Center for Tropical Forest Science R Package Manual

Pamela Hall, Suzanne Lao, Ellen Connell and Marie Massa

Version 1.00 March 29, 2006

6.0 Reading and Writing Datafiles

Data can be entered using the keyboard (short datasets or test files) or with the CTFS read and write data functions. To use the read and write functions in R, a data file created in data management software or a spreadsheet must be converted to a tab delimited text file without quotes. Include the column headings (variable names) as the first line of the file. Use database and spreadsheet software that provide this option, either as part of the "Save As" or "Export".

One concept to keep in mind is that data files exist external to the R environment, that is they exist on the harddrive when R isn't running. To address these files in R, you have to designate the location of the file by providing a path or designating the working directory for R to be in a specific folder where the files are AND you have to provide the name of the file. Since the file is outside the R environment, the file name is consider a character string and must always appear in quotes and as the entire file name.

There are several ways to read large text files into the R environment. In addition to the information provided here, please read the R help pages for each function described below and for the CTFS specific functions.

The related CTFS R help pages for reading and writing CTFS files is : *CTFS.readwrite*. Take a look at it while reading this manual chapter. *CTFS.datafiles* provides a full description of the structure of all of the CTFS datafiles.

6.1 Reading Datafiles

The named files to be read into R must be located by an explicit **path** name or be in the working directory of the current R session. In R, use *setwd()* to make the working directory the one containing the data file to be read in or give the ENTIRE path and file name for each function. The following examples assume that *setwd()* was used and the working directory for the R session is the directory in which the data files are located. Since reading a data file from disk into an R session only has to be done once, then it is easier to just temporarily change directories. Once an R data file structure has been created and that file has been saved to disk, then the Data Manager menu can be used to read that file into a current R session.

scan()

reads data from a text file into a vector or a list of vectors directly from the console. The general syntax of scan() is:

```
>scan(file , what , sep , quote , skip)
```

file : name of the input file to read data from. The path should be included if it is not the default path, i.e. the working directory. If the file is specified as """, which is an empty name, then input is taken from the keyboard.
what : specifies the type of data to be read: logical, integer, numeric, character, or list.
sep : specifies what the field separator is, which by default, is a blank space between 2 fields as known as "white space". Other common separators are tab ("\t") and commas (",")
quote : specifies the set of quoting characters used to define where one character string begins and ends.
skip : designates the number of lines of the input file to skip before beginning to read the data values, as in the case of column names, which you would skip.

Scan returns a list of vectors with the types given by the types of the elements in *'what'*. This provides a way of reading columnar data. For example:

```
scancensdata=function(alsebl.txt)
{
 data.read=scan(censdata,skip=1,what=list("","","","","","",""))
  tag=as.numeric(data.read[[1]])
  dbh=as.numeric(data.read[[2]])
  status=data.read[[3]]
  pom=as.numeric(data.read[[4]])
  stems=as.numeric(data.read[[5]])
  date=as.numeric(data.read[[6]])
  codes=data.read[[7]]

return(data.frame(tag=tag,dbh=dbh,status=status,pom=pom,stems=stems,
date=date,codes=codes))
}
```

This function reads in your text *datafile* which is called *alsebl.txt* on the disk into a file in the R session called *censdata*. The file contains the fields *tag, dbh, status, pom, number of stems, date,* and *codes*. The specified parameters for this function skip the first row containing the names of the columns, and specifies that the data be read in as a list of character vectors. The vectors are converted to numeric when applicable and a data.frame is returned as output. Note that the column names as provided in the text file are not included in the output except with the explicit inclusion of the column names in the *data.frame* statement.

Here is a datafile called *alsebl.txt* that can be made into a dataset accessible by R that can be read in using *scan()*. This file is white space delimited.

tag	dbh	status	pom	stems	date	codes
000047	426	A	3	1	5382	B
000049	228	A	1	1	5396	*
000068	277	A	1	1	5390	*

```
000071     318   A   1   1    5390   *
000073     368   A   2   1    5382   B
000089     580   A   2   1    5382   B
000092     -1    D   0   -1   5387   *
000109     318   A   2   1    5382   B
000122     351   A   2   1    5377   B
000138     246   A   2   1    5377   B
000169     404   A   2   1    5377   B
000172     308   A   2   1    5377   BQ
```

Using the above function, *scanscensdata()* to create a data frame Rdata file. Note that *alsebl.txt* is in quotes because it is the name of a file on the disk.

```
> alsebl=scancensdata("alsebl.txt")
> alsebl[1:5,]

  tag dbh status pom stems date codes
1  47 426    A    3    1 5382    B
2  49 228    A    1    1 5396    *
3  68 277    A    1    1 5390    *
4  71 318    A    1    1 5390    *
5  73 368    A    2    1 5382    B

> str(alsebl)

`data.frame':    11128 obs. of  14 variables:
 $ tag   : num  -27784    47    49    68    71 ...
 $ dbh   : num  NA 437 228 278 269 360 580 NA 311 348 ...
 $ status:Class 'AsIs'  chr [1:11128] NA "A" "A" "A" ...
 $ pom   : num  0 2 1 1 1 2 2 0 2 2 ...
 $ stems :num 1 1 1 1 1 1 1 1  …
 $ date  : num     0 3702 3632 3627 3627 ...
 $ codes :Class 'AsIs'  chr [1:11128] "*" "B" "*" "*" ...
```

Notice that the fields *tag, dbh, pom, stems,* and *date* are numeric, while *status* and *codes* are character fields. The left most numbers on each line are row numbers and serve as row names once the data frame is created.

read.table()
reads a file in table format and outputs a data frame. This be accomplished with a single line command line making it much easier to use than *scan()*. This function is appropriate for the majority of text datasets that have been created by a spread sheet. However, *read.table()* is an inefficient way to read in very large numerical files, especially those with many columns. The *scan()* function is faster and takes up less memory. In fact *read.table()* actually uses scan to read the file, and then processes the results of

scan().The difference between these two functions includes the format the variables in the output file take on. By default in *read.table()*, numeric fields are read in as numeric variables and character fields are read in as factors. The general syntax of read.table() is:

```
>read.table(file, header, sep, quote, dec, row.names, col.names, as.is, skip)
```

file : name of the input file to read data from.
header : a logical value indicating whether the input file contains the names of the variables as its first line.
sep : specifies what the field separator is, which by default, is a blank space between 2 fields as known as "white space". Other common separators are tab ("\t") and commas (",")
quote : specifies the set of quoting characters used to define where one character string begins and ends.
dec : designates the character used for decimal points. In the CTFS datasets the period is used (".").
row.names : a vector of names, or a single number giving the column of the table which contains the names of the rows. The default is sequentially numbered rows.
col.names : a vector of names for the variables. The default is a '"V"' followed by the column number eg V1 V2 V3….
as.is : a logical value. FALSE defines the default behavior, which is to convert the character variables to factors. Designate *as.is* = TRUE to suppress conversion of character variables to factors, leaving them "as is".
skip : designates the number of lines of the input file to skip before beginning to read the data values, as in the case of column names, which you would skip.

Here is a text file called *bci.spp.info.txt* that can be made into a dataset accessible by R and can be read in using *read.table()* . The file is tab delimited text, with no quoting characters and contains a header row with the variable names. The character variables are to remain as characters and are not to be converted to factors.

Example:

sp	genus	species	family	grform	repsize	breedsys	maxht
acacme	Acacia	melanoceras	Fabaceae:Mimos.	U	4	B	6
acaldi	Acalypha	diversifolia	Euphorbiaceae	S	2	M	6
acalma	Acalypha	macrostachya	Euphorbiaceae	U	2	M	5
ade1tr	Adelia	triloba	Euphorbiaceae	U	10	D	5

```
>bcispp=read.table(file="/datasets/bci/spplist.txt", as.is=T, header=T, sep="\t",
quote="")
```

6.2 Writing Datafiles

Once an R dataset has been created, you may want to continue using it in future analyses. If you quit an R session without saving the workspace or saving specific objects, you will lose all objects created in the current session. Instead of recreating the R datasets every time an R session begins, you will want to save them and be able to call them up in a later session.

The best way to accomplish this is to save the created file in your directory structure so that it exists when you exit R.

Alternatively, especially when in a hurry, save the current workspace and loading it again in the next session and the datafile will be available. Remember, that until the file is saved into your directory structure, it does not exist permanently on your disk

save()
saves an R object to a specified file on disk which can be called up at a later date using the *load()* or *attach().* The general syntax is:

```
> save(...., file)
```

…. : a list of the names of the objects to be saved.
file : name of the file on disk, in quotes with the suffix *.rdata*. By default the file is saves in the working directory. Use *setwd()* to change the directory or include the full path of the folder where you want to save the file.

Save the R file *alsebl* created above in the working directory
```
> save(alsebl,file=”alsebl.rdata”)
```

Or in an explicitly provided path /R/results/ folder:

```
> save(alsebl, file=”/R/results/alsebl.rdata”)
```

A binary file called *alsebl.rdata* will be created in the /R/results/ folder, that is portable across all R platforms (Windows, MacOS, Linux). The *name* of the file for the R session will be the file name used when it was created in R. The saved *name* is the name of the file on disk which always includes the *.rdata* suffix. These names DO NOT have to be the same, though it greatly eases remembering what each file contains (see Chapter 4 on file management).

load()

To call up this same file in the next R working session use:

```
> load(“/R/results/alsebl.rdata”)
```

or if the file is in the current working directory:

```
> load("alsebl.rdata")
```

The R file "alsebl" will be available in the first environment. You can also use the "Load Workspace" option under "File" in the menu bar to load the file.

attach()

this effectively does the same thing as *load()* but the file is placed in a different location within the R session. It is places in environment #2 also referred to as R search path #2.

```
> attach("/R/results/alsebl.rdata")
```

By default, the R file "alsebl" will be available in the second environment (*ls(2)*). Once attached or loaded, the variables in *alsebl* can be accessed by giving their names alone.

write.table()

This function is used to export a data frame to a text file for use in other software, such as MS Excel or Word. Remember, R datasets are binary files and are not readable by other software without conversation to a text file. If the R dataset to export is not a data frame, this function will try to convert it to a data frame first. The general syntax is:

```
>write.table(x, file = "", append = FALSE, quote = TRUE, sep = " ",
        na = "NA", dec = ".", row.names = TRUE, col.names = TRUE)
```

x : name of the data frame to be exported. If *x* is not a data frame, the function will try to convert it to a data frame.
file : the name of the file (in quotes) where the data frame is to be saved. The path should be included if you want to save the file in a folder that is not the default.
append : a logical value which specifies what to do if *file* already exists. If 'TRUE', the output is appended to the existing file, if 'FALSE', the file is overwritten, destroying all previous data.
quote : a logical value or a numeric vector. If 'TRUE', all character and factor variables will be surrounded by double quotes. If 'FALSE', nothing is quoted. If it is a numeric vector, its elements are taken as the indices of the columns to quote.
sep : the separator to use to separate the field values within each row. Common field separators are blank spaces, commas, tabs, or semicolons.
na : the string to use for missing values in the data. By default, it is assigned the string "NA", but you may assign it any other character.
dec : the character to use for the decimal point. The most common characters used are "." or ",".
row.names : either a logical value indicating whether the row names of the data frame are to be written, or it can be assigned a character vector of row names to be written.

col.names : either a logical value indicating whether the column names of the data frame are to be used, or it can be assigned a character vector of column names to be written.

To export an R table called *bci.mort.spp* as a text file to the /bci/mortality/ folder use:

```
> write.table(bci.mort.spp, file="/bci/mortality/spp.mort.txt", quote=F, sep="\t")
```

This creates a tab-delimited text file, with no quotes surrounding the character variables, called *spp.mort.txt*. This text file can be called up in MS Word, MS Excel, or any other database management software.

6.3 Creating CTFS Datasets

6.3.1 Preparing Text Datasets

Primary Files

In previous workshops, the datasets at each site were set up as separate species text files under several folders:

- In the \census folder, the species files contained the location information for each tree: tag, gx, gy.
- In the \census0 folder, the species files contained the census information from the first census: tag, dbh (diameter at breast height), status code, pom (point of measure), number of stems, date of measurement (Julian dates), and other site-specific codes.
- Sites with recensus data had folders called \census1, \census2, and so on with measurement data from each recensus.

Currently, however, instead of separate species files, the datasets should be set up as <u>one large text file for each census</u> with all the trees together, in order of species. These files will be referred to as the full text datasets, named using the following pattern:

Siteyear.full.rdata

For example, the BCI 1990 enumeration is named: *bci90.full.rdata*
The BCI 1990 and 1995 merged enumeration file is named : *bci9095.full.rdata*
The BCI 1985 to 1995 (3 censuses) file is named: *bci85to90.full.rdata*

The files for each of the census should include the same exact number of records and should be in exactly the same tree order. This means that as trees die they are NOT removed from a census dataset. When trees are recruited, they are added into ALL previous censuses.

FORMAT

The columns in these full text datasets include:

tag : the number of the tree. Each record of this file should refer to one tree. All multiple stem measurements other than the main stem should be moved to the multiple stem file.
sp : the 4 to 6-letter species code. Every species code should appear in the species file described below.
gx, gy : the x and y coordinates within the plot.
dbh : the diameter measurement from the current census.
pom : point of measure of the diameter.
date : date of measurement in Julian dates, i.e. number of days since a fixed date (we suggest 1/1/80 when BCI, the first plot, was established).
codes : codes that each site wants to keep for later analyses or explanations.

Following are more detailed explanations of these variables.

tag : Each record in the dataset refers to one individual, identified by the tag number. There should be no duplicate tag numbers. Tag numbers should always be in identical and numerical order in all datasets.

sp : All species codes should be valid and included in the species dataset described below. There shouldn't be any extra species codes either in these datasets or in the species list. All valid morphospecies should appear in the species list.

gx, gy : All gx and gy coordinates should fall within the range of the coordinates of your plot. If a location is unknown, it should be given a code of –9. If any coordinate falls exactly on the rightmost or uppermost border, change it to 1 decimal less. For example, for a plot 1000 x 500 m, make sure that your gx coordinates go from 0 to 999.9 and your gy coordinates from 0 to 499.9.

dbh : Dbh should be above 10 mm in all cases where the tree is alive. For multiple-stemmed plants, the largest dbh should appear in this dataset, the rest of the measurements should appear in the multiple stem file (see below). The following dbh codes should be used otherwise:

0 - if the tree is alive, but its main stem broke off and its dbh is below 1 cm, or its stem is under 1.3 m. This is used in the case of resprouts also.
–1 - if the tree died in a later census.
–2 - if the tree had not entered the census yet but will in a later census.
–9 - if the dbh measurement was missed and is unknown.
5 - For Pasoh only, a dbh of 5 refers to trees that were alive and were >=10 but were not measured.

pom : Pom refers to the point of measurement of the diameter. Pom is used when we calculate growth rates. All trees ≥1 cm dbh should be given a pom code of 1 the first time they enter a census, and in all subsequent censuses if the height at which the dbh was measured does not change. If the diameter of the tree is measured at a different height or at another stem (as in the case of multiple-stemmed trees) in a subsequent census, the pom should be increased to the next number. This may happen in trees with buttresses where the height at which the diameter was measured changed, in trees whose main stem broke off or died and whose largest stem is another one, or in trees whose main stem died but have resprouts. Trees that have not entered the census yet, whose dbh is missing, or trees that died should all be given a pom of 0 (zero).

date : Dates refer to the date when the individual was measured. They should be julian dates, calculated from the number of days since January 1st,1980 (the day the first census at BCI began). Make sure the dates are all in the correct format before converting them to julian dates. Some countries specify the month before the day, while others specify the day before the month. Make sure it is consistent throughout the dataset. Check that the dates fall within the range of the census interval. It is very common to find the wrong year entered.

codes : Codes refer to any codes you have in your database that you want to keep and may want to use in your analyses. They should not have spaces between them. Put in underscores, periods, or any other character if you want to separate the codes.

ERROR CHECKING

Following are other inconsistencies to check in those sites with recensus data.

Number and order of records:

Make sure all tag numbers are in exactly the same order in each of your full datasets. There should be the same number of records in all your census datasets.

Screen the dbh remeasurements:

You should recheck all dbhs that are smaller (allowing for some shrinkage or slight measurement error) or abnormally larger than the previous census. A dbh may be smaller than the previous census if the main stem broke off or died, or if the height at which the measurement was taken changed. If so, make sure that it is annotated and increase the pom to the next larger number.

If you do find an error in a previous dbh measurement, do not change the previous measurement in your main database, since you will be estimating. Keep the measurements, make a note of the problem, and let each scientist/analyst make their own decision on what to do with the error.

For plots with 3 or more censuses, verify that all **dead** trees remain dead in the later censuses. If a tree was found alive in a later census, the tag number may be incorrect or you may have to change the previous "dead" code to "alive" and change the dbh to –9 (missing).

For plots with 3 or more censuses, verify that all **recruit** trees are indicated as to be recruits in earlier censuses.

SUPPORT FILES

Besides the full text datasets, each site should also prepare the following files.

Species information file: site.spp.info.txt

This file contains the information for each species found at each site. Each species is a row and the columns are the information about each species. Every single species code that appears in your full datasets should appear in this list (including all morphospecies) and every species code that appears in this species list should appear in your full datasets. The file should have the following columns:

sp : the four to six-letter species code,
genus : the genus name
species : the species name
family : the family the species belongs to
grform : the growth form of the species, i.e. shrub, understory tree, mid-sized tree, canopy tree.
repsize : the reproductive dbh (cm) of the species
breedsys : breeding system, i.e. bisexual, dioecious, monoecious, polygamous, etc.
maxht : maximum height (m) attained by the species.

All columns should have something filled in for every record and there should be no spaces within each column. If your species name consists of 2 or more words, for example, make sure to insert an underscore, dash or period in the space between the words.

The first 5 columns should be filled in as best as possible. No columns should be left blank. If a species has not been identified yet, put "Unidentified" or "Unknown" in the relevant columns (*family, genus, species*). In the cases where information is missing, put in a –1 or –9 in the columns for numeric fields (*repsize, maxht*) and a '*' in the character fields (*grform, breedsys*).

Any other information that is identified with a species can be placed into this file. For instance, *timber type, pioneer status* etc. Classification of species that is derived from analyses can also be kept here such as the *degree of habitat specificity* that can be determined by using the torus habitat analysis. These classifications of species can be used in further analyses.

An rdata file should be made from this text file and saved in the same directory that the site rdata census files are kept. See *read.table()* and *save()* functions above. Save the file as *site.spp.info.rdata.* eg. *bci.spp.info.rdata.*

Quadrate information file: site.quad.info.txt

This file contains information about each 20 by 20 m quadrate. Each quadrate is a row and the columns are the information about each quadrate. There must be a row for each quadrate. There are 1250 rows for a standard 1000 by 500 m CTFS plot. At a minimum this file should contain the elevation data (m) for each 5 meter interval of the plot in order of x and y. In addition, slope and convexity can be added for use in habitat analysis. The elevation file should be tab-delimited. The columns are:

x : x coordinate of tree (east-west axis of plot)
y : y coordinate of tree (north-south axis of plot)
elev : elevation in m a.s.l.
slope : degrees (see "torus" analysis for computation)
convex : degrees (see "torus" analysis for computation)

The results of any form of habitat classification can also be put into this file. For instance, the "torus" habitat analysis can be used to provide 8 *classes of habitat* and these are assigned to each quadrate.

An rdata file should be made from this text file and saved in the same directory that the site rdata census files are kept. See *read.table()* and *save()* functions above. Save the file as *site.quad.info.rdata.* eg. *bci.quad.info.rdata.*

Multiple stem file: siteyear.mult.txt

There should be one multiple stem file for each census. Each file should include at least a tag and dbh column. Each multiple stem file includes all stem measurements of all multiple-stemmed individuals for that census, **excluding** the measurement of the largest main stem, which is in the full database. The multiple stem files do not include the largest stem measurement of each tree. The multiple stem files from each census will not necessarily have the same number of records nor the same tag numbers, unlike the full datasets for each census. At this time the multiple stem files are named: *mult0* (first census), *mult1* (second census), etc.

An rdata file should be made from this text file and saved in the same directory that the site rdata census files are kept. See *read.table()* and *save()* functions above. To be consistent with the naming of other files, save the file as *siteyear.mult.rdata.* eg. *bci95.mult.rdata.*

6.3.1 Preparing R Datasets

Once your text datasets are cleaned up, checked for errors and inconsistencies, and in the correct format, make them into rdata files.

Once your tab-delimited text file (with no quotes) is ready with the appropriate column headings, create the full dataset in R with the following command:

```
> siteyear.full=read.table(file=”FILENAME”, as.is=T, header=T, sep="\t", quote="")
```

where *site* refers to the site (bci, yasuni, sinharaja, hkk, etc.), *year* refers to the census year, and *FILENAME* refers to the name of your tab-delimited text file. Remember to include the path in the *FILENAME* if your file is not found in working directory.

Similarly, to create a tab-delimited species text file with no quotes to an R species dataset, use the following command:

```
> sitespp.info=read.table(file=”FILENAME”, as.is=T, header=T, sep="\t", quote="")
```

To create an R multiple stem dataset:

```
> siteyear.mult0= read.table(file=”mult0.txt”, as.is=T, header=T, sep="\t", quote="")
```

To create the R elevation file:

```
> sitequad.info=read.table(file=”FILENAME”, as.is.=T, header=T, sep=”\t”, quote=“”)
```

You may then use this file to create an elevation matrix with the CTFS R function *readelevdata().*

Split datasets by species

After the R full datasets are created, you can create a list of data frames separated by species with the CTFS R function called *sep.data().*

```
> siteyear.spp=sep. data(siteyear.full,sepcol=”sp”,handle.na=NA)
```

This contains the same fields as the full dataset, but separated by species. The file *siteyear.spp* is a list of data frames, one for each species.

To extract a single species' data frame from *siteyear.spp*, use the CTFS R function *load.species()*. To create a subset of the data frames in *siteyear.spp* use *sep.species().*

Merged datasets from two censuses

To run the growth, mortality, and recruitment functions, you need to merge the full datasets from at least two censuses, using the CTFS R function *mergecensus().* Following is an example using the 90 and 95 BCI censuses.

```
> bci9095.full=mergecensus("bci", census0=bci90.full, census1=bci95.full)
```

Again, it is very important that the full datasets from the two censuses be in the exact same tag order, since the program just merges the two datasets, it does not verify that the tag numbers match before merging.

After merging the two datasets, you can create the split files on this merged dataset with the sep.fulldata() function mentioned above:

```
> bci9095.spp=sep.data(bci9095.full)
```

Save all these R datasets to a folder on your computer with the *save()* function described above.