCES:
C                none
C
C       CHANGES:
C                1.00    08-28-90        Ron Janetzke
C                        Original text.
C                1.01    09-24-90        Ron Janetzke
C                        Delete equal type and add string type.
CCCCCC

C       temp  = scratch area
C       type  = function name
```

```
C       c     = function argument
C       ASTER = string parameter
C       UNKNO = string parameter
C       ALPHA = string parameter
C       NUMER = string parameter
C       INTEG = string parameter
C       FLOAT = string parameter
C       OPER  = string parameter
C       STRING = string parameter

        CHARACTER*6  temp
        CHARACTER*6  type
        CHARACTER*24 c
        CHARACTER*6  ASTER
        CHARACTER*6  UNKNO
        CHARACTER*6  ALPHA
        CHARACTER*6  NUMER
        CHARACTER*6  INTEG
        CHARACTER*6  FLOAT
        CHARACTER*6  OPER
        CHARACTER*6  STRING


        parameter (ASTER = '*')
        parameter (UNKNO = 'unkno')
        parameter (ALPHA = 'alpha')
        parameter (NUMER = 'numer')
        parameter (INTEG = 'integ')
        parameter (FLOAT = 'float')
        parameter (OPER  = 'oper')
        parameter (STRING = 'string')

CCCCCC
C       Start here.
CCCCCC
        temp = UNKNO

        if (len(c) .gt. 0) then

C          Assign temp the value of OPER, NUMER, STRING, or ALPHA.

           if (c .eq. ASTER) then
                temp = OPER
           else if (c(1:1) .eq. '+' .or.
     &               c(1:1) .eq. '-' .or.
     &               c(1:1) .eq. '.' .or.
     &              (c(1:1) .ge. '0' .and.
     &               c(1:1) .le. '9'      )) then
                temp = NUMER
           else if (c(1:1) .eq. '''') then
                temp = STRING
           else
                temp = ALPHA
           end if
           if (temp .eq. NUMER) then

C          Refine the token classification to be either INTEG or FLOAT.
```

```
            if (index(c,'.') .ne. 0 .or.
     &         index(c,'E') .ne. 0 .or.
     &         index(c,'D') .ne. 0) then
               temp = FLOAT
            else
               temp = INTEG
            end if
         end if

         type = temp
      end if

      return
      end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine rdtok (iunit, ounit, eunit, maxtok, token, toknum,
     &                  title, rderr)
cccccc
c     NAME: rdtok
c
c     PURPOSE:
c             RDTOK reads records from the input unit.  Each record is
c             analyzed character by character in order to group the
c             characters into token strings.  Comments beginning with \
c             are ignored.  Reading of the records continues until an
c             end of line terminator is found (|).
c
c     METHOD:
c             A series of logical variables are used to hold the current
c             state of the token generator.  These variables determine
c             which set of characters are permitted for a given token
c             type, which set are separators , and which set may initiate
c             the processing of another token.  A keyword token is
c             different than a modifier token only in the sense that it
c             appears first in the input stream.
c
c     ARGUMENTS:
c             input:
c             iunit = INTEGER FORTRAN input unit number.
c             ounit = INTEGER FORTRAN output unit number.
c             eunit = INTEGER FORTRAN error output unit number.
c             maxtok= INTEGER maximum number of tokens allowed.
c
c             output:
c             rderr = INTEGER error status of read statement.
c             token = CHARACTER*(*) array of token strings.
c             toknum = INTEGER number of tokens returned.
c             title  = CHARACTER*80 title string stored separate from
c                      tokens.
c
c     REFERENCES:
c             lastr
c             lcmnt
c             ldigt
c             lexin
c             lflnm
c             lkwin
```

```
C               lkywd
C               lmdfr
C               lmdin
C               lnmbr
C               lnmin
C               lperd
C               lquote
C               lsepr
C               lsign
C               lvbar
C               upcase
C
C       CHANGES:
C               1.00    08-23-90        Ron Janetzke
C                       Original text.
C               1.01    09-20-90        Ron Janetzke
C                       RDERR is set to -1 for end of file and added to
C                       argument list.
C               1.02    09-24-90        Ron Janetzke
C                       Insert '.' in floating point number if missing before
C                       exponent.  Use ' for string delimiter, and allow
C                       multiple string data values.  Move '=' to separator
C                       list.
C               1.03    10-19-90        Ron Janetzke
C                       Add RDERR=1, and RDERR=2.
C                       Add exponent specific code. (lfexp)
C               1.04    02-05-91        Ron Janetzke
C                       Function LSEPR changed to add TAB as separator.
CCCCCC

C        RECLEN = the numbers of columns used for input should match this
C                 specification of record length.
C        title  = special variable to hole a title string, if one is found
C                 in the input records.
C        rec    = input record buffer.
C
C        blank  = string parameter
C        upcase = function
C        null   = null string character
C        lastc  = previous character
C        c      = current character
C        token  = output argument
C        temp5  = scratch area for the first 5 characters of 'rec'.
C
C        eunit  = error I/O unit
C        i      = loop indes
C        iunit  = input unit
C        maxtok = maximum number of tokens
C        mxtlen = maximum token length
C        ounit  = output unit
C        pcount = count of periods in the current numeric token (1 max).
C        rderr  = error number as a result of the read.
C        toklen = length of the current token
C        toknum = number of the current token
C        xcount = count of the number of initial exponent characters [E,D]
C                 (1 max).
C
```

```
C         lfastr = look for asterisk
C         lfcmnt = look for comment character
C         lfdigt = look for digit
C         lfexin = look for exponent initial character
C         lfexp  = look for exponent
C         lfflnm = look for file name or any quoted string
C         lfkwin = look for initial keyword character
C         lfkywd = look for keyword character
C         lfmdfr = look for modifier character
C         lfmdin = look for initial modifier character
C         lfnmbr = look for number
C         lfnmin = look for initial number
C         lfperd = look for period
C         lfqtin = look for initial quote
C         lfqtfn = look for final quote
C         lfsepr = look for separator character
C         lfsign = look for + or - sign
C         lfvbar = look for vertical bar
C
C         lkywdf = keyword found flag
C
C         lastr  = function
C         lcmnt  = function
C         ldigt  = function
C         lexin  = function
C         lflnm  = function
C         lkwin  = function
C         lkywd  = function
C         lmdfr  = function
C         lmdin  = function
C         lnmbr  = function
C         lnmin  = function
C         lperd  = function
C         lquote  = function
C         lsepr  = function
C         lsign  = function
C         lvbar  = function
C
C         false  = parameter
C         true   = parameter

        INTEGER RECLEN
        parameter (RECLEN = 80)
        CHARACTER*80 title
        CHARACTER*80 rec

        CHARACTER*1  blank
        CHARACTER*1  upcase
        CHARACTER*1  null
        CHARACTER*1  lastc
        CHARACTER*1  c
        CHARACTER*(*) token(*)
        CHARACTER*5  temp5

        INTEGER eunit
        INTEGER i
        INTEGER iunit
        INTEGER maxtok
        INTEGER mxtlen
        INTEGER ounit
        INTEGER pcount
        INTEGER rderr
        INTEGER toklen
        INTEGER toknum

        LOGICAL lfastr
        LOGICAL lfcmnt
        LOGICAL lfdigt
        LOGICAL lfexin
        LOGICAL lfexp
        LOGICAL lfflnm
        LOGICAL lfkwin
        LOGICAL lfkywd
        LOGICAL lfmdfr
        LOGICAL lfmdin
        LOGICAL lfnmbr
        LOGICAL lfnmin
        LOGICAL lfperd
        LOGICAL lfqtin
        LOGICAL lfqtfn
        LOGICAL lfsepr
        LOGICAL lfsign
        LOGICAL lfvbar

        LOGICAL lkywdf

        LOGICAL lastr
        LOGICAL lcmnt
        LOGICAL ldigt
        LOGICAL lexin
        LOGICAL lflnm
        LOGICAL lkwin
        LOGICAL lkywd
        LOGICAL lmdfr
        LOGICAL lmdin
        LOGICAL lnmbr
        LOGICAL lnmin
        LOGICAL lperd
        LOGICAL lquote
        LOGICAL lsepr
        LOGICAL lsign
        LOGICAL lvbar

        LOGICAL false
        LOGICAL true

        parameter (false=.false.)
        parameter (true=.true.)
        parameter (blank = ' ')

CCCCCC
C       Start here.
C       Initialize all state variables.
CCCCCC
```

```
        mxtlen = len (token(1))
        null   = char(0)
        lfastr = false
        lfcmnt = true
        lfdigt = false
        lfexin = false
        lfexp  = false
        lfflnm = false
        lfkwin = true
        lfkywd = false
        lfmdfr = false
        lfmdin = false
        lfnmbr = false
        lfnmin = false
        lfperd = false
        lfqtin = false
        lfqtfn = false
        lfsepr = true
        lfsign = false
        lfvbar = true

        pcount = 0

        toklen = 0
        toknum = 0
        lastc = blank

        do 99 i = 1,maxtok
           token(i) = ' '
   99   continue

  100   continue
        read  (iunit, '(a)', end=990) rec
        write (ounit, '(x,a)') rec
cccccc
c       Look for title.
cccccc
        temp5 = rec
        do 140 i=1,5
           temp5(i:i) = upcase(temp5(i:i))
  140   continue
        if (temp5 .eq. 'TITLE') then
cccccc
c          Title is stored outside of the token array in a separate variable.
c          'temp5' will be returned as a keyword in the token array.
cccccc
           title = rec(7:80)
           toknum = 1
           toklen = min(5,mxtlen)
           token(toknum) = temp5
           go to 999
        end if
cccccc
c       Analyze each character in 'rec'.
cccccc
        do 200 i = 1,RECLEN
```

```
C          Only convert to uppercase when not analyzing a quoted string.

           if (.not. lfflnm) then
              c = upcase(rec(i:i))
           else
              c = rec(i:i)
           end if
CCCCCC
C          Look for end of line vertical bar (¦).
CCCCCC
           if (lfvbar .and. lvbar(c)) then
C                write (*,*) c,'lfvbar'

                 go to 999
           end if

CCCCCC
C          Look for comment initial back-slash (\).
CCCCCC
           if (lfcmnt .and. lcmnt(c)) then
C                write (*,*) c,'lfcmnt'
C             Terminate the current token if needed.
              if (.not. lsepr(lastc)) then
                 toknum = toknum + 1
                 if (toknum .gt. maxtok) then
                    write (eunit,*) 'ERROR: number of tokens exceeds '
     &                              ,'MAXTOK'
                    go to 999
                 end if
                 toklen = 0
              end if

                 pcount = 0
C                lfastr =
                 lfcmnt = true
                 lfdigt = true
                 lfexin = false
                 lfexp  = false
                 lfflnm = false
                 lfkwin = false
                 lfkywd = false
                 lfmdfr = false
                 lfmdin = true
                 lfnmbr = false
                 lfnmin = true
                 lfperd = true
                 lfqtin = true
                 lfqtfn = false
                 lfsepr = true
                 lfsign = true
                 lfvbar = true

              go to 100
           end if

CCCCCC
C          Look for keyword initial letter.
```

```
cccccc
            if (lfkwin .and. lkwin(c)) then
c                  write (*,*) c,'lfkwin'
                   if (toklen .eq. 0) then
                      toknum = toknum + 1
                      if (toknum .gt. maxtok) then
                         write (eunit,*) 'ERROR: number of tokens exceeds '
     &                                   ,'MAXTOK'
                         go to 999
                      end if
                   end if
                   toklen = min(mxtlen,toklen +1)
                   token (toknum) (toklen:toklen) = c

cccccc
c                  Logical flag indicates keyword has been found.
cccccc
                   lkywdf = true

c                  Set analyzer state variables.

                   lfastr = false
                   lfcmnt = true
                   lfdigt = true
                   lfexin = false
                   lfexp  = false
                   lfflnm = false
                   lfkwin = false
                   lfkywd = true
                   lfmdfr = false
                   lfmdin = false
                   lfnmbr = false
                   lfnmin = false
                   lfperd = false
                   lfqtin = false
                   lfqtfn = false
                   lfsepr = true
                   lfsign = false
                   lfvbar = true

                   go to 199
            end if
cccccc
c           Look for keyword secondary characters.
cccccc
            if (lfkywd .and. lkywd(c)) then
c                  write (*,*) c,'lfkywd'
                   toklen = min(mxtlen,toklen +1)
                   token (toknum) (toklen:toklen) = c

c                  Set analyzer state variables.

                   lfastr = false
                   lfcmnt = true
                   lfdigt = true
                   lfexin = false
                   lfexp  = false
```

```
                  lfflnm = false
                  lfkwin = false
                  lfkywd = true
                  lfmdfr = false
                  lfmdin = false
                  lfnmbr = false
                  lfnmin = false
                  lfperd = false
                  lfqtin = false
                  lfqtfn = false
                  lfsepr = true
                  lfsign = false
                  lfvbar = true

                  go to 199
            end if

cccccc
c           Look for modifier initial letter.
cccccc
            if (lfmdin .and. lmdin(c)) then
c                 write (*,*) c,'lfmdin'
                  if (toklen .eq. 0)  then
                     toknum = toknum + 1
                     if (toknum .gt. maxtok) then
                        write (eunit,*) 'ERROR: number of tokens exceeds '
     &                                  ,'MAXTOK'
                        go to 999
                     end if
                  end if
                  toklen = min(mxtlen,toklen [illegible])
                  token (toknum) (toklen:toklen) = c

                  lfastr = false
                  lfcmnt = true
                  lfdigt = true
                  lfexin = false
                  lfexp  = false
                  lfflnm = false
                  lfkwin = false
                  lfkywd = false
                  lfmdfr = true
                  lfmdin = false
                  lfnmbr = false
                  lfnmin = false
                  lfperd = false
                  lfqtin = false
                  lfqtfn = false
                  lfsepr = true
                  lfsign = false
                  lfvbar = true

                  go to 199
            end if
cccccc
c           Look for modifier secondary characters.
cccccc
```

```
            if (lfmdfr .and. lmdfr(c)) then
C                 write (*,*) c,'lfmdfr'
                  toklen = min(mxtlen,toklen +1)
                  token (toknum) (toklen:toklen) = c

                  lfastr = false
                  lfcmnt = true
                  lfdigt = true
                  lfexin = false
                  lfexp  = false
                  lfflnm = false
                  lfkwin = false
                  lfkywd = false
                  lfmdfr = true
                  lfmdin = false
                  lfnmbr = false
                  lfnmin = false
                  lfperd = false
                  lfqtin = false
                  lfqtfn = false
                  lfsepr = true
                  lfsign = false
                  lfvbar = true

                  go to 199
            end if

CCCCCC
C           Look for number initial sign.
CCCCCC
            if (lfnmin .and. lfsign .and. lsign(c)) then
C                 write (*,*) c,'lfnmin'
                  if (toklen .eq. 0)  then
                     toknum = toknum + 1
                     if (toknum .gt. maxtok) then
                        write (eunit,*) 'ERROR: number of tokens exceeds '
     &                                  ,'MAXTOK'
                        go to 999
                     end if
                  end if
                  toklen = min(mxtlen,toklen +1)
                  token (toknum) (toklen:toklen) = c

                  lfastr = false
                  lfcmnt = true
                  lfdigt = true
                  lfexin = false
                  lfexp  = false
                  lfflnm = false
                  lfkwin = false
                  lfkywd = false
                  lfmdfr = false
                  lfmdin = false
                  lfnmbr = true
                  lfnmin = false
                  lfperd = true
                  lfqtin = false
```

```
                lfqtfn = false
                lfsepr = true
                lfsign = false
                lfvbar = true

                go to 199
          end if

cccccc
c         Look for number initial digit.
cccccc
          if (lfnmin .and. lfdigt .and. ldigt(c)) then
c               write (*,*) c,'lfnmin,lfdigt'
                if (toklen .eq. 0) then
                   toknum = toknum + 1
                   if (toknum .gt. maxtok) then
                      write (eunit,*) 'ERROR: number of tokens exceeds '
     &                                ,'MAXTOK'
                      go to 999
                   end if
                end if
                toklen = min(mxtlen,toklen +1)
                token (toknum) (toklen:toklen) = c

                lfastr = true
                lfcmnt = true
                lfdigt = true
                lfexin = true
                lfexp  = false
                lfflnm = false
                lfkwin = false
                lfkywd = false
                lfmdfr = false
                lfmdin = false
                lfnmbr = true
                lfnmin = false
                lfperd = true
                lfqtin = false
                lfqtfn = false
                lfsepr = true
                lfsign = false
                lfvbar = true

                go to 199
          end if

cccccc
c         Look for number initial period.
cccccc
          if (lfnmin .and. lfperd .and. lperd(c)) then
c               write (*,*) c,'lfnmin,lfperd'
                if (toklen .eq. 0) then
                   toknum = toknum + 1
                   if (toknum .gt. maxtok) then
                      write (eunit,*) 'ERROR: number of tokens exceeds '
     &                                ,'MAXTOK'
                      go to 999
```

```
                    end if
                 end if
                 toklen = min(mxtlen,toklen +1)
                 token (toknum) (toklen:toklen) = c

                 lfastr = false
                 lfcmnt = true
                 lfdigt = true
                 lfexin = false
                 lfexp  = false
                 lfflnm = false
                 lfkwin = false
                 lfkywd = false
                 lfmdfr = false
                 lfmdin = false
                 lfnmbr = true
                 lfnmin = false
                 lfperd = false
                 lfsepr = false
                 lfsign = false
                 lfvbar = true

                 go to 199
              end if

cccccc
c             Look for number digit.
cccccc
              if (lfnmbr .and. lfdigt .and. ldigt(c)) then
c                      write (*,*) c,'lfnmbr,lfdigt'
                 toklen = min(mxtlen,toklen +1)
                 token (toknum) (toklen:toklen) = c

                 lfastr = true
                 lfcmnt = true
                 lfdigt = true
                 lfexin = true
                 lfexp  = false
                 lfflnm = false
                 lfkwin = false
                 lfkywd = false
                 lfmdfr = false
                 lfmdin = false
                 lfnmbr = true
                 lfnmin = false
                 lfperd = true
                 lfqtin = false
                 lfqtfn = false
                 lfsepr = true
                 lfsign = false
                 lfvbar = true

                 go to 199
              end if

cccccc
c             Look for number internal period.
```

```
cccccc
          if (lfnmbr .and. lfperd .and. lperd(c)) then
c               write (*,*) c,'lfnmbr,lfperd'
                toklen = min(mxtlen,toklen +1)
                token (toknum) (toklen:toklen) = c
                pcount = pcount + 1

                lfastr = false
                lfcmnt = true
                lfdigt = true
                lfexin = true
                lfexp  = false
                lfflnm = false
                lfkwin = false
                lfkywd = false
                lfmdfr = false
                lfmdin = false
                lfnmbr = true
                lfnmin = false
                lfperd = false
                lfqtin = false
                lfqtfn = false
                lfsepr = true
                lfsign = false
                lfvbar = true

                go to 199
          end if

cccccc
c         Look for exponent initial letter E or D.
cccccc
          if (lfnmbr .and. lfexin .and. lexin(c)) then
Cc              write (*,*) c,'lfnmbr,lfexin'
c
c               Add a period if there is none for this token yet.
c
                if (pcount .eq. 0) then
                   toklen = min(mxtlen,toklen + 1)
                   token(toknum) (toklen:toklen) = '.'
                   pcount = pcount + 1
                end if
                toklen = min(mxtlen,toklen +1)
                token (toknum) (toklen:toklen) = c

                lfastr = false
                lfcmnt = true
                lfdigt = true
                lfexin = false
                lfexp  = true
                lfflnm = false
                lfkwin = false
                lfkywd = false
                lfmdfr = false
                lfmdin = false
                lfnmbr = false
                lfnmin = false
                lfperd = false
                lfqtin = false
                lfqtfn = false
                lfsepr = false
                lfsign = true
                lfvbar = true

                go to 199
          end if

cccccc
c         Look for exponent initial sign.
cccccc
          if (lfexp .and. lfsign .and. lsign(c)) then
Cc              write (*,*) c,'lfexp,lsign'
                toklen = min(mxtlen,toklen +1)
                token (toknum) (toklen:toklen) = c

                lfastr = false
                lfcmnt = false
                lfdigt = true
                lfexin = false
                lfexp  = true
                lfflnm = false
                lfkwin = false
                lfkywd = false
                lfmdfr = false
                lfmdin = false
                lfnmbr = false
                lfnmin = false
                lfperd = false
                lfqtin = false
                lfqtfn = false
                lfsepr = false
                lfsign = false
                lfvbar = false

                go to 199
          end if

cccccc
c         Look for exponent initial digit.
cccccc
          if (lfexp .and. lfdigt .and. ldigt(c)) then
Cc              write (*,*) c,'lfexp,lfdigt,ldigt'
                toklen = min(mxtlen,toklen +1)
                token (toknum) (toklen:toklen) = c

                lfastr = false
                lfcmnt = true
                lfdigt = true
                lfexin = false
                lfexp  = true
                lfflnm = false
                lfkwin = false
                lfkywd = false
                lfmdfr = false
```

```
                 lfmdin = false
                 lfnmbr = false
                 lfnmin = false
                 lfperd = false
                 lfqtin = false
                 lfqtfn = false
                 lfsepr = true
                 lfsign = false
                 lfvbar = true

                 go to 199
            end if

cccccc
c           Look for exponent secondary digits.
cccccc
c            if (lfnmbr .and. lfdigt .and. ldigt(c)) then
Cc                write (*,*) c,'lfnmbr,lfdigt'
c                  toklen = min(mxtlen,toklen +1)
c                  token (toknum) (toklen:toklen) = c
c
c                  lfastr = false
c                  lfcmnt = true
c                  lfdigt = true
c                  lfexin = false
c                  lfexp  = false
c                  lfflnm = false
c                  lfkwin = false
c                  lfkywd = false
c                  lfmdfr = false
c                  lfmdin = false
c                  lfnmbr = true
c                  lfnmin = false
c                  lfperd = false
c                  lfqtin = false
c                  lfqtfn = false
c                  lfsepr = true
c                  lfsign = false
c                  lfvbar = true
c
c                  go to 199
c            end if

cccccc
c           Look for asterisk (*) operator.
cccccc
            if (lfastr .and. lastr(c)) then
Cc               write (*,*) c,'lfastr'
                 toknum = toknum + 1
                 if (toknum .gt. maxtok) then
                    write (eunit,*) 'ERROR: number of tokens exceeds '
     &                               ,'MAXTOK'
                    go to 999
                 end if

                 toklen = 1
                 token (toknum) (toklen:toklen) = c
```

```
C                A token length of 0 will flag other sections of the code
C                that an initial character of a number or keyword is
C                expected.

                 toklen = 0

                 lfastr = false
                 lfcmnt = true
                 lfdigt = true
                 lfexin = false
                 lfexp  = false
                 lfflnm = false
                 lfkwin = false
                 lfkywd = false
                 lfmdfr = false
                 lfmdin = false
                 lfnmbr = false
                 lfnmin = true
                 lfperd = true
                 lfqtin = true
                 lfqtfn = false
                 lfsepr = true
                 lfsign = true
                 lfvbar = true

                 go to 199
            end if

CCCCCC
C           Look for initial quote (') operator.
CCCCCC
            if (lfqtin .and. lquote(c)) then
C                write (*,*) c,'lfquote'
                 toknum = toknum + 1
                 if (toknum .gt. maxtok) then
                    write (eunit,*) 'ERROR: number of tokens exceeds '
     &                              ,'MAXTOK'
                    go to 999
                 end if

                 toklen = 1
                 token (toknum) (toklen:toklen) = c

                 lfastr = false
                 lfcmnt = false
                 lfdigt = true
                 lfexin = false
                 lfexp  = false
                 lfflnm = true
                 lfkwin = false
                 lfkywd = false
                 lfmdfr = false
                 lfmdin = false
                 lfnmbr = false
                 lfnmin = false
                 lfperd = false
                 lfqtin = false
```

```
                  lfqtfn = true
                  lfsepr = true
                  lfsign = false
                  lfvbar = true

                  go to 199
            end if

cccccc
c           Look for final quote (') operator.
cccccc
            if (lfqtfn .and. lquote(c)) then
c                write (*,*) c,'final quote'
cccccc
c                 Disregard final quote as part of a token, but change
c                 the analyzer state variables.
cccccc
                  toklen = 0

                  lfastr = false
                  lfcmnt = true
                  lfdigt = true
                  lfexin = false
                  lfexp  = false
                  lfflnm = false
                  lfkwin = false
                  lfkywd = false
                  lfmdfr = false
                  lfmdin = false
                  lfnmbr = false
                  lfnmin = false
                  lfperd = false
                  lfqtin = true
                  lfqtfn = false
                  lfsepr = true
                  lfsign = false
                  lfvbar = true

                  go to 199
            end if

cccccc
c           Look for generic string, also used for file names.
cccccc
            if (lfflnm .and. lflnm(c)) then
c                write (*,*) c,'lfflnm'
                  toklen = min(mxtlen,toklen +1)
                  token (toknum) (toklen:toklen) = c

                  lfastr = true
                  lfcmnt = true
                  lfdigt = true
                  lfexin = false
                  lfexp  = false
                  lfflnm = true
                  lfkwin = false
                  lfkywd = false
                  lfmdfr = false
                  lfmdin = false
                  lfnmbr = false
                  lfnmin = false
                  lfperd = false
                  lfqtin = false
                  lfqtfn = true
                  lfsepr = false
                  lfsign = false
                  lfvbar = true

                  go to 199
            end if

cccccc
c           Look for separators.
cccccc
            if (lfsepr .and. lsepr(c)) then
c                 write (*,*) c,'lfsepr'
                  if (.not. lsepr(lastc)) then
                    toklen = 0
                  end if

                  pcount = 0
c                 lfastr =
                  lfcmnt = true
                  lfdigt = true
                  lfexin = false
                  lfexp  = false
                  lfflnm = false
                  lfkwin = false
                  lfkywd = false
                  lfmdfr = false
                  lfmdin = true
                  lfnmbr = false
                  lfnmin = true
                  lfperd = true
                  lfqtin = true
                  lfqtfn = false
                  lfsepr = true
                  lfsign = true
                  lfvbar = true

c                 write (*,*)
c     &            'lfastr', lfastr ,
c     &            'lfcmnt', lfcmnt ,
c     &            'lfdigt', lfdigt ,
c     &            'lfexin', lfexin ,
c     &            'lfexp ', lfexp  ,
c     &            'lfflnm', lfflnm ,
c     &            'lfkwin', lfkwin ,
c     &            'lfkywd', lfkywd ,
c     &            'lfmdfr', lfmdfr ,
c     &            'lfmdin', lfmdin ,
c     &            'lfnmbr', lfnmbr ,
c     &            'lfnmin', lfnmin ,
c     &            'lfperd', lfperd ,
C     &          'lfqtin', lfqtin ,
C     &          'lfqtfn', lfqtfn ,
C     &          'lfsepr', lfsepr ,
C     &          'lfsign', lfsign ,
C     &          'lfvbar', lfvbar

                 go to 199
              end if

CCCCCC
C             All others are errors.
CCCCCC
              if (toknum .eq. 0) then
                 write (eunit,*) 'Keyword was not found as first token.'
                 rderr = 1
              else
                 write (eunit,*)
     &              ' ERROR:',c,' in column',i,' toknum=',toknum,
     &              ' toklen=', toklen
                 rderr = 2
                 go to 999
              end if

  199 continue

C         Save current character into previous character buffer.

          lastc = c
  200 continue
CCCCCC
C         End of record.
CCCCCC
          if (.not. lsepr(lastc)) then
             toklen = 0
          end if

          pcount = 0
C         lfastr =
          lfcmnt = true
          lfdigt = true
          lfexin = false
          lfexp  = false
          lfflnm = false
          lfkwin = false
          lfkywd = false
          lfmdfr = false
          lfmdin = true
          lfnmbr = false
          lfnmin = true
          lfperd = true
          lfqtin = true
          lfqtfn = false
          lfsepr = true
          lfsign = true
          lfvbar = true

          go to 100
```

```
CCCCCC
C       Exit.
CCCCCC
  990   CONTINUE
        RDERR = -1

  999   CONTINUE
        return
        end
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
        function upcase (c)
CCCCCC
C       NAME: upcase
C
C       PURPOSE: Change the case of any lower case alphabetic character to
C                upper case.
C
C       ARGUMENTS:
C               input:
C               c = CHARACTER*1 to be used for possible upshift.
C
C               output:
C               upcase = CHARACTER*1 return value.
C
C       EXAMPLE 1:
C               c = 'a'
C               upcase = 'A'
C
C       REFERENCES:
C               none
C
C       CHANGES:
C               1.00    08-25-90        Ron Janetzke
C                       Original text.
CCCCCC
C
        CHARACTER*1 upcase
        CHARACTER*1 c

        if (ichar(c).ge.ichar('a') .and. ichar(c).le.ichar('z')) then
           upcase = char(ichar(c)-32)
        else
           upcase = c
        end if

        return
        end
CCCCCCCCCCC
CCCCCCCCCCC
        function lastr(c)
CCCCCC
C       NAME: lastr
C
C       PURPOSE: Compare c to the replication operator character (*) for RDTOK.
C
C       ARGUMENTS:
C               input:
```

```
C               c = CHARACTER*1 to be used for the compare.
C
C               output:
C               lastr = LOGICAL return value.
C
C       EXAMPLE 1:
C               c = '*'
C               lastr = .true.
C
C       REFERENCES:
C               none
C
C       CHANGES:
C               1.00    08-25-90        Ron Janetzke
C                       Original text.
CCCCCC
C
        LOGICAL lastr
        CHARACTER*1 c

        if (c .eq. '*') then
          lastr = .true.
        else
          lastr = .false.
        end if

        return
        end
CCCCCCCCCCC
CCCCCCCCCCC
        function lcmnt(c)
CCCCCC
C       NAME: lcmnt
C
C       PURPOSE: Compare c to the comment initiator character (\) for RDTOK.
C
C       ARGUMENTS:
C               input:
C               c = CHARACTER*1 to be used for the compare.
C
C               output:
C               lcmnt = LOGICAL return value.
C
C       EXAMPLE 1:
C               c = '\'
C               lcmnt = .true.
C
C       REFERENCES:
C               none
C
C       CHANGES:
C               1.00    08-25-90        Ron Janetzke
C                       Original text.
CCCCCC
C
        LOGICAL lcmnt
        CHARACTER*1 c
        if (c .eq. '\') then
          lcmnt = .true.
        else
          lcmnt = .false.
        end if

        return
        end
CCCCCCCCCCC
CCCCCCCCCCC
        function ldigt(c)
CCCCCC
C       NAME: ldigt
C
C       PURPOSE: Compare c to the digit (0-9) character set for RDTOK.
C
C       ARGUMENTS:
C               input:
C               c = CHARACTER*1 to be used for the compare.
C
C               output:
C               ldigt = LOGICAL return value.
C
C       EXAMPLE 1:
C               c = '0'
C               ldigt = .true.
C
C       REFERENCES:
C               none
C
C       CHANGES:
C               1.00    08-25-90        Ron Janetzke
C                       Original text.
CCCCCC
C
        LOGICAL ldigt
        CHARACTER*1 c

        if (ichar(c).ge.ichar('0') .and. ichar(c).le.ichar('9')) then
          ldigt = .true.
        else
          ldigt = .false.
        end if

        return
        end
CCCCCCCCCCC
CCCCCCCCCCC
        function lquote(c)
CCCCCC
C       NAME: lquote
C
C       PURPOSE: Compare c to the string assignment operator (=) for RDTOK.
C
C       ARGUMENTS:
C               input:
C               c = CHARACTER*1 to be used for the compare.
```

```
C
C               output:
C               lquote = LOGICAL return value.
C
C       EXAMPLE 1:
C               c = ''''
C               lquote = .true.
C
C       REFERENCES:
C               none
C
C       CHANGES:
C               1.00    09-24-90        Ron Janetzke
C                       Redo of function 'lequl' to make paired quotes
C                       required for string data values.
CCCCCC
C
        LOGICAL lquote
        CHARACTER*1 c

        if (c .eq. '''') then
          lquote = .true.
        else
          lquote = .false.
        end if

        return
        end
CCCCCCCCCCC
CCCCCCCCCCC
        function lexin(c)
CCCCCC
C       NAME: lexin
C
C       PURPOSE: Compare c to the exponent initiator characters (E,D)
C                for RDTOK.
C
C       ARGUMENTS:
C               input:
C               c = CHARACTER*1 to be used for the compare.
C
C               output:
C               lexin = LOGICAL return value.
C
C       EXAMPLE 1:
C               c = 'E'
C               lexin = .true.
C
C       REFERENCES:
C               none
C
C       CHANGES:
C               1.00    08-25-90        Ron Janetzke
C                       Original text.
CCCCCC
C
        LOGICAL lexin
        CHARACTER*1 c

        if (c .eq. 'E' .or. c .eq. 'D') then
          lexin = .true.
        else
          lexin = .false.
        end if

        return
        end
CCCCCCCCCCC
CCCCCCCCCCC
        function lflnm(c)
CCCCCC
C       NAME: lflnm
C
C       PURPOSE: Compare c to the file name character set for RDTOK.
C
C       ARGUMENTS:
C               input:
C               c = CHARACTER*1 to be used for the compare.
C
C               output:
C               lflnm = LOGICAL return value.
C
C       EXAMPLE 1:
C               c = 'A'
C               lflnm = .true.
C
C       REFERENCES:
C               none
C
C       CHANGES:
C               1.00    08-25-90        Ron Janetzke
C                       Original text.
C               1.01    09-24-90        Ron Janetzke
C                       Extend the beginning of the set from $ to !.
C               1.02    10-19-90        Ron Janetzke
C                       Extend the end of the range from ` to z.
C               1.03    10-24-90        Ron Janetzke
C                       Extend the beginning of the set from ! to SPACE.
CCCCCC
C
        LOGICAL lflnm
        CHARACTER*1 c

        if (ichar(c).ge.ichar(' ') .and. ichar(c).le.ichar('&') .or.
     &      ichar(c).ge.ichar('(') .and. ichar(c).le.ichar('z')) then
          lflnm = .true.
        else
          lflnm = .false.
        end if

        return
        end
CCCCCCCCCCC
CCCCCCCCCCC
```

```
        function lkwin(c)
CCCCCC
C       NAME: lkwin
C
C       PURPOSE: Compare c to the keyword initiator character set for RDTOK.
C
C       ARGUMENTS:
C               input:
C               c = CHARACTER*1 to be used for the compare.
C
C               output:
C               lkwin = LOGICAL return value.
C
C       EXAMPLE 1:
C               c = 'A'
C               lkwin = .true.
C
C       REFERENCES:
C               none
C
C       CHANGES:
C               1.00    08-25-90        Ron Janetzke
C                       Original text.
CCCCCC
C
        LOGICAL lkwin
        CHARACTER*1 c

        if (ichar(c).ge.ichar('A') .and. ichar(c).le.ichar('Z')) then
          lkwin = .true.
        else
          lkwin = .false.
        end if

        return
        end
CCCCCCCCCCC
CCCCCCCCCCC
        function lkywd(c)
CCCCCC
C       NAME: lkywd
C
C       PURPOSE: Compare c to the keyword secondary character set for RDTOK.
C
C       ARGUMENTS:
C               input:
C               c = CHARACTER*1 to be used for the compare.
C
C               output:
C               lkywd = LOGICAL return value.
C
C       EXAMPLE 1:
C               c = 'A'
C               lkywd = .true.
C
C       REFERENCES:
C               none
```

```
C
C       CHANGES:
C               1.00    08-25-90        Ron Janetzke
C                       Original text.
C               1.01    09-24-90        Ron Janetzke
C                       Extend the beginning of the set from $ to !.
CCCCCC
C
        LOGICAL lkywd
        CHARACTER*1 c

        if (ichar(c) .ge. ichar('!') .and. ichar(c) .le. ichar('+')
     &  .or.ichar(c) .ge. ichar('-') .and. ichar(c) .le. ichar('9')
     &  .or.ichar(c) .ge. ichar('@') .and. ichar(c) .le. ichar('_'))
     &  then
          lkywd = .true.
        else
          lkywd = .false.
        end if

        return
        end
CCCCCCCCCCC
CCCCCCCCCCC
        function lmdfr(c)
CCCCCC
C       NAME: lmdfr
C
C       PURPOSE: Compare c to the modifier secondary character set for RDTOK.
C
C       ARGUMENTS:
C               input:
C               c = CHARACTER*1 to be used for the compare.
C
C               output:
C               lmdfr = LOGICAL return value.
C
C       EXAMPLE 1:
C               c = 'A'
C               lmdfr = .true.
C
C       REFERENCES:
C               none
C
C       CHANGES:
C               1.00    08-25-90        Ron Janetzke
C                       Original text.
C               1.01    09-24-90        Ron Janetzke
C                       Extend the beginning of the set from $ to !.
CCCCCC
C
        LOGICAL lmdfr
        CHARACTER*1 c

        if (ichar(c) .ge. ichar('!') .and. ichar(c) .le. ichar('+')
     &  .or.ichar(c) .ge. ichar('-') .and. ichar(c) .le. ichar('9')
     &  .or.ichar(c) .ge. ichar('@') .and. ichar(c) .le. ichar('_'))
     &  then
          lmdfr = .true.
        else
          lmdfr = .false.
        end if

        return
        end
CCCCCCCCCCC
CCCCCCCCCCC
        function lmdin(c)
CCCCCC
C       NAME: lmdin
C
C       PURPOSE: Compare c to the modifier initiator character set for RDTOK.
C
C       ARGUMENTS:
C               input:
C               c = CHARACTER*1 to be used for the compare.
C
C               output:
C               lmdin = LOGICAL return value.
C
C       EXAMPLE 1:
C               c = 'A'
C               lmdin = .true.
C
C       REFERENCES:
C               none
C
C       CHANGES:
C               1.00    08-25-90        Ron Janetzke
C                       Original text.
CCCCCC
C
        LOGICAL lmdin
        CHARACTER*1 c

        if (ichar(c).ge.ichar('A') .and. ichar(c).le.ichar('Z')) then
          lmdin = .true.
        else
          lmdin = .false.
        end if

        return
        end
CCCCCCCCCCC
CCCCCCCCCCC
        function lnmbr(c)
CCCCCC
C       NAME: lnmbr
C
C       PURPOSE: Compare c to the number secondary character set for RDTOK.
C
C       ARGUMENTS:
C               input:
C               c = CHARACTER*1 to be used for the compare.
```

```
C
C               output:
C               lnmbr = LOGICAL return value.
C
C       EXAMPLE 1:
C               = '0'
C               lnmbr = .true.
C
C       REFERENCES:
C               none
C
C       CHANGES:
C               1.00    08-25-90        Ron Janetzke
C                       Original text.
CCCCCC
C
        LOGICAL lnmbr
        CHARACTER*1 c

        if (ichar(c) .ge. ichar('$') .and. ichar(c) .le. ichar('9')
     &  .or.ichar(c) .ge. ichar('@') .and. ichar(c) .le. ichar('_'))
     &  then
          lnmbr = .true.
        else
          lnmbr = .false.
        end if

        return
        end
CCCCCCCCCCC
CCCCCCCCCCC
        function lperd(c)
CCCCCC
C       NAME: lperd
C
C       PURPOSE: Compare c to the period (.) for RDTOK.
C
C       ARGUMENTS:
C               input:
C               c = CHARACTER*1 to be used for the compare.
C
C               output:
C               lperd = LOGICAL return value.
C
C       EXAMPLE 1:
C               c = '.'
C               lperd = .true.
C
C       REFERENCES:
C               none
C
C       CHANGES:
C               1.00    08-25-90        Ron Janetzke
C                       Original text.
CCCCCC
C
        LOGICAL lperd
        CHARACTER*1 c

        if (c .eq. '.') then
          lperd = .true.
        else
          lperd = .false.
        end if

        return
        end
ccccccccccc
ccccccccccc
        function lsepr(c)
cccccc
c       NAME: lsepr
c
c       PURPOSE: Compare c to the separator characters (SPACE, COMMA, COLON,
c                TAB, and EQUAL) for RDTOK.
c
c       ARGUMENTS:
c               input:
c               = CHARACTER*1 to be used for the compare.
c
c               output:
c               lsepr = LOGICAL return value.
c
c       EXAMPLE 1:
c               c = ','
c               lsepr = .true.
c
c       REFERENCES:
c               none
c
c       CHANGES:
c               1.00    08-25-90        Ron Janetzke
c                       Original text.
c               1.01    09-24-90        Ron Janetzke
c                       Add equal sign to list of separators.
c               1.02    02-05-91        Ron Janetzke
c                       Add TAB to list of separators.
cccccc
c
        LOGICAL lsepr
        CHARACTER*1 c

        if (c .eq. ' ' .or. c .eq. ',' .or. c .eq. ':' .or.
     &      c .eq. char(9) .or. c .eq. '=') then
          lsepr = .true.
        else
          lsepr = .false.
        end if

        return
        end


ccccccccc
```

```
CCCCCCCCCCC
        function lsign(c)
CCCCCC
C       NAME: lsign
C
C       PURPOSE: Compare c to the numeric sign characters (+,-)
C                for RDTOK.
C
C       ARGUMENTS:
C               input:
C               c = CHARACTER*1 to be used for the compare.
C
C               output:
C               lsign = LOGICAL return value.
C
C       EXAMPLE 1:
C               c = '-'
C               lsign = .true.
C
C       REFERENCES:
C               none
C
C       CHANGES:
C               1.00    08-25-90        Ron Janetzke
C                       Original text.
CCCCCC
C
        LOGICAL lsign
        CHARACTER*1 c

        if (c .eq. '+' .or. c .eq. '-') then
          lsign = .true.
        else
          lsign = .false.
        end if

        return
        end
CCCCCCCCCCC
CCCCCCCCCCC
        function lvbar(c)
CCCCCC
C       NAME: lvbar
C
C       PURPOSE: Compare c to the end of line character (¦)
C                for RDTOK.
C
C       ARGUMENTS:
C               input:
C               c = CHARACTER*1 to be used for the compare.
C
C               output:
C               lvbar = LOGICAL return value.
C
C       EXAMPLE 1:
C               c = '¦'
C               lvbar = .true.
```

```
c
c       REFERENCES:
c               none
c
c       CHANGES:
c               1.00    08-25-90        Ron Janetzke
c                       Original text.
cccccc
c
        LOGICAL lvbar
        CHARACTER*1 c

        if (c .eq. '¦') then
          lvbar = .true.
        else
          lvbar = .false.
        end if

        return
        end
ccccccccccc
      function exist (c,lenmst,master,elepos)
cccccc
c       Name: exist
c
c       Purpose:
c                This routine is part of a suite of routines used with
c                the data structures required for RDFREE.  It is for two
c                different purposes.  The first is as a function to check
c                a newly found token by the RDFREE routine to see if it
c                is a member of the master keyword list.  The second is
c                as a function for the application program to check for
c                a specific modifier in the keyword/modifier (mdfyr) list
c                returned from RDFREE.  This routine searches
c                the array 'master' for an element equal to 'c'.  It then
c                returns the element number in 'elepos'.  NOTE: 'c' must
c                be CHARACTER*n, where n is greater than or equal to 6.
c
c       ARGUMENTS:
c               input:
c              c        = CHARACTER*6 string to be matched to element of master.
c               lenmst  = INTEGER length of the 'master' keyword array.
c               master  = CHARACTER*6 array of keywords to be searched.
c
c               output:
c               elepos  = INTEGER element number of first matched string,
c                        0 if string is not found to match any in array master.
c               exist   = LOGICAL if true then string was found, if false
c                         the string was not found.
c
c       REFERENCES:
c               none
c
c       HISTORY:
c               1.00    09-04-90        Ron Janetzke
c                       Original text.
c               1.01    03-20-91        Ron Janetzke
```

```
C                       Error message for incorrect length of first argument.
C               2.00    05-05-91         Ron Janetzke
C                       Change to function for version 2.00 of RDFREE.
CCCCCC
      LOGICAL        exist
      CHARACTER*(*) c
      INTEGER        lenmst
      CHARACTER*6    master(lenmst)
      INTEGER        elepos
CCCCCC
      CHARACTER*6  temp
      INTEGER i
CCCCCC
C     Start here.
CCCCCC
      if (len(c) .lt. 6) then
         write (*,'(x,a76/x,a)')
     &          'The length of the first argument in call EXIST '//
     &          'must be at least CHARACTER*6.', c
         stop
      end if
      exist = .FALSE.
      temp = c
      elepos = 0
      if (lenmst .gt. 0) then
         do 199 i=1,lenmst
            if (temp .eq. master(i)) then
               elepos = i
               exist = .TRUE.
               go to 999
            end if
  199    continue
      end if
  999 continue
      return
      end
```

Appendix D

RDFREE GRAMMAR BACKUS-NAUR FORM

The Backus-Naur Form (BNF) for the RDFREE grammar can be given as

```
<input line> ::= <data line> | <comment line> | <data line> <comment line>
```

where

```
<comment line> ::= \<character string>
<data line>    ::= <keyword sequence> | <data line> <keyword sequence>
```

where

```
<keyword sequence> ::= <keyword> <string data>  |
                       <keyword> <numeric data> |
                       <keyword>
```

where

```
<keyword> ::= <alphanumeric string>
<numeric data> ::= <numeric> | <numeric data> <numeric>
<string data>  ::= '<character string>' | <string data> '<character string>'
```

where

```
<numeric> ::= <integer data> | <fixed point data> | <floating point data>
```

where

```
<integer data> ::= <integer> | <integer>*<integer>
<fixed point data> ::= <fixed point> | <integer>*<fixed point>
<floating point data> ::= <floating point> | <integer>*<floating point>
```