MacAnova Version 4.07

This file consists of Chapter 7 of **MacAnova User's Guide** by Gary W. Oehlert and Christopher Bingham, issued as Technical Report Number 617, School of Statistics, University of Minnesota, revised August 1998, describing Version 4.07 of MacAnova.

Fonts used in this manual are Palatino, Courier, and Symbol.

For information concerning MacAnova, write University of Minnesota, Department of Applied Statistics, 352 Classroom Office Building, 1994 Buford Avenue, St. Paul, MN 55108-6042.

7. Files

7.1 **Format of data readable by matread() and read()** Functions `matread()` and `read()` (Sec. 2.11.3) expect that data will be in a plain text or ASCII file in a specific format, described in this section. Most word processors have an option for saving text or ASCII files.

Here is a listing of file `data.txt` containing several data sets that `matread()` and `read()` can read. It illustrates most aspects of the required format.

```
y1              3     4
) Sample 3 by 4 matrix with 1 missing value
) MISSING -1
)) Comment line that will not normally be echoed
)) Such lines might give very extensive background information
)"%f %f %f %f"
  3.31662  3.00000  3.46410  3.31662
  3.16228  3.46410  2.00000 -1.00000
  2.44949  3.74166  2.82843  1.73205

y2              3     4 LOGICAL COLUMNS FORMAT
) This is the matrix (y1 > 3)
) MISSING -99
)"5x%f %f %f"
(5x,4f13.0)
Col.1           1            1            0
Col.2           0            1            1
Col.3           1            0            0
Col.4           1          -99            0

y3              2     2     4
) 2 by 2 by 2 array
) Rows in order are y3[1,1,],y3[1,2,],y3[2,1,],y3[2,2,]
) MISSING -99
)"%f %f %f %f"
 53.222   56.057   44.683   46.343
 36.147   56.357   47.651   54.889
 49.979   62.120  -99.000   35.926
 45.278   31.604   27.964   54.206

people          2     2 CHARACTER
) 2 by two matrix of unquoted CHARACTER elements with no embedded
) blanks or tabs
)"%s %s"
Tom             Harry
Dick            Elizabeth

experinfo       3 CHARACTER
) Each line of data is a CHARACTER element
Data were obtained in a randomized complete block experiment
in 7 replicates.  Treatments were control plus 3 levels of
Nitrogen.
```

```
Desserts           3 QUOTED COLUMNS
) Data are quoted CHARACTER elements
)"%s %s %s"
"Ice cream" "Strawberries" "Short cake
with whipped cream"

nulldata    NULL
) This is a null data set.  It must be followed by a blank line

mystruc  2  STRUCTURE
) This is a structure with two components, a and b
) The blank line before the header of each component is required

mystruc$a  2 QUOTED COLUMNS
) CHARACTER vector of length 2
) Two quoted fields
"The quick brown fox" "Jumps over the lazy dog"

mystruc$constants  2 STRUCTURE
) This component is a structure with two components, pi and e

mystruc$constants$pi  1   1
) 1 by 1 matrix
3.14159265358979

mystruc$constants$e   1
) vector of length 1
2.71828182845905

labelled     4     3 COLUMNS LABELS NOTES
) Small REAL data set with one missing value coded as -99.
) Each line contains data for one column (COLUMNS on header)
) MISSING -99
) '4x' in the following format skips 4 characters (variable label)
)"4x%f %f %f %f"
Temp   34.5   45.2  23.1   20.1
Conc   .170   -99  .883    .401
Secs    3.5   4.7   3.2     5.8

labelled$LABELS    7   QUOTED COLUMNS
) Labels for sample data in quoted format by columns
)"%s %s %s %s %s %s %s"
"@" "@" "@" "@" "Temp" "Conc" "Secs"

labelled$NOTES     2 CHARACTER
) Notes for sampledata in "by line" format
Small REAL data set with row and column labels.
There is one missing value.
```

The first line or *name line* of each data set contains its name, together with dimension information or number of components and sometimes other information. Here the first two data sets are 3 by 4 matrices, a REAL matrix with name y1 and a LOGICAL matrix with name y2. The next one is 2 by 2 by 3 REAL array y3. Next come three CHARACTER data sets, a 2 by 2 matrix people and vectors, experinfo and Desserts, each of length 3. Data set nulldata represents a NULL variable. Data set mystruc is a structure with two components, one of which is itself a structure. Its components have the same form as other data sets, but their names reflect their position in the structure.

The last data set, labelled, is a REAL matrix with labels for rows and columns (Sec. 8.4) as well as attached descriptive notes (Sec. 8.9). The labels and notes are CHARACTER vectors named labelled$LABELS and labelled$NOTES, respectively, and are in the same form as other data sets.

The name line is the only *required* header line. It may optionally be followed by comment lines and then lines containing data. See below

Information on the name line in addition to the name and dimensions is in the form of keywords. Here is a table of the recognized keywords.

Keyword	Meaning
CHARACTER	The data set is a CHARACTER variable in either "by fields" or "by lines" format (see below)
COLS or COLUMNS	The data follow in transposed form. For a matrix, this is in column by column order, each column starting on a new line.
FORMAT	Indicates that a Fortran format starting with '(' will follow the last comment line. It is ignored by MacAnova but would be helpful for a program written in Fortran to read the data.
LABELS	The data set has coordinate labels which follow the data in the file (see below)
LOGICAL	The data are to be interpreted as LOGICAL with False and True coded as 0 and 1, respectively (Sec. 7.1.1).
NOTES	The data set has attached descriptive notes which follow the data in the file (see below).
NULL	The data set is a NULL variable, containing no data, although there may be comment lines. There can be no other keywords.
QUOTED	The data set is a CHARACTER variable in "by quoted fields" format (see below).
REAL	The data set is a REAL variable. Since this is the default, REAL is never required (Sec. 7.1.1).
ROWS	The data follow a row at a time (constant value for first subscript), each row starting on a new line. Since this is the default, ROWS is never required
STRUCTURE	The data set is a structure.

REAL, LOGICAL, CHARACTER, NULL and STRUCTURE on the name line specify the type of variable. QUOTED also implies CHARACTER variable. Since the default is REAL data, keyword REAL is never required.

When neither COLUMNS or COLS is on the name line, the data, whether REAL, LOGICAL or CHARACTER will be expected to be ordered so that the last subscript changes fastest, and whenever the next to last subscript changes, data start on a new line. For a vector, each element is on a separate line. For a matrix, this means that the data are read row by row, with each row starting on a new line. For example the first row of y1 consists of

the numbers 3.31662, 3.00000, 3.46410, and 3.31662, the first row of numbers. This order can also be signalled by ROWS on the name line, but this is never required.

When either COLUMNS or COLS *is* on the name line, the data are in the file *column by column*, that is with the first subscript changing most rapidly, and a new line starting whenever the second subscript, if any, changes. For a vector, all the elements are on a single line, unless split between several; see below. Thus, although y2 is 3 by 4, it is on the file in transposed form with 4 lines of 3 values each, with the first line of data being column 1 of matrix y2. Note that Desserts, with COLUMNS, puts the entire vector on one line, except that the last element spills over to a second line. Here is how array y3 would be with COLUMNS on the name line.

```
y3                2     2     4 COLUMNS
) 2 by 2 by 2 array
) Rows in order are y3[,1,1],y[,2,1],y[,1,2],y[,2,2],
)                   y3[,1,3],y[,2,3],y[,1,4],y[,2,4]
) MISSING:          -99
)"%f %f"
 53.222   49.979
 36.147   45.278
 56.057   62.120
 56.357   31.604
 44.683  -99.000
 47.651   27.964
 46.343   35.926
 54.889   54.206
```

Each line of data corresponds to a different value of the last subscript.

Following the name line may be *comment lines* starting with with ")" at the start of the line. The purpose of comment lines is to provide descriptive and/or formatting information about the data set. The only time a comment line is required is when there are MISSING values in the data set; see below.

matread() and read() normally echo comment lines except lines starting with "))", to output when the data set is found. You can suppress this echoing by including quiet:T as an argument (Sec. 7.1.5). You can force echoing of *all* lines starting with ")" by including quiet:F as an argument. The suppression of echoing lines starting with "))" is to allow you to include extensive comments with a data set without them all being echoed. Matrix y1 in the file contains such lines.

```
Cmd> y <- matread("data.txt", "y1") # '))' lines not echoed.
y1                3     4
) Sample 3 by 4 matrix with 1 missing value
) MISSING -1
```

```
Cmd> y <- matread("data.txt", "y1", quiet:F) #echo '))' lines
y1                3     4
) Sample 3 by 4 matrix with 1 missing value
) MISSING -1
)) Comment line that will not normally be echoed
)) Such lines might give very extensive background information
```

7.1.1 **REAL and LOGICAL data** The absence of LOGICAL, CHARACTER, QUOTED, NULL, and STRUCTURE or the presence of REAL on the name line means the data set is a REAL vector, matrix or array. The presence of LOGICAL on the name line indicates the data set is LOGICAL with 1 translated as True and 0 as False. Any other numerical values will be treated as True and a warning message printed. Note that the values must be numerical and not "T" and "F".

FORMAT is also legal on the name line of REAL or LOGICAL data sets, but has no special meaning in MacAnova. It indicates the presence of a Fortran style format specification after the comment lines but before the data. A computer program written in the Fortran programming language might use this information to retrieve data from the file.

For y1, y2 and y3, the comment lines "`) MISSING -1`" or "`) MISSING -99`" informs `matread()` or `read()` that data items with value –1 or –99 should be considered to be missing. Any value that does not otherwise appear in the data set can be used, but preferably an integer.

For compatibility with earlier versions of MacAnova, an apparently REAL data set with a comment line of the form

```
) LOGICAL
```

is considered to be LOGICAL even the first line doesn't contain LOGICAL.

A comment line of the form `)"%f %f ..."` or `)"%lf %lf ..."` has a special meaning. What matters is the number *m* of repetitions of `%f` or `%lf`. *m* is the definitive specification of the maximum number of REAL or LOGICAL data items on any line in the data set in the file. Usually *m* will be the same as the number of columns (or the number of rows, if COLUMNS was specified). However, when *m* is smaller, this indicates each row or column is split among several lines, each, except possibly the last, with *m* values. For example, if each row of y1 were split between two lines, with the first line having *m* = 3 numbers and the second 1 number, `)"%f %f %f"` would be required for it to be read correctly. If *m* is larger than what the name line specifies, the name line specification prevails.

As exemplified by data set y2, the `)"%f ...` line can also be of the form, say, `)"5x%f %f %f"`. The "5x" specifies that the first 5 characters of each line should be skipped, perhaps because, as is the case here, they provide labeling information for the line.

The line following the `)"%f..."` line for y2 is the optional Fortran format line starting with "(" whose presence was signalled by FORMAT on the first line. This is ignored by MacAnova except as a signal that there are no more comment lines.

The data immediately follow the optional comment and Fortran format lines. Data items must be separated by blanks or tabs; each change of the second subscript (or next to last subscript when COLUMNS is on the name line) must start on a new line. Unless COLUMNS is on the name line, each element of a vector must be on a separate line. Non-numerical items in a line (including "T" and "F" for LOGICAL data) stop the scanning of the line, with any further items expected in the line set to MISSING.

7.1.2 **CHARACTER data** The presence of QUOTED or CHARACTER on the name line indicates that the data set is a CHARACTER variable.

QUOTED on the name line signals that every element is enclosed in double quotes ("..."). There is no restriction on what can be inside the quotes, except that you must use \" and \\ to include double quotes and backslashes. A single element can in fact span several lines until terminated by a closing '"'. For example the third element of Desserts consists of two lines. In the default form, each element must start on a separate line.

CHARACTER on the name line indicates CHARACTER data that will *not* be enclosed in double quotes. The default form is *one element per line* with no restrictions on what can be in the line. A CHARACTER data element can be split between two or more lines by ending every line but the last with "\". See the next paragraph for how you can specify several items per line.

For CHARACTER data a comment line of the form)"%s %s ... " has a special meaning. If *m* is the number of "%s" in the line, then *m* is the maximum number of CHARACTER elements per line. When CHARACTER appears on the name line, each *word* (sequence of non blank characters including quotation marks) is read as a single element. If QUOTED appears on the name line, elements of CHARACTER data are expected to be enclosed in quotation marks ("...") and a single element can actually span several lines in the file, as is the case with data set Desserts.

A comment line of the form, for example,)"4x%s %s" specifies the first 4 characters in each line are to be skipped.

When CHARACTER is on the name line and there is no comment line of the form)"%s %s ... ", the entire contents of a line (or several lines, when a line is continued by ending it with "\") forms a single data element. This is referred to as "by lines" format. Data set Desserts could also have been in the following form.

```
Desserts           3 CHARACTER
) Data are in by lines format
Ice cream
Strawberries
Short cake\
with whipped cream
```

If QUOTED is on the first line, and there is no)"%s %s ... " comment line, each element is quoted and must start on a new line regardless of the dimensions.

Following the header and comment lines come the CHARACTER data. Each row of data (column if COLUMNS was specified) must start on a separate line. If there is a line of the form)"%s ..., multiple data items in a line must be separated by blanks or tabs.

7.1.3 **Structures** The presence of STRUCTURE on the name line indicates the data set is a structure. After comments lines, if any, come the components. Each component is in the standard form for a REAL, LOGICAL, CHARACTER or NULL data set or is itself a structure. As shown in the sample file, each component data set's name is of the form structurename$compname, analogously to how you reference structure components by name. If a structure component is itself a structure, its components have names of

the form structurename$compname1$compname2 (for example, mystruc$constants$pi).

Each component of a structure can be read separately by specifying its full name.

```
Cmd> matread("data.txt","mystruc$constants$pi")
mystruc$constants$pi  1   1
) 1 by 1 matrix
(1,1)       3.1416
```

7.1.4 **Labels and notes** Coordinate labels (Sec. 8.4) and descriptive notes (Sec. 8.9) attached to a variable can be included as CHARACTER vectors in the file in a similar format as structure components.

The presence of labels or notes attached to a variable is signalled by LABELS or NOTES on the name line. If the name of the data set is x, say, the names of its labels and notes, if any, is x$LABELS and x$NOTES. If a structure component, say y$time, has labels or notes, they have names y$time$LABELS and y$time$NOTES.

x$LABELS and/or x$NOTES must follow all the content of a variable. If x is a structure, x$LABELS and/or x$NOTES must follow all the information, data, labels and notes, of the last component of x.

The length of the CHARACTER vector x$LABELS must match the sum of the dimensions of x. The elements of the vector are the labels for the first dimension, followed by the labels for the second dimension, if any, and so on. Generally, since labels tend to be short, the name line for x$LABELS should contain QUOTED and COLUMNS and there should be a comment line of the form)"%s %s ...%s" specifying how many labels are expected per line. See Sec. 7.1.2.

There are no limitations on the length of CHARACTER vector x$NOTES. The will usually in unquoted “by lines” format signalled by CHARACTER on the name line and no comment line of the form)"%s %s ...%s". See Sec. 7.1.2.

Because labels and notes are CHARACTER data sets in their own right, they can be read directly by, say, matread(fileName,"x$LABELS") and matread(fileName, "x$Notes").

Here is an example consisting of reading data set labelled in file data.txt listed in Sec. 7.1.

```
Cmd> labelled <- read("data.txt","labelled")
labelled     4     3 COLUMNS LABELS NOTES
) Small REAL data set with one missing value coded as -99.
) Each line contains data for one column (COLUMNS on header)
) MISSING -99
) '4x' in the following format skips 4 characters (variable label)
```

```
Cmd> labelled
            Temp         Conc         Secs
(1)         34.5         0.17          3.5
(2)         45.2      MISSING          4.7
(3)         23.1        0.883          3.2
(4)         20.1        0.401          5.8

Cmd> getnotes(labelled) # see Sec. 8.9.1
(1) "Small REAL data set with row and column labels."
(2) "There is one missing value."

Cmd> read("data.txt","labelled$LABELS") # just read labels
labelled$LABELS    7   QUOTED COLUMNS
) Labels for sample data in quoted format by columns
(1) "@"
(2) "@"
(3) "@"
(4) "@"
(5) "Temp"
(6) "Conc"
(7) "Secs"

Cmd> read("data.txt","labelled$NOTES") # just read notes
labelled$NOTES     2 CHARACTER
) Notes for sampledata in "by line" format
(1) "Small REAL data set with row and column labels."
(2) "There is one missing value.
```

7.1.5 **matread() and read() keywords** Here are is a table of keyword phrases that can be used as arguments to `matread()` and `read()`.

Keyword Phrase	Meaning
`quiet:T`	Header and descriptive comments will not be printed
`quiet:F`	*All* header and descriptive comments will be printed
`echo:T`	Data lines will be printed as they are read
`silent:T`	Print only error messages; incompatible with `quiet:F` or `echo:T`
`notfoundok:T`	Failure to find data set is not an error

When `notfoundok:T` is an argument and the wanted data set is not found, no error message is printed and the value returned by `matread()` and `read()` is `NULL`.

In addition to these, you can use keyword `string` to "read" from a `CHARACTER` variable rather than a file. See Sec. 7.3.

7.2 **Reading CHARACTER data from files** `matread()`, `read()` and `vecread()` can read `CHARACTER` data from a file.

`matread()` and `read()` can read a named `CHARACTER` data set, which may be a scalar, vector, matrix or array in the special format described in Sec. 7.1.2. Here is an example reading from the file `data.txt` listed in Sec. 7.1.1-7.1.4.

```
Cmd> desserts <- matread("data.txt","Desserts") # or read(...)
Desserts           3 QUOTED COLUMNS
) Data are quoted CHARACTER elements

Cmd> desserts
(1) "Ice cream"
(2) "Strawberries"
(3) "Short cake
with whipped cream"
```

No distinction is made between upper and lower case letters in the names of data sets in a file, so `matread("data.txt","desserts")` or `matread("data.txt", "DESSERTS")` would also retrieve `Desserts`. See Sec. 7.1.2 for full information on how to format `CHARACTER` data in a file so that it can be read by `matread()` and `read()`.

You can also read `CHARACTER` data using `vecread()`. You must include one of the keyword phrases `bywords:T`, `bylines:T` or `bychars:T` as an argument in addition to the file name. `character:T` means the same as `bywords:T`. Here is a listing of file `chardata.txt` which will be used as an example.

```
#data on 2 children
Henry   Male   67.3,10.5
Susan   Female 59.2,15.1
    !
Other stuff ...
```

`vecread(fileName,bywords:T)` or `vecread(fileName,character:T)` reads `CHARACTER` data from a file with each "word" – a sequence of visible characters, excluding commas and the "stopping" character – being read as a single `CHARACTER` element. "Words" may be separated by spaces, tabs or commas (see also Sec. 2.11.1). As when reading `REAL` data, scanning of the file is terminated by a "stop character" which by default is "`!`".

```
Cmd> vecread("chardata.txt",bywords:T) # or character:T
 (1) "#data"
 (2) "on"
 (3) "2"
 (4) "children"
 (5) "Henry"
 (6) "Male"
 (7) "67.3"
 (8) "10.5"
 (9) "Susan"
(10) "Female"
(11) "59.2"
(12) "15.1"
```

Note that `vecread()` did not retrieve `"Other"`, `"stuff"` and `"..."` because they came after the stop character "`!`".

You can change the stop character to "`$`", say, by keyword phrase `stop:"$"`. To ensure the *entire* file will be read, change the stop character to one you know is not in the file. A good bet is `stop:"\377"` but you can use any non-ascii character between `"\200"` and `"\377"` that does not occur in the file. Skipped lines will be echoed to the screen if

you use quiet:F.

```
Cmd> words <- vecread("chardata.txt",bywords:T,stop:"\377")

Cmd> words[-run(11)] # this time we got everything
(1) "15.1"
(2) "!"
(3) "Other"
(4) "stuff"
(5) "..."
```

You can use also keywords skip, go and quiet as when reading REAL data (Sec. 2.11.1).

```
Cmd> words <- vecread("chardata.txt",bywords:T,skip:"#",quiet:F)
#data on 2 children        Printed because of quiet:F

Cmd> words
(1) "Henry"
(2) "Male"
(3) "67.3"
(4) "10.5"
(5) "Susan"
(6) "Female"
(7) "59.2"
(8) "15.1"
```

vecread(fileName,bylines:T) reads CHARACTER data from a file, treating each line in the file as a separate item. The stop character terminates the reading *only* when it occurs at the start of the line.

```
Cmd> lines <- vecread("chardata.txt",bylines:T,skip:"#",quiet:T)

Cmd> lines
(1) "Henry   Male   67.3,10.5"
(2) "Susan   Female 59.2,15.1"
(3) "    !"
(4) "Other stuff .."
```

The "!" did not stop the read because it was not the first character in the line. The line starting with "#" was skipped, but not printed because of quiet:T.

vecread(fileName,bychars:T) reads CHARACTER data from a file, returning a CHARACTER vector each element of which is an individual *character* in the file. As with bylines:T, the stop character terminates the reading *only* when it is the first character in the line. All characters, in the file including end-of-line characters "\n" are returned.

```
Cmd> chars <- vecread("chardata.txt",bychars:T,skip:"#",quiet:T)
```

chars is a CHARACTER vector equivalent to

```
vector("H","e","n","r","y"," "," "," "," ","M","a","l","e"," "," "," ","\
"6","7","."."3","","","1","0",".","5","\n","S","u","s","a","n"," ","\
" "," ","F","e","m","a","l","e"," ","5","9",".","2","","","1","5","\
".","1","\n"," "," "," "," "," ","!","\n","O","t","h","e","r"," ","s","\
"t","u","f","f"," ",".",".",".","\n")
```

7.3 **"Reading" from CHARACTER variables** All commands that read files (except batch() and restore()) can also "read" from a quoted string or CHARACTER variable which is the value of keyword string. Some examples are the following.

```
Cmd> x <- vecread(string:"1 3 2 7 10.11"); x
(1)           1           3           2           7        10.11

Cmd> x <- vecread(string:vector("1 3","2 7","10.11"));x
(1)           1           3           2           7        10.11

Cmd> charx1 <- "x1  1    5
) 1 by 5 matrix with up to 3 numbers per line
)\"%f %f %f \"
2 5 2
7 13.21" # this is a single quoted string, a CHARACTER scalar

Cmd> x1 <- matread(string:charx1,"x1")
x1  1    5                                          Lines echoed
) 1 by 5 matrix with up to 3 numbers per line       by matread()

Cmd> x1 # Here's what was read
(1,1)          2           5           2           7        13.21

Cmd> charx2 <- vector("x  1    5",\
") 1 by 5 matrix with up to 3 numbers per line",\
")\"%f %f %f \"","1 3 1","8 17.1") # this is a CHARACTER vector

Cmd> x2 <- matread(string:charx2,"x")
x  1    5
) 1 by 5 matrix with up to 3 numbers per line

Cmd> x2
(1,1)          1           3           1           8         17.1
```

As two of these examples show, the value of string can be a vector and the last two examples show that you can even read a named matrix from a CHARACTER variable. If the value of string is a vector, every element is read as a single line in a file (or several lines if there are embedded end-of-line characters.)

On a windowed version (Macintosh, Windows, Motif), the value of string can be CLIPBOARD, allowing easy importing of data from other applications. For example, if you select a 20 by 5 array of numbers in a spreadsheet program such as Excel™, and copy it to the Clipboard using **Copy** on the **Edit** menu, you can create a matrix in MacAnova by

```
Cmd> data <- matrix(vecread(string:CLIPBOARD),5)' # note transpose
```

Predefined macro fromclip makes this easier.

```
Cmd> data <- fromclip(5) # argument is number of columns
```

vecread(string:CharVec,bylines:T), where CharVec is a CHARACTER vector, treats each element of CharVec as starting a new line.

```
Cmd> vecread(string:vector("Line 1","Line 2","Line 3"),bylines:T)
(1) "Line 1"
(2) "Line 2"
(3) "Line 3"
```

vecread(string:CharVec,bychars:T) inserts an empty string ("") between each of the elements of charvec.

```
Cmd> vecread(string:vector("ab","mn","yz"),bychar:T)
(1) "a"
(2) "b"
(3) ""
(4) "m"
(5) "n"
(6) ""
(7) "y"
(8) "z"
```

7.3.1 **"Decoding" a CHARACTER variable** Here is an example showing how to use vecread() to extract information from predefined CHARACTER variable VERSION which contains information on the version number and compilation date for the copy of MacAnova you are using.

```
Cmd> VERSION
(1) "MacAnova 4.07 of 08/12/98 (Power Macintosh [CW])"

Cmd> tmp <- vecread(string:VERSION,bywords:T);tmp #see Sec. 7.2
(1) "MacAnova"
(2) "4.07"
(3) "of"
(4) "08/12/98"
(5) "(Power"
(6) "Macintosh"
(7) "[CW])"

Cmd> versionNo <- vecread(string:tmp[2]); versionNo
(1)        4.07

Cmd> date <- vecread(string:tmp[4],silent:T); date
(1)           8          12          98
```

Note the use of silent:T to suppress the warning message that would normally occur because of the /'s in the data. See Sec. 2.11.1.

The ability to read a file as CHARACTER data and then take apart the lines sometimes makes it possible to to read text files that have non-standard formats. For example, suppose file presdent.txt looks like the following:

```
G. Washington  1789 1797 VA  2/22/1732 12/14/1799 Episcopalian
J. Adams       1797 1801 MA 10/30/1795   7/4/1826 Unitarian
T. Jefferson   1801 1809 VA  4/13/1743   7/4/1826 Deist
```

Then you can create variables containing information from specific fields of each line. Here are some examples (see Sec. 9.2.3 for the use of for(...){...}).

```
Cmd> data <- vecread("presdent.txt",bylines:T) #See Sec. 7.2

Cmd> data
(1) "G. Washington  1789 1797 VA  2/22/1732 12/14/1799 Episcopalian"
(2) "J. Adams       1797 1801 MA 10/30/1795   7/4/1826 Unitarian"
(3) "T. Jefferson   1801 1809 VA  4/13/1743   7/4/1826 Deist"
```

```
Cmd> names <- rep("",3) # create big enough variable

Cmd> for(i,run(3)){ names[i] <- \
paste(vecread(string:data[i],bywords:T)[run(2)]);;}

Cmd> # See Sec. 9.2.3 for for(...){...}; Sec. 8.3.1 for paste()

Cmd> names
(1) "G. Washington"
(2) "J. Adams"
(3) "T. Jefferson"

Cmd> terms <- matrix(rep(0,6),3) # create matrix of right size

Cmd> for(i,run(3)){ # Sec. 9.2.3
    @tmp <- vecread(string:data[i],char:T)[run(3,4)]
    terms[i,] <- vecread(string:@tmp)[run(2)]';;}

Cmd> terms
(1,1)         1789         1797
(2,1)         1797         1801
(3,1)         1801         1809

Cmd> birthyear <- rep(0,3) # create vector of right size

Cmd> for(i,run(3)){
    @tmp <- vecread(string:data[i],char:T)[6]
    birthyear[i] <- vecread(string:@tmp,silent:T)[3];;}

Cmd> birthyear
(1)         1732         1795         1743
```

In each case, we start by creating a variable of the right size and shape to hold the data extracted. See Sec. 8.3.1 for the use of paste().

Here is an example where we separate the left and right hand sides of a GLM model (see Sec. 3.4), squeezing out any spaces.

```
Cmd> cmodel <- vecread(string:"y = a + b",bychars:T)

Cmd> if (match("=", cmodel, 0) == 0){# See Sec. 9.2.2, 9.4.1
        print("ERROR: no \"=\" in the model")
    } else {
        for (i,1,length(cmodel)){#See Sec. 9.2.3
            if(cmodel[i] == "="){break}
        }
        leftside <- if (i == 1) {""} else {cmodel[run(i-1)]}
        rhs <- if (i == length(cmodel)) {""} else {cmodel[-run(i)]}
        lhs[lhs == " "] <- ""; lhs <- paste(lhs,sep:"")
        rhs[rhs == " "] <- ""; rhs <- paste(rhs,sep:"")
        print(lhs,rhs) # See Sec. 7.4
    }
lhs:
(1) "y"
rhs:
(1) "a+b"
```

7.3.2 **Finding ASCII codes for a CHARACTER variable – getascii()** In your computer's memory, every character, letter, number or punctuation, of a CHARACTER variable is

represented by a number between 0 and 255, its ASCII code. For certain specialized purposes, you may want to determine the codes correponding to particular characters. That's what getascii() allows you to do. It's probably best explained by examples.

```
Cmd> getascii("ABCDE") # ascii code of 'A' is 65, etc.
(1)          65          66          67          68          69
```

This is the simplest usage, and usually the only one needed. getascii() returns the ASCII codes of the characters in it's CHARACTER scalar argument. 65 is the code for A, 66 is the code for B, and so on. See Sec. 7.4.3 for a table of the ASCII codes for printable characters. getascii("") returns NULL.

When there are several CHARACTER vectors as arguments, they are all pasted together before decoding:

```
Cmd> getascii(vector("AB","C"), "DE") # same as preceding
(1)          65          66          67          68          69
```

You can include in a string any character, even one you cannot type directly, using the so called escaped octal representation of its ASCII code. Thus, since 1×8 + 5 = 13, 15 is the octal (base 8) representation of decimal 13, "\15" or "\015" is the character (usually interpreted as a Return character) with code 13. Similarly, because 123 is the octal representation for $1\times8^2 + 2\times8 + 3 = 83$, the ASCII code for "S", "\123" is equivalent to "S". You can use getascii() to confirm that these codes are generated.

```
Cmd> getascii("\123")
(1)          83

Cmd> getascii("\001\002\003\004\005") #or getascii("\1\2\3\4\5")
(1)           1           2           3           4           5
```

Function putascii() with keyword phrase keep:T (Sec. 7.4.3) can be viewed as an inverse to getascii().

7.4 **Writing data to the screen and to files** There are several commands for writing data in human readable form. These include print(), write(), matprint(), and matwrite(). The first two of these normally write to the screen or output window, while the remainder write plain text or ASCII files. All require one or more MacAnova variables or expressions as arguments and allow you to specify the format in which information is written using keyword phrases.

write() and matwrite() differ from their twins print() and matprint() only in the default format that is used for REAL and LOGICAL data. The default format for print() and matprint() is taken from option format; the default format for write() and matwrite() is taken from option wformat (See Sec. 8.1). These defaults are initialized to "12.5g" and "16.9g", that is, floating point format with 5 and 9 significant digits, respectively. Thus commands ending in print normally provide 5 significant digits, and those ending in write provide 9 significant digits.

matprint(), and matwrite() write to files. Their first argument is a quoted string or a CHARACTER variable specifying the file name. Their default action is to add information to the *end* of an existing file, not disturbing anything already in the file. However, if you include keyword phrase new:T as an argument, writing starts at the *beginning* of

the file, overwriting any information in the file. An example of their use might be

```
Cmd> matwrite("myfile",a,b,logb:log(b),new:T)
```

This also illustrates another use of keywords on all these commands – to specify a name to label a printed item. Without the keyword logb, log(b) would have been given a generic label such as NUMBER or VECTOR.

If you use variable CONSOLE in place of a file name, output will be written to the screen and not to a file. This can be helpful for exploring the format of data sets written by matprint() and matwrite(). The value of CONSOLE is ignored.

You can modify the format used on all output commands using keywords nsig and format.

Keyword phrase	Meaning
nsig:n	Number of significant digits will be positive integer n
format:Format	Format used taken from CHARACTER scalar Format

An argument of either form affects how subsequent arguments are printed until another nsig or format keyword phrase, if any, resets the formatting. Format must be a quoted string or CHARACTER variable. See the discussion of option format in Sec. 8.1.3 for details on the value of Format.

When nsig:n or format:Format *follows* all data argument in the command (for example, print(x,y,nsig:15) or matprint("myfile",x,y, nsig:15)), it is used as if it *preceded* all other arguments (except the file name) (print(nsig:15,x,y) or matprint("myfile", nsig:15,x,y)). See Sec. 7.4.1 for examples of keywords format and nsig.

Keywords nsig and format affect only the current command and have no effect on option settings.

7.4.1 **print() and write()** These both write to the screen or output window and have identical usage, print(var1,var2,...) and write(var1,var2,...), possibly with interspersed nsig and format keyword phrases (see Sec. 7.4). Arguments var1, var2, ... can be arbitrary expressions, variables or macros, even variables of type GRAPH.

```
Cmd> x <- 1000*PI

Cmd> print(x,format:"16.9g",x1:x,format:".4g",y:x,\
    format:"10.3f",z:x)
x:
(1)       3141.6      Default print format 12.5g
x1:
(1)       3141.59265  Format 16.9g
y:
(1)        3142       Format .4g = 11.4g  (11 is 4+7)
z:
(1)   3141.593        Format 10.3f, fixed with 3 decimal places

Cmd> write(x)
x:
(1)       3141.59265  Uses default write format 16.9g
```

There are several keywords besides nsig and format.

Keyword Phrase	Meaning
header:F	Suppress writing data set name as header
labels:F	Suppress writing index labels
name:CharScalar	Use CharScalar instead of variable name in header
missing:CharScalar	Substitute for MISSING for missing values
file:fileName	Write to file fileName instead of screen or window
new:T	Overwrite contents of files (used only with file)

The value for name affects only the next variable printed; the values for header, labels and missing affect all following variables, unless changed by a new value. If name, header, labels or missing follow all variables being printed, they are treated as if they were before them.

If a variable printed is a structure, all its components are printed using any keywords affecting the printing of the variable itself.

Here header:F and labels:F are used together to suppress both the name and the index labels.

```
Cmd> print(sqrt(matrix(run(9),3)),header:F,labels:F)
           1            2       2.6458
      1.4142       2.2361       2.8284
      1.7321       2.4495            3
```

Here is an example of output omitting missing and including it.

```
Cmd> print(vector(1,3,?,2.5,2))
VECTOR:
(1)           1           3     MISSING         2.5           2
Cmd> print(vector(1,3,?,2.5,2),missing:"??")
VECTOR:
(1)           1           3          ??         2.5           2
```

Functions print() and write() normally write to the screen, but can write to a file. Here is how you might write to a file, overwriting anything already there.

```
Cmd> print(sqrt(matrix(run(9),3)),file:"results.txt",new:T)
```

new:T insures that any information in the file is deleted before writing. Without new:T, output is written at the end of the file.

Normally, unless you use header:F, each item printed by print() and write() is preceded by its name. If the item is an expression or the result of a function, a generic name like MATRIX is used. As illustrated above, you can replace these names by specifying the item in a keyword phrase.

```
Cmd> print(sqrtpi:sqrt(PI))
sqrtpi:
(1)       1.7725
```

Such a name must be a legal keyword name, that is, up to 10 characters starting with a letter and containing only letters, digits or "_". You can provide longer and more descriptive names using keyword name.

```
Cmd> print(name:"Square root of pi",sqrt(PI),name:"Log(pi)",log(PI))
Square root of pi:
(1)       1.7725
Log(pi):
(1)       1.1447
```

You may use name several times, but each value is used just once for the next item being printed (or the first, if name:Name follows all the items being printed).

Because of the header line, data written to a file by print(...,file:fileName) and write(...,file:fileName) normally cannot be reread correctly by MacAnova. However, if x is a REAL variable,

```
Cmd>print(x,header:F,labels:F,file:fileName,new:T) # or write()
```

creates a file with no header or row labels that can be read by vecread() as well as by many programs that can read data from plain text or ASCII files.

7.4.2 **matprint() and matwrite()** These write their arguments to a file in a form that can be read by matread() and read(). See Sec. 7.1 and 7.1.1-7.1.3 for a description of the file format. The standard usages are identical:

```
Cmd>matprint(fileName,arg1,arg2,...)
```

and

```
Cmd>matwrite(fileName,arg1,arg2,...)
```

The arguments arg1, arg2, ... to be written can be of any type, including CHARACTER, NULL and structures. Each will be written as a named data set with the same name as the argument. You can specify a name different from the variable name using a keyword. For example,

```
Cmd> x <- run(5); y <- vector(1,2,3,5,8)
Cmd> matwrite("mydata.txt",sqrtx:sqrt(x'),logy:log(y'),\
    format:"8.6f")
```

creates the file mydata.txt as listed here.

```
sqrtx             1     5
)"%lf %lf %lf %lf %lf"
 1.000000 1.414214 1.732051 2.000000 2.236068
logy              1     5
)"%lf %lf %lf %lf %lf"
 0.000000 0.693147 1.098612 1.609438 2.079442
```

You can use the following keywords in addition to nsig and format.

Keyword Phrase	Meaning
header:F	Suppress first header line of data set
name:CharScalar	Use CharScalar instead of data set name in header
comments:CharVector	Elements of CharVector written as comments
missing:RealScalar	Value to be written for missing values
sep:SingleChar	Separator between values in line
new:T	Overwrite contents of file

When new:T is an argument, any data currently in the file will be discarded before writing.

The values for name and comments affect only the next following variable being written. The value for missing applies to all following variables, unless replaced by a new instance of missing.

Unless you supply an alternative value using keyword missing, any MISSING values in a variable will be printed exactly as -99999.9999, regardless of the format, and a comment line of the form ")MISSING -99999.9999" will be written to the file before the data. When matread() or read() read the variable you have written, this value or the value specified by missing will be translated back to MISSING. Note that this usage is different from the use of missing in print() and write() (Sec. 7.4.1). On those commands you use missing to specify a CHARACTER code such as "NA" or "?" for MISSING data; on matprint() and matwrite() you use missing to specify a REAL code for MISSING.

In the following examples, the file name is given as CONSOLE, resulting in writing the the screen or output window.

```
Cmd> y <- sqrt(matrix(vector(11,10,6,9,12,14,12,4,8,11,?,3),3))
WARNING: missing values in argument(s) to sqrt()

Cmd> matprint(CONSOLE,missing:-1,y) # print to screen (CONSOLE)
y                3     4
) MISSING:           -1    Because of missing:-1
)"%lf %lf %lf %lf"
       3.3166            3       3.4641       3.3166
       3.1623       3.4641            2           -1
       2.4495       3.7417       2.8284       1.7321

Cmd> matwrite(CONSOLE,y,format:"9.5f",missing:-99,ysq:y^2)
WARNING: arithmetic with missing value(s); operation is ^
y                3     4      Default value of missing applies
) MISSING: -99999.9999        to first matrix printed
)"%lf %lf %lf %lf"
       3.31662479               3      3.46410162       3.31662479
       3.16227766       3.46410162               2 -99999.9999
       2.44948974       3.74165739       2.82842712       1.73205081
```

```
ysq              3     4
) MISSING: -99.00000          Specified value of missing
)"%lf %lf %lf %lf"            applies to second printed
  11.00000   9.00000  12.00000  11.00000
  10.00000  12.00000   4.00000 -99.00000
   6.00000  14.00000   8.00000   3.00000
```

Here is an example of the use of header:F.

```
Cmd> matprint(CONSOLE,missing:-1,y,header:F)
       3.3166            3       3.4641       3.3166
       3.1623       3.4641            2           -1
       2.4495       3.7417       2.8284       1.7321
```

Any program that can read data items separated by spaces will be able to read a file that looks like this. When writing to a file rather than to CONSOLE, you would normally use new:T to ensure that nothing precedes the data in the file.

Keyword sep allows you to write files that can be read by programs, such as some spreadsheets, that expect data items to be separated by commas or some other character such as tab. The value of sep should be a single quoted character, for example "," or "\t" (tab).

```
Cmd> matprint(CONSOLE,missing:-1,y,sep:",") # comma-separated
3.3166,3,3.4641,3.3166
3.1623,3.4641,2,-1
2.4495,3.7417,2.8284,1.7321

Cmd> matprint(CONSOLE,missing:-1,y,sep:"\t") # Tab-separated
3.3166  3       3.4641  3.3166
3.1623  3.4641  2       -1
2.4495  3.7417  2.8284  1.7321
```

As you can see, the use of sep automatically suppresses the header. If for some reason you want a header, use header:T.

In a Windowed version (Macintosh, Windows, Motif), if you write data with sep:"\t" to the output window by specifying CONSOLE as file name, and then Copy it to the to the Clipboard using **Copy** on the **Edit** menu, it may be possible to paste it directly into a spreadsheet.

When you don't use header:F as an argument, keyword name allows you to specify longer and non-standard names to be put in the header.

```
Cmd> matprint(CONSOLE,name:"sqrt(data)",y,missing:-99)
sqrt(data)       3     4
) MISSING:           -99
)"%lf %lf %lf %lf"
       3.3166            3       3.4641       3.3166
       3.1623       3.4641            2          -99
       2.4495       3.7417       2.8284       1.7321
```

Any CHARACTER scalar or quoted string can be used for the name. However, if it contains any spaces or starts with any character other than a letter, "_" or "@", you won't be able to read it back in using matread(). Non-alphabetic and non-numeric characters after the first are OK. You may use name several times, but each value is used only once

for the next item being printed (or the first, if name:Name follows all the items being printed).

If you are using matprint() or matwrite() to create a library of data sets, it is helpful to be able to include descriptive comment lines. This can be done by including keyword comments whose value must be a CHARACTER scalar or vector. Here is an example.

```
Cmd> matprint(CONSOLE,y,missing:-99,\
    comments:vector("Sample data","which includes 1 MISSING value"))
y                3     4
) Sample data
) which includes 1 MISSING value
) MISSING:          -99
)"%lf %lf %lf %lf"
       3.3166            3       3.4641       3.3166
       3.1623       3.4641            2          -99
       2.4495       3.7417       2.8284       1.7321
```

Here the value of comments is a vector of length 2, each element of which, preceded by "`) `", becomes a line. The line giving a value for MISSING was added by matprint().

7.4.3 **putascii()** Command putascii() is rather specialized. It allows you to output arbitrary symbols, even ones that are not normally printable. Its original purpose was to allow you to do such things as clear the screen or ring the bell on a terminal but if you are creative you may find other uses for it.

putascii(codes) where codes is a vector of integers between 1 and 255 "emits" length(codes) characters specified by codes[1], codes[2], ... , interpreted as ASCII codes (see Sec. 7.3.2).

putascii(code1,code2,...), where the arguments are integer scalars or vectors is equivalent to putascii(vector(code1,code2,...)).

Code 7 is usually an audible signal like a bell or a beep, code 9 is the tab character ("\t"), code 10 (13 on a Macintosh) is the new line character ("\n") and code 32 is a space. Codes between 33 and 126 are the following normal visible printable characters.

Codes	Characters
33 to 63	! " # $ % & ' ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ?
64 to 95	@ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [\ ] ^ _
96 to 126	` a b c d e f g h i j k l m n o p q r s t u v w x y z { | } ~

```
Cmd> putascii(7,7,7) # rings bell or "beeps" 3 times

Cmd> putascii(run(65,90)) # uppercase alphabet
ABCDEFGHIJKLMNOPQRSTUVWXYZ

Cmd> putascii(77,97,99,65,110,111,118,97)
MacAnova
```

In a non-Motif Unix version, putascii() may be helpful in switching between different terminal emulations. For example, when running in an xterm window, putascii(vector(29,27,56)) switches to Tektronix 4014 emulation mode and

`putascii(vector(27,50))` switches back to VT100 emulation mode.

See Sec. 8.3.4 for the use of `putascii()` to create a `CHARACTER` variable.

`putascii(x,keep:T)` can be viewed as an inverse to `getascii(c)`, in the sense that `putascii(getascii(c),keep:T)` returns `c` and `getascii(putascii(x,keep:T))` returns `x`, where `c` is a `CHARACTER` scalar and `x` is a `REAL` vector of integers between 1 and 255, inclusive.

```
Cmd> getascii(putascii(run(30,34),keep:T))
(1)          30          31          32          33          34

Cmd> putascii(getascii("MacAnova"),keep:T)
(1) "MacAnova"
```

7.5 **Macro files** A powerful feature of MacAnova is its ability to use libraries of macros in files. You can retrieve macros from a file using command `macroread()` and predefined macro `getmacros`. You can use `macrowrite()` to write macros to a file in a form readable by `macroread()` and `getmacros`. Files `MacAnova.mac`, `Tser.mac` and `Design.mac` distributed with MacAnova are libraries of macros that extend the power of MacAnova. See Sec. 9.3 for details on what goes on in a macro.

When `macroread()` and `getmacros` do not find a specific macro file in the default directory or folder, they look for it in the directories or folders whose names are in `CHARACTER` vector `DATAPATHS`. See Sec. 2.11.6.

7.5.1 **macroread(), read() and the format of macro files** Commands `macroread()` and `read()` read macros from a file in a standard format. Here `macroread()` is used to retrieve macro `covar` from file `macanova.mac`.

```
Cmd> compcovar <- macroread("macanova.mac","covar")
covar              12 MACRO
) usage: d <- covar(x), x a matrix, computes structure with components
) n (scalar), mean (row vector), and covariance (matrix)
```

This reads `covar` from file `macanova.mac` and saves it in the workspace under the name `compcovar`. The lines printed are header lines for the macro in the file. Lines starting with "`)`" usually contain descriptive or usage information. They are normally echoed unless they start with "`))`" or you use keyword phrase `quiet:T`.

```
Cmd> compcovar <- macroread("macanova.mac","covar",quiet:T)
```

does the same, without printing these lines. To force echoing of lines starting with "`))`", use keyword phrase `quiet:F`.

Each macro that `macroread()` can read from a file must be in one of two similar forms. They are best illustrated by examples. Here is a listing of a file containing two macros, `fences` and `standardize`, one in each form (see Sec. 9.3 for an explanation of symbols such as `$1`, `$S` and `@x$$`):

```
fences        8 MACRO
) macro with line count on the name line
) usage: d <- fences(x), compute inner and outer
)    fences of vector x
)) This is a comment that will not be echoed unless quiet:F is used
)) Such lines can be used to include extensive documentation that
)) it would not be desirable to echo whenever read.
# usage: d <- $S(x)
@x$$ <- $1
if (!isvector(@x$$) || !isreal(@x$$)){
    error("ERROR: $1 is not a REAL vector")}
@stats$$ <- describe($1, q1:T, q3:T)
@iqr$$ <- @stats$$$q3 - @stats$$$q1
vector(@stats$$$q1 - vector(3, 1.5)*@iqr$$,\
   @stats$$$q3 + run(1.5, 3)*@iqr$$)

standardize MACRO OUTOFLINE
) macro with no line count but with special line to terminate it
) usage: d <- standardize(x)
# usage: d <- $S(x)
@x$$ <- $1
@dims$$ <- dim(@x$$)
@dims$$[1] <- 1
@mean$$ <- array(describe(@x$$,mean:T),@dims$$)
@sd$$ <- sqrt(array(describe(@x$$,var:T),@dims$$))
(@x$$ - @mean$$)/@sd$$
%standardize%
```

Macro `fences` is in a format for which the first header line specifies its length in lines. The first line, the name line, contains the name of the macro at the start of the line, followed by the number of lines of commands in the macro and keyword `MACRO`. *The number of lines does not include the name line and lines starting with ")".*

The *absence* of a line count on the name line for `standardize` signals that its end will be indicated by a line starting `%standardize%`, that is, the name of the macro preceded and followed by "%". In this format, too, keyword `MACRO` is required.

Keyword `OUTOFLINE` on the name line specifies that the macro is to be expanded "out of line", rather than "in line" (see Sec. 9.3).

The name line can also contain `NOTES`, just as with data sets readable by `matread()` and `read()` (see Sec. 7.1 and 7.1.4).

In searching the file for a line starting with the macro name and containing "`MACRO`", MacAnova ignores case, so , for example, the first line of `fences` could be replaced by

```
Fences        8 MACRO
```

Lines starting with "`)`" immediately after the name line are *comment lines* (there are three in `fences`) and are part of the header and not part of the macro. After these is the text of the macro itself, consisting of MacAnova commands, possibly including lines starting with "#". If the number of lines was specified on the name line as it was for `fences`, there must be exactly that many lines. Quotation marks do not need to be preceded by "\" and indeed should not be, unless they themselves are part of a quoted string as in `print("\"Hello\"")`.

You can have any number of macros in a file, as long as each has one of these forms. If you do not provide a macro name to `macroread()` (for example, `macroread("myfile.mac")`), the first macro in the file is read. Macros can even be interspersed with data sets of the form readable by `matread()` (See Sec. 7.1).

It is not an error for a macro on a file to have 0 lines. In that case `macroread()` prints the header lines and returns `NULL`. It is a useful idea to have the first macro in a file have zero lines but with comment lines that list the actual macros available in the file. Then you can obtain a brief catalogue of what's in the file by `macroread(fileName)`, with no data set specified.

Here are keyword phrases that can be used as additional arguments to `macroread()`.

Keyword Phrase	Meaning
`quiet:T`	Header and descriptive comments will not be printed
`quiet:F`	*All* header and descriptive comments will be printed
`echo:T`	Macro will be printed as it is read
`silent:T`	Print only error messages; incompatible with `quiet:F` or `echo:T`
`notfoundok:T`	Failure to find macro is not an error

When `notfoundok:T` is an argument and the wanted macro is not found, no error message is printed and the value returned by `macroread()` is `NULL`.

7.5.2 **macrowrite()** You can write macros to a file in a format that `macroread()` can read using command `macrowrite()`. Its usage is `macrowrite(fileName,macro1, macro2,...)`, where `fileName` is a `CHARACTER` variable or quoted string and `macro1`, `macro2`, ... are macros to be written. If the file already exists, the default behavior is to write the macros at the end, not changing anything already in the file. If the keyword phrase `new:T` is an argument, any information in the file is discarded and the macros are written at the beginning.

```
Cmd> macrowrite(CONSOLE,colplot,rowplot)
colplot       MACRO
#$S(y [,title:"Title of your choice]")
if($N<1){error("ERROR: $S needs at least 1 argument")}
chplot(1,$01,lines:T,$K,xlab:"Row Number")
%colplot%

rowplot       MACRO
#$S(y [,title:"Title of your choice]")
if($N<1){error("ERROR: $S needs at least 1 argument")}
chplot(1,y:($01)',lines:T,$K,xlab:"Column Number")
%rowplot%
```

writes pre-defined macros `colplot` and `rowplot` to the screen because `fileName` is `CONSOLE`. If, say, `"Myfile.mac"` were substituted for `CONSOLE`, the macros would be written to file `Myfile.mac`, in which case you would probably want to include `new:T` as an argument so as to make a new start on the file. If you prefer to write macros in the older style, with a line count in the header line, use keyword phrase `oldstyle:T`.

```
Cmd> macrowrite(CONSOLE,colplot,oldstyle:T)
colplot          3 MACRO
#$S(y [,title:"Title of your choice]")
if($N<1){error("ERROR: $S needs at least 1 argument")}
chplot(1,$01,lines:T,$K,xlab:"Row Number")
```

You can use a keyword to provide a different macro name to be used in the file.

```
Cmd> macrowrite("trnsform.mac",transform:mymacro)
```

writes mymacro to file trnsform.mac giving it the name transform.

You can also use the following keywords. Their usage is identical to that in matprint() and matwrite() (Sec. 7.4.2):

Keyword Phrase	Meaning
header:F	Suppress first header line of macro
name:CharScalar	Use CharScalar instead of macro name in header
comments:CharVector	Elements of CharVector written as comments
new:T	Overwrite contents of file

7.5.3 **getmacros** Predefined macro getmacros is a shortcut that uses macroread() to retrieve one or more macros from the files whose names are in CHARACTER vector MACROFILES. The default value of MACROFILES is vector("macanova.mac", "tser.mac","design.mac"). If MACROFILES does not exist, the file whose name is in CHARACTER scalar MACROFILE is read.

```
Cmd> MACROFILES # default value
(1) "macanova.mac"
(2) "tser.mac"
(3) "design.mac"

Cmd> getmacros(groupcovar,detrend)
groupcovar          15 MACRO
) groupcovar(groups,y), N by 1 vector groups, N by p matrix y
detrend          22 MACRO
) usage: detrend(x [,degree]), remove polynomial trend from cols of x
```

Here groupcovar and detrend were read from files macanova.mac and tser.mac, respectively.

```
Cmd> getmacros(groupcovar,detrend,quiet:T)
```

does the same, suppressing the echoing of the header lines.

7.5.4 **addmacrofile** As mentioned in Sec. 7.5.3, getmacros() scans the files whose names are in CHARACTER vector MACROFILES When MacAnova is launched, this is pre-defined to be vector("macanova.mac","tser.mac","design.mac"), containing the names of the macro files distributed with MacAnova. Since MACROFILES is just an ordinary variable, you can modify it or re-define it in any way that you find convenient.

Pre-defined macro addmacrofile allows you easily to add the name of a file to MACROFILES so that the file will also be searched by getmacros.

addmacrofile(FileName) adds the name to the *start* of MACROFILES and addmacrofile(FileName,T) adds the name at the *end*. If MACROFILES does not already exist, addmacrofile(FileName) creates it. FileName must be a quoted string or CHARACTER scalar.

```
Cmd> MACROFILES # here is the current default list
(1) "macanova.mac"
(2) "tser.mac"
(3) "design.mac"

Cmd> addmacrofile("survival.mac") # add file name at start

Cmd> MACROFILES # updated list
(1) "survival.mac"
(2) "macanova.mac"
(3) "tser.mac"
(4) "design.mac"

Cmd> addmacrofile("multivar.mac",T) # add file name at end

Cmd> MACROFILES # updated list
(1) "survival.mac"
(2) "macanova.mac"
(3) "tser.mac"
(4) "design.mac"
(5) "multivar.mac"
```

With the final form of MACROFILES, getmacros(macroname) will search up to five files, starting with survival.mac and ending with multivar.mac. If a macro with name macroname is on more than one file, the first one found will be read. It is clearly advantageous to have the file you will reference most often at the head of the list.

You can use pre-defined macro adddatapath to add to CHARACTER variable DATAPATHS the name of a directory or folder where macroread() and getmacros will look for macro files. See Sec. 2.11.6.

7.6 **Executing commands in a file – batch()** Command batch() is a way to execute many commands together. Its usage is batch(fileName), where fileName is a quoted string or CHARACTER variable that specifies the name of a file containing a sequence of MacAnova commands. The file must be a text (ASCII) file and may contain any MacAnova commands, including other batch() commands. As usual, on a windowed version (Macintosh, Windows, Motif), if fileName is the null file name "", you can select the file using a dialog box.

The commands in the file are read line by line and executed exactly as if they were typed on the keyboard, with the output being written to the screen. In addition, the commands in the file are echoed to the screen, preceded by a "prompt" constructed from the name of the file.

You cannot use batch() in a macro or in a for or while loop (Sec. 9.2.3). It is an error if any commands follow batch() on the same line or in the same compound command.

Suppose file mybatch.txt contains the following four lines:

```
# Sample batch file of MacAnova commands
data <- matread("macanova.dat","halddata",quiet:T)
makecols(data,x1,x2,x3,x4,y)
regress("y=x1+x2+x3+x4") # regress column 5 on 1st 4
```

Here is output produced when `mybatch.txt` is executed using `batch()`:

```
Cmd> batch("mybatch.txt") # initiate executing commands in file

mybatch.txt> data <- matread("macanova.dat","halddata",quiet:T)

mybatch.txt> makecols(data,x1,x2,x3,x4,y)

mybatch.txt> regress("y=x1+x2+x3+x4") # regress column 5 on 1st 4
Model used is y=x1+x2+x3+x4
                Coef       StdErr            t
CONSTANT       62.405       70.071       0.8906
x1             1.5511      0.74477       2.0827
x2            0.51017      0.72379      0.70486
x3            0.10191      0.75471      0.13503
x4           -0.14406      0.70905     -0.20317

N: 13,  MSE: 5.983, DF: 8,  R^2: 0.98238
Regression F(4,8): 111.48, Durbin-Watson: 2.0526
To see the ANOVA table type 'anova()'

mybatch.txt> (end of file on mybatch.txt)

Cmd> # continue with input at the prompt level
```

The echoed commands are not in *italics* to emphasize that they are printed by MacAnova and not typed in.

You can suppress the echoing of commands by `batch(fileName,echo:F)` or by `setoptions(batchecho:F)` (see Sec. 8.1.3):

```
Cmd> batch("mybatch.txt",echo:F) # echoing of commands suppressed
Model used is y=x1+x2+x3+x4
                Coef       StdErr            t
CONSTANT       62.405       70.071       0.8906
x1             1.5511      0.74477       2.0827
x2            0.51017      0.72379      0.70486
x3            0.10191      0.75471      0.13503
x4           -0.14406      0.70905     -0.20317

N: 13,  MSE: 5.983, DF: 8,  R^2: 0.98238
Regression F(4,8): 111.48, Durbin-Watson: 2.0526
To see the ANOVA table type 'anova()'
```

You can temporarily change the default prompt using keyword `prompt`:

```
Cmd> batch("testfiles/mybatch.txt",prompt:"What? ")

What? # Sample batch file of MacAnova commands

What? data <- matread("macanova.dat","halddata",quiet:T)

What? makecols(data,x1,x2,x3,x4,y)
*****  Interrupt  *****    Output terminated by interrupt
```

When launching MacAnova under Unix , DOS or Windows, when `-b fileName`

appears on the command line, MacAnova will first do the equivalent of `batch("fileName")` before the first regular prompt (see Appendices C, D, E and F). If, in addition, `-bprompt "What? "`, the command executed will be `batch("fileName", prompt:"What? ")`. Alternatively, under Unix and DOS when MacAnova is started up by `macanova < fileName`, commands will be read from file `fileName` until the end of the file, at which point MacAnova will terminate. The output will be printed on the screen or can be redirected to a file by `macanova < fileName > output.txt`. You can specify the prompt printed by, say, `prompt "Next? "` on the command line.

When launching MacAnova on a Macintosh, if you hold down ⌘ you will be given an opportunity to specify a batch file to execute and a file to which output will be written. See Appendix B.

7.7 **Additional options on save() and restore()** Commands `save()` and `asciisave()` (Sec. 2.17) normally save your *entire* workspace. If you want to save only a few variables and macros you can just include the items to be saved as arguments to `save()` or `asciisave()`. For example,

```
Cmd> save("variables.sav",names,terms,birthyear)#or asciisave(...)
```

saves only the variables `names`, `terms` and `birthyear` (see Sec. 7.3). This is sometimes called a *partial save*. You can save a variable under a different name using a keyword phrase. For example, `save(saveFile, height:x)` will save variable `x` in such a way that it will be restored as variable `height`.

Ordinarily when you do a complete save, the current values of all the options (Sec. 8.1.3), including the random number seeds (Sec. 2.13.1) are also saved to be later restored by `restore()`. If you don't want them saved, use `options:F` on `save()` or `asciisave()`.

```
Cmd> save("savefile.sav", options:F)
```

Most versions of MacAnova maintain a command *history*, an internal `CHARACTER` vector containing some number of recent command lines. See Sec. 8.8.2 and 8.8.3. These can be recalled and re-executed, possibly after editing. When you do a complete save, these lines are automatically saved together with the workspace. When the file is restored, these commands normally replace the current history. When you use keyword phrase `history:F` on `save()` or `asciisave()`, the history of commands is not saved. When you use `history:F` on `restore()`, any history of commands in the file are ignored.

If you find this feature inconvenient or confusing, you can suppress it by setting options `savehistry` to False by `setoptions(savehistry:F)`. See Sec. 8.1.3. If you do this, keyword phrase `history:T` will force saving the command history.

There are some things that `save()` and `asciisave()` do *not* normally save. After executing a GLM command (see Chapters 3 and 4), MacAnova preserves a lot of information that is not put in side effects variables such as `RESIDUALS` (Sec. 3.7). This information is used by commands such as `secoefs()` (Sec. 3.13) and `regpred()` (Sec. 3.18) and can be retrieved by `modelvars()` (Sec. 3.24.1) and `modelinfo()` (Sec. 3.24.4). After using `restore()` to recover your workspace, you can recreate this information by

repeating the linear model command, perhaps as simply as typing anova() or regress() which will use the restored value of STRMODEL. However, if the model was large and the computation took a long time, you may not want to repeat the computation. In that case, you can use keyword phrase all:T on a save() or asciisave() command to save the additional additional linear model related information along the workspace in the file.

```
Cmd> save("workspac.sav",all:T) # or asciisave("workspac.asc",all:T)
```

A subsequent use of restore() restores all this information so that functions liek coefs() and modelinfo() work.

```
Cmd> restore("workspac.sav")
Restoring workspace from file workspac.sav
Workspace saved Sat Aug  8 11:25:33 1998

Cmd> coefs()$x2 # works because all:T used when saving
(1)      0.51017
```

This option should be used with caution, because the inclusion of the additional information can make for a very large file.

When restoring from a complete save file, or any save file produced by a version earlier than 4.07, restore() normally deletes all the variables in the current workspace before restoring variables from a file. Sometimes, this may not be what you want. You can suppress this behavior by keyword phrase delete:F.

```
Cmd> delete(birthyear) # birthyear was saved on variables.sav

Cmd> restore("variables.sav", delete:F)
Restoring workspace from file variables.sav
Workspace saved Sat Feb  8 09:48:21 1997

Cmd> list(birthyear,y) # birthyear restored, y still there
birthyear       REAL   3
y               REAL   13
```

Nothing is deleted when delete:F is an argument, although any variables with the same names as those in the file are replaced. delete:F is not necessary when restoring from a partial save file produced by MacAnova 4.07 or later. You can use delete:T if you really want to delete existing variables.

Some users prefer the delete:F behavior be the default on restore(). This can be accomplished by setting option restoredel to False. See Sec. 8.1.3.

```
Cmd> getoptions(restoredel:T) # default value of restoredel is T
(1) T

Cmd> setoptions(restoredel:F) # change it to F

Cmd> restore("variables.sav") # note absence of delete:F
Restoring workspace from file variables.sav
Workspace saved Sat Feb  8 09:48:21 1997

Cmd> list(y) # y not deleted
y               REAL   13
```

Keyword phrase list:T is another option that is sometimes helpful. This causes restore() to report what it is restoring.

```
Cmd> restore("variables.sav", delete:F,list:T)
Restoring workspace from file variables.sav
Restoring names           CHAR   3
Restoring terms           REAL   3     2
Restoring birthyear       REAL   3
Restoring option values
Workspace saved Sat Feb  8 09:48:21 1997
```

As MacAnova has evolved, the format of both save and asciisave files has changed. However, previous formats are still recognized and can be restored. Moreover, although is is hard to see why you would want to, you can use keyword phrases to direct save() or asciisave() to create files that earlier MacAnova versions can restore. Versions 2.4x and older can restore asciisave files saved with v24:T; versions 3.0 and 3.1x can restore files saved using v31:T; version 3.35 can restore files saved using v335:T and versions 4.0 through 4.06 can restore files saved using v406:T.

7.8 **Customizing MacAnova** There are many default variables and options in MacAnova. Because these may not always be optimal for your use MacAnova, there are ways you can change some of them. Specifically, you can provide a special "start up" file and, in all versions except Macintosh, define the environmental variable MACANOVA.

7.8.1 **Using a start up file** As you get to know MacAnova better you may find some defaults, for example, the number of significant digits in the output, are not what you prefer. Or you may develop some favorite macros that you almost always need. You can always take a few minutes when you start up to set things up to your liking. Having a customized *start up file* allows you to do this automatically.

The first thing MacAnova does when it starts up is to look for a start up file containing MacAnova commands. If the file is found, MacAnova simulates a batch() command with echo:F as second argument (see Sec. 7.6), silently executing all the commands in the file. Under DOS, Windows or Windows 95 and on a Macintosh the start up file must be named MacAnova.ini and must be located in the directory or folder where the executable or application file (MACANOVA.EXE or MacAnova) is located. On Unix, including the Motif version, the start up file must have name .macanova.ini (the leading period is important) and must be located in your home directory.

Typically a start up file defines some macros, sets some options, and perhaps deletes some predefined macros or variables that you don't want. It is also a good place to redefine certain standard CHARACTER variables such as DATAFILE and MACROFILE which are used by macros getdata and getmacros. Here is a possible start up file.

```
# start up file for MacAnova
setoptions(nsig:4, angles:"cycles") #(see Sec. 8.1.3)

# add additional standard macro file
addmacrofile("survival.mac",T) # (see Sec. 7.5.4)

# macros defining Unix-like aliases for certain functions
rm        <- macro("delete($0)") # (see Sec. 9.3.1)
ls        <- macro("listbrief($0)")
ll        <- macro("list($0)")
```

```
# macro to print the first few lines of a vector or matrix
head      <- macro("#head(x [,nlines])
@y$$ <- $1
@n$$ <- dim(@y$$)[1]
@y$$[run(min(@n$$,{if($N > 1){$2}else{10}})),]")

# macro to print the last few lines of a vector or matrix
tail      <- macro("#tail(x[,nlines])
@y$$ <- $1
@n$$ <- dim(@y$$)[1]
@y$$[run(max(@n$$-{if($N>1){$2}else{10}}+1,1),@n$$),]")

# macro to make it easy to reset default output format
setformat <- macro("setoptions(format:\"$2$1\")")
```

This file changes two options, adds a file to the list of files `getmacros` searches, and defines six macros. See Sec. 9.3 for information on writing macros.

The start up file is not read if MacAnova starts up by restoring a `save` or `asciisave` file. On DOS, Windows or Unix this happens when the command line includes "`-r savefile`". On a MacIntosh this happens when MacAnova is started up by double clicking on a `save` or `asciisave` file.

7.8.2 **Environmental variable MACANOVA** In all versions for which you can use command line arguments (all except the Macintosh), MacAnova recognizes and uses the value of an environmental variable `MACANOVA`.

The value of `MACANOVA` should be a list of command line options, for example `-l 26 -w 75 -q`. These options are scanned by MacAnova *before* the command line options and thus are overridden by any options on the command line. By setting `MACANOVA` appropriately, you can change the default values of several pre-defined variables and options. See Appendices C, D, E and F for full information about command line arguments.

To use this feature on DOS/Windows/Windows 95 computers, you need to put a line like the following in your `AUTOEXEC.BAT` file.

```
SET MACANOVA=-l 26 -q -dpath c:\macanova\macros
```

This specifies a screen height of 26 lines (`-l 26`), that you don't want to see the "banner" at start up (`-q`) and that `"c:\\macanova\\macros"` should be added to `DATAPATHS`. Include only options or file or path names whose defaults you want to change.

On Unix, if your shell is `csh` or a variant such as `tcsh`, you should put a line similar to the following in file `.cshrc` in your home directory:

```
setenv MACANOVA '-l 26 -w 75 -mpath ~/macanova/macros'
```

If your Unix shell is `sh` or a variant such as `ksh`, you should put a line similar to the following in file `.profile` in your home directory:

```
MACANOVA='-l 26 -w 75 -mpath ~/macanova/macros';export MACANOVA
```

One purpose of this option is to make it easier to use a Unix binary executable file on a computer configured differently from the one for which it was compiled. By including

```
-help helpFile -mpath macroPath -dpath dataPath
```

in environmental variable MACANOVA, where helpFile includes the complete path name (directory and file name) for the help file, and macroPath and dataPath are the complete path names for directories where macro and data files are kept, all installation dependent information is suppressed. It's o.k. for macroPath and dataPath to be the same.