Data loading methods for the R package glatos

Updated: 2019-02-08

Contents

1 Overview

This vignette describes methods for loading data into the R package *glatos*. Sections are organized by data type (*detections*, *receiver locations*, etc), and each section contains examples for:

1. data in standardized formats from the Great Lakes Acoustic Telemetry Observation System (GLATOS), the Ocean Tracking Network (OTN), and VEMCO using built-in data loading functions; and
2. data in non-standard formats that require loading using non-*glatos* functions and modification to meet *glatos* requirements.

1.1 Loading data from GLATOS, OTN, and VEMCO

The *glatos* package contains five functions (see Built-in functions) designed to load data files in standardized formats from the GLATOS, the OTN, and VEMCO. Each data loading function:

1. loads data into an R session consistently and efficiently using the best available methods and
2. returns an object that meets the requirements of *glatos* package functions.

Thus, using *glatos* load functions ensures that resulting data conform to the requirements of other functions in the package (e.g., *summarize_detections*, *detection_bubble_plot*) and relieves users from the work of reformatting their data to meet requirements of each specific function.

1.1.1 Built-in functions

The *glatos* package includes five data loading functions:

- ***read_glatos_detections*** reads detection data from a comma-separated-values text file obtained from the GLATOS Data Portal and returns an object of class *glatos_detections* that is also a *data.frame*.
- ***read_otn_detections*** reads detection data from a comma-separated-values text file obtained from the Ocean Tracking Network and returns an object of class *glatos_detections* that is also a *data.frame*.
- ***read_glatos_receivers*** reads receiver data from a comma-separated-values text file obtained from the GLATOS Data Portal and returns an object of class *glatos_receivers* that is also a *data.frame*.
- ***read_glatos_workbook*** reads data from a GLATOS project-specific MS Excel workbook (*.xlsm file) and returns a list of class *glatos_workbook* with two-elements; one of class *glatos_receivers* and one of class *glatos_animals* (both are also *data.frame*s).
- ***read_vemco_tag_specs*** reads tag specification data from an MS Excel workbook (*.xls file) provided by VEMCO and returns a list with two elements; one containing tag specifications and one containing tag operating schedules. (both are also *data.frame*s).

1.1.2 Data objects and classes

Most of the functions listed above return an object with a *glatos*-specific S3 class name (e.g., *glatos_detections*) in addition to a more general class (e.g., *data.frame*). Currently, no methods exist for *glatos* classes and such classes are not explicitly required by any function, so *glatos* classes can merely be thought of as labels showing that the objects were produced by a *glatos* function and will therefore be compatible with other *glatos* functions. Beware, as with any S3 class, that it is possible to modify a *glatos* object to the point that is will no longer be compatible with *glatos* functions. The Data Requirements vignette provides an overview of data requirements of *glatos* functions.

1.2 Loading data from other sources

To use *glatos* functions with data that are not in one of the standard formats described above, those data will need to be:

1. loaded into R using some other function (e.g., *read_csv*) and

2. modified to ensure that all requirements of the desired function are met.

Strictly speaking, there are no requirements of the package as a whole, but input data are checked within each individual function to determine if requirements are met. Nonetheless, the Data Requirements vignette provides a set of data requirements, including column names, types, and formats, that will ensure compatibility with all *glatos* functions.

For each data type (e.g., *detection*, *receiver location*, etc), this vignette shows how data from a comma-separated-values text file can be loaded into R using non-*glatos* functions and then modified to meet the *glatos* requirements.

1.3 Tips to improve speed and efficiency

The main examples in the this vignette use only base R functions. However, there are many contributed packages that can provide functions that can improve workflow speed and efficiency. After most examples using base R functions, boxed examples are also given to expose users to alternative examples using functions from contributed packages. Most boxed tips show use of functions from the *data.table* package. If you are not familiar with *data.table*, then read through the introductory vignette (see `vignette("datatable-intro", package = "data.table")`). For more on *data.table*, see the vignettes (`browseVignettes("data.table")`). In addition to *data.table*, one boxed tip draws from the *lubridate* package because it is, to our knowledge, the fastest way to coerce timestamps strings to the date-time class *POSIXct*. No other examples show use of *tidyverse* packages (*dplyr*, *ggplot2*, *readr*), simply because some of us have not yet drunk the kool-aid. A future version of this vignette may include more examples using the *tidyverse* or other packages.

A few notes about boxed-example code:

1. Make sure the relevant packages are installed and attached:

```
#install.packages("data.table")
library(data.table)

#install.packages("lubridate")
library(lubridate)
```

2. For *data.table* functions, make sure you've converted the target object to *data.table* class using *setDT*:

```
dx <- data.frame(a = 1 , b = 2)
setDT(dx) #convert to data.table
```

3. Go all-in. Running some base R examples, some boxed examples, or both will cause trouble. For example if a base R example changes column names in an object, then the old column names won't exist if you later run the boxed example that changes those same column names using *data.table* functions.

2 Detection data

2.1 Requirements

glatos functions that accept detection data as input will typically require a *data.frame* with the following seven columns:

- detection_timestamp_utc
- receiver_sn
- deploy_lat
- deploy_long
- transmitter_codespace
- transmitter_id

- sensor_value
- sensor_unit
- animal_id

Some functions will also require at least one categorical column to identify location (or group of locations). These can be specified by the user, but examples of such columns in a GLATOS standard detection file are:

- glatos_array
- station
- glatos_project_receiver

For definitions of any of the above fields, see the Data Requirements vignette) and function-specific help files (e.g., `?summarize_detections`).

Any *data.frame* that contains the above columns (in the correct formats) should be compatible with all *glatos* functions that accept detection data as input. Use of the data loading functions *read_glatos_detections* and *read_otn_detections* will ensure that these columns are present and formatted correctly, but can only be used on data in GLATOS and OTN formats. Data in other formats will need to be loaded using other functions (e.g., *read.csv*, *fread*, etc.) and carefully checked for compatibility with *glatos* functions (see Other formats - CSV file exported from a VUE database).

2.2 Examples

2.2.1 Loading GLATOS data

The *read_glatos_detections* function reads detection data from a standard detection export file (*.csv file) obtained from the GLATOS Data Portal and checks that the data meet *glatos* package requirements. Data are read using *fread* in the *data.table* package, timestamps are formatted as class *POSIXct* and dates are formatted as class *Date*.

First, we will use *system.file* to get the path to the *walleye_detections.csv* file included in the *glatos* package.

```
# Set path to walleye_detections.csv example dataset
wal_det_file <- system.file("extdata", "walleye_detections.csv",
                            package = "glatos")
```

Next, we will load data from *walleye_detections.csv* using *read_glatos_detections*.

```
# Attach glatos package
library(glatos)

# Read in the walleye_detections.csv file using `read_glatos_detections`
walleye_detections <- read_glatos_detections(wal_det_file)
```

Let's view the structure of the resulting data frame (we've modified the `str` default arguments to show only first record in each column).

```
# View the structure and data from first row
str(walleye_detections)
#> Classes 'glatos_detections' and 'data.frame':    7180 obs. of  30 variables:
#> $ animal_id : chr "153" ...
#> $ detection_timestamp_utc : POSIXct, format: "2012-04-29 01:48:37" ...
#> $ glatos_array : chr "TTB" ...
#> $ station_no : chr "2" ...
#> $ transmitter_codespace : chr "A69-9001" ...
#> $ transmitter_id : chr "32054" ...
#> $ sensor_value : num NA NA ...
#> $ sensor_unit : chr NA ...
#> $ deploy_lat : num 43.4 ...
```

```
#> $ deploy_long : num -84 ...
#> $ receiver_sn : chr "113213" ...
#> $ tag_type : chr NA ...
#> $ tag_model : chr NA ...
#> $ tag_serial_number : chr NA ...
#> $ common_name_e : chr "walleye" ...
#> $ capture_location : chr "Tittabawassee River" ...
#> $ length : num 0.565 0.565 ...
#> $ weight : num NA NA ...
#> $ sex : chr "F" ...
#> $ release_group : chr NA ...
#> $ release_location : chr "Tittabawassee" ...
#> $ release_latitude : num NA NA ...
#> $ release_longitude : num NA NA ...
#> $ utc_release_date_time : POSIXct, format: "2012-03-20 20:00:00" ...
#> $ glatos_project_transmitter: chr "HECWL" ...
#> $ glatos_project_receiver : chr "HECWL" ...
#> $ glatos_tag_recovered : chr "NO" ...
#> $ glatos_caught_date : Date, format: NA ...
#> $ station : chr "TTB-002" ...
#> $ min_lag : num 258 137 ...
```

The result is an object with 30 columns and two classes: *glatos_detections* and *data.frame*. The *glatos_detections* class label indicates that the data set was created using a *glatos* load function and therefore should meet requirements of any *glatos* function that accepts detection data as input. See the Data Requirements vignette) for field definitions.

2.2.2 Loading OTN data

The *read_otn_detections* function reads in detection data (*.csv files) obtained from the Ocean Tracking Network and reformats the data to meet requirements of *glatos* functions. Data are read using *fread* in the *data.table* package, timestamps are formatted as class *POSIXct* and dates are formatted as class *Date*.

```
# Set path to blue_shark_detections.csv example dataset
shrk_det_file <- system.file("extdata", "blue_shark_detections.csv",
                             package = "glatos")

# Read in the blue_shark_detections.csv file using `read_otn_detections`
blue_shark_detections <- read_otn_detections(shrk_det_file)


# View the structure of blue_shark_detections
str(blue_shark_detections)
#> Classes 'glatos_detections' and 'data.frame':    3000 obs. of  34 variables:
#> $ collectioncode : chr "NSBS" ...
#> $ animal_id : chr "NSBS-Hooker" ...
#> $ scientificname : chr "Prionace glauca" ...
#> $ commonname : chr "blue shark" ...
#> $ datelastmodified : Date, format: "2014-12-18" ...
#> $ detectedby : chr "HFX" ...
#> $ receiver_sn : chr "HFX" ...
#> $ glatos_array : chr "HFX047" ...
#> $ receiver : chr "146" ...
#> $ bottom_depth : num 151 151 ...
```

```
#> $ receiver_depth : num 146 146 ...
#> $ transmitter_id : chr "A69-9001-24395" ...
#> $ transmitter_codespace : chr "A69-9001" ...
#> $ sensorname : chr NA ...
#> $ sensorraw : num NA NA ...
#> $ sensortype : chr "pinger" ...
#> $ sensorvalue : num NA NA ...
#> $ sensorunit : chr NA ...
#> $ detection_timestamp_utc: POSIXct, format: "2014-08-29 06:11:09" ...
#> $ timezone : chr "UTC" ...
#> $ deploy_long : num -63.2 ...
#> $ deploy_lat : num 44.2 ...
#> $ st_setsrid_4326 : chr "0101000020E61000008A1F63EE5A9E4FC04F1E166A4D1B4640" ...
#> $ yearcollected : int 2014 2014 ...
#> $ monthcollected : int 8 8 ...
#> $ daycollected : int 29 29 ...
#> $ julianday : int 241 241 ...
#> $ timeofday : num 6.19 ...
#> $ datereleasedtagger : Date, format: NA ...
#> $ datereleasedpublic : Date, format: NA ...
#> $ local_area : chr "HALIFAX" ...
#> $ notes : chr NA ...
#> $ citation : chr "Hebert, D., Barthelotte, J., O'Dor, R., Stokesbury, M., Branton,
#>    R. 2009. Ocean Tracking Network Halifax Canada"| __truncated__ ...
#> $ unqdetecid : chr "HFX-A69-9001-24395-180148" ...
```

The resulting object has 34 columns, many of which are not present in the GLATOS standard format. However, some columns have been modified to meet *glatos* requirements, and thus, the *glatos_detections* class name has been added.

2.2.3 Other formats - CSV file exported from a VUE database

Detection data in any format than GLATOS or OTN will need to be
modified to meet the requirements of *glatos* functions. Here, we show an example using detection data that have been exported from a VEMCO VUE database. There is currently no *glatos* function to load detection data directly into an R session from VUE software, so data in that format will need to be:

1. loaded into R using some other function (e.g., *read_csv*) and
2. modified to ensure that all requirements of the desired function are met.

In the example below, we will use the base R functions *read.csv* and *as.POSIXct* to load detection data from a csv file and reformat the data to be consistent with the schema described above. Tip boxes will also show alternatives (simpler and/or faster) for these methods using functions in the *data.table* and *lubridate* packages.

First, get the path to a file (*.csv) that contains detection data exported from VEMCO VUE software. Such a file is included in the *glatos* package.

```
#get path to example CSV file included with glatos package
csv_file <- system.file("extdata", "VR2W_109924_20110718_1.csv",
                        package = "glatos")
```

Now that we have the path to a VUE export file, we will read the data using *read.csv*. In this case we are also setting some *read.csv* arguments to non-default values. First, we set `as.is = TRUE` so that character values are treated as characters and not converted to factors. Second, we set `check.names = FALSE` to prevent conversion of syntactically-invalid column names to syntactically valid names. This simply keeps the names exactly as they appear in the source text file rather than, for example, replacing spaces with a dot (`.`). This does mean that we need to wrap those column names in back-ticks when called (e.g., `` dtc$`Sensor Value` ``). Third, we set `fileEncoding = "UTF-8-BOM"` to match the encoding of the text file. If this argument is omitted then you might see the special characters added to the first column name. Setting the *fileEncoding* may also slow down the import.

```
dtc <- read.csv(csv_file, as.is = TRUE, check.names = FALSE,
                fileEncoding = "UTF-8-BOM")
```

data.table tip: Use *fread* instead of *read.csv*.

```
#read data from csv file using data.table::fread
dtc <- fread(csv_file)
```

fread is fast. That's one reason it is used used by *read_glatos_detections* and other *glatos* functions.

Now we will reformat to be consistent with a *glatos_detections* object. We will do this for each of the required columns described above.

2.2.3.1 *detection_timestamp_utc*

Change the column name from *Date and Time (UTC)* to *detection_timestamp_utc*. There are many ways to do this (e.g., reference columns by number; `names(dtc)[1] <- "detection_timestamp_utc"`) but in the code below use of `match()` to get the column number is robust to changes in column order.

```
#change column name
names(dtc)[match("Date and Time (UTC)", names(dtc))] <- "detection_timestamp_utc"
```

data.table tip: Use *setnames* to change column names.

```
#use data.table::setnames to change column names via old and new names
setnames(dtc, "Date and Time (UTC)", "detection_timestamp_utc")
```

setnames(x, old, new)... could it be more intuitive?

Notice that there are no assignment operators (<- or =) in this code. This is because *setnames*, like other *data.table* functions, updates the target object (in this case *dtc*) directly (aka: *by reference*).

Finally, we format the timestamp column using base R function *as.POSIXct*. All *POSIXct* objects are stored internally as a number representing the number of elapsed seconds since "1970-01-01 00:00:00" in UTC. When we convert a character string to *POSIXct*, we need to tell R *how* to convert it–namely the time zone of the input data. By default, *as.POSIXct* will assume your local system time zone (e.g., the one returned by `Sys.timezone()`. To prevent timezone errors, *always* specify time zone (using the *tz* argument) whenever you coerce any timestamp to *POSIXct*. In this case, the timestamps were exported from VUE in UTC, so we use the following:

```
dtc$detection_timestamp_utc <- as.POSIXct(dtc$detection_timestamp_utc,
                                          tz = "UTC")

#take a peek
str(dtc$detection_timestamp_utc)
#>  POSIXct[1:69708], format: "2011-04-11 20:17:49" ...
```

```
#first few records
head(dtc$detection_timestamp_utc, 3)
#> [1] "2011-04-11 20:17:49 UTC" "2011-05-08 05:38:32 UTC" "2011-05-08 05:41:09 UTC"
```

~~~~~~

data.table tip: Use *:=* to add or modify a column.

:= is an assignment operator for *data.table* objects that assigns objects by reference. We use it because it is more compact than base R methods.

```
#use ':=' to format timestamps
dtc[ , detection_timestamp_utc := as.POSIXct(detection_timestamp_utc,
                                             tz = "UTC")]
```

Note that, like in the previous boxed tip, there are no assignment operators <- or = because *dtc* is updated by reference.

~~~~~~

~~~~~~

lubridate tip: Use *fast_strptime* to set timestamps.

```
#lubridate::fast_strptime is the fastest way we know to ceorce strings to POSIX
dtc[ , detection_timestamp_utc :=
         lubridate::fast_strptime(detection_timestamp_utc,
                                  format = "%Y-%m-%d %H:%M:%OS",
                                  tz = "UTC",
                                  lt = FALSE)]
```

fast_strptime requires a bit more code because we have to specify `format` and set `lt = FALSE` so that *POSIXct* is returned instead of the default (*POSIXlt*) for this function.

Notice that we formatted the timestamps using *fast_strptime* but also used *data.table*'s *set* operator (*:=*) to assign it to the target column.

~~~~~~

2.2.3.2 *receiver_sn*

There is no single column in the VUE export data with receiver serial number, so we need to extract it from the *Receiver* column.

```
dtc$Receiver[1]
#> [1] "VR2W-109924"
```

To do this, we will write a function (*get_rsn*) to extract the second element of the hyphen-delimited string in the *Receiver* column using the base R function *strsplit*. We then use the *sapply* function to "apply" our custom function to each element (record) of the *Receiver* column.

```
#make new function to extract second element from a hyphen-delimited string
get_rsn <- function(x) strsplit(x, "-")[[1]][2]

#apply get_rsn() to each record in Receiver column
dtc$receiver_sn <- sapply(dtc$Receiver, get_rsn)
```

~~~~~~

data.table tip: Use *by* argument in *data.table* to update a column by groups.

```
#make new column "receiver_sn"; parse from "Receiver"
dtc[ , receiver_sn := get_rsn(Receiver), by = "Receiver"]
```
~~~~~~

This is more efficient because it operates on groups instead of each individual record.

~~~~~~

2.2.3.3 *deploy_lat* and *deploy_long*

The *Latitude* and *Longitude* values are all zero in this data set because the VUE database from which these data were exported did not contain any latitude or longitude data. To add those data to these detections, we will make a new data frame containing *latitude* and *longitude* data along with other receiver data and merge the new receiver data with the detection data.

The code below shows a simple left join on *receiver_sn*, which assigns the same receiver location data to all detection records on that receiver without time consideration.

```
#make an example receiver data frame
rcv <- data.frame(
        glatos_array = "DWM",
        station = "DWM-001",
        deploy_lat = 45.65738,
        deploy_long = -84.46418,
        deploy_date_time = as.POSIXct("2011-04-11 20:30:00", tz = "UTC"),
        recover_date_time = as.POSIXct("2011-07-08 17:11:00", tz = "UTC"),
        ins_serial_no = "109924",
        stringsAsFactors = FALSE)

#left join on receiver serial number to add receiver data to detections
dtc <- merge(dtc, rcv, by.x = "receiver_sn", by.y = "ins_serial_no",
             all.x = TRUE)

# take a look at first few rows
head(dtc, 3)
#>   receiver_sn detection_timestamp_utc    Receiver    Transmitter Transmitter Name
#> 1      109924     2011-04-11 20:17:49 VR2W-109924 A69-1303-63366               NA
#> 2      109924     2011-05-08 05:38:32 VR2W-109924  A69-9002-4043               NA
#> 3      109924     2011-05-08 05:41:09 VR2W-109924  A69-9002-4043               NA
#>   Transmitter Serial Sensor Value Sensor Unit Station Name Latitude Longitude
#> 1                 NA           NA                       NA        0         0
#> 2                 NA            5         ADC           NA        0         0
#> 3                 NA            7         ADC           NA        0         0
#>   glatos_array station deploy_lat deploy_long    deploy_date_time
#> 1          DWM DWM-001   45.65738   -84.46418 2011-04-11 20:30:00
#> 2          DWM DWM-001   45.65738   -84.46418 2011-04-11 20:30:00
#> 3          DWM DWM-001   45.65738   -84.46418 2011-04-11 20:30:00
#>     recover_date_time
#> 1 2011-07-08 17:11:00
#> 2 2011-07-08 17:11:00
#> 3 2011-07-08 17:11:00
```

Note that new columns have been added to *dtc*, including *deploy_lat*, *deploy_long*, and two columns (*glatos_array* and *glatos_station*) that could serve as optional location grouping variables. Columns *deploy_date_time* and *recover_date_time* (*POSIXct* objects) are not required columns, but are useful for removing detections that occurred before receiver deployment or after recovery.

Two limitations of this simple join shown above are that it:

- is inadequate if any receiver is deployed at more than one location.
- includes detections that occurred before receiver deployment and after receiver recovery.
~~~~~~

To ensure that detections are correctly associated with a location, we will subset detections to omit any that occurred before deployment or after recovery. For convenience, we use the base R function *with* so that we do not have to repeatedly call *dtc$...* but note that this can be somewhat risky (see `?with`).

```
#count rows before subset
nrow(dtc)
#> [1] 69708

#subset deployments between receiver deployment and recovery (omit others)
dtc <- with(dtc, dtc[detection_timestamp_utc >= deploy_date_time &
                     detection_timestamp_utc <= recover_date_time, ])
```

~~~~~~

data.table tip: Use *between* to subset records by intervals.

```
#subset deployments between receiver deployment and recovery (omit others)
dtc <- dtc[between(detection_timestamp_utc,
                   deploy_date_time, recover_date_time), ]
```

Note that the <- assignment operator is used here because we are subsetting the *data.table* object `dtc` and there is no assignment operator inside the square brackets.

~~~~~~

```
#count rows after subset
nrow(dtc)
#> [1] 69703
```

We removed five rows. Those detections either occurred before receiver deployment or after receiver recovery, so the location of those detections are unknown.

2.2.3.4 *transmitter_codespace* and *transmitter_id*

There is no single column in the VUE export data with transmitter code space or transmitter ID code, so we need to extract them from the *Transmitter* column. Like we did with *receiver_sn*, we'll make new functions to extract the id and codespace from each record, then use *sapply* to "apply" each of those functions to each record in *Transmitter*. Note that the codes space requires an extra step, after we split the string on "-" we then paste the first and second back together to create the code space string.

```
#make a new function to extract id from Transmitter
#i.e., get third element of hyphen-delimited string
parse_tid <- function(x) strsplit(x, "-")[[1]][3]

#make a new function to extract codespace from Transmitter
#i.e., get first two elements of hyphen-delimited string
parse_tcs <- function(x) {
    #split on "-" and keep first two extracted elements
    tx <- strsplit(x, "-")[[1]][1:2]
    #re-combine and separate by "-"
    return(paste(tx[1:2], collapse = "-"))
  }

#apply parse_tcs() to Transmitter and assign to transmitter_codespace
dtc$transmitter_codespace <- sapply(dtc$Transmitter, parse_tcs)

#apply parse_tid() to Transmitter and assign to transmitter_id
dtc$transmitter_id <- sapply(dtc$Transmitter, parse_tid)
```

~~~~~~

data.table tip: Use the *functional form* of *:=* to add/modify more than one column.

```
dtc[ , `:=`(transmitter_codespace = parse_tcs(Transmitter),
            transmitter_id = parse_tid(Transmitter)),
       by = "Transmitter"]
```

See `?data.table::set` for description and examples of functional form of `:=`.

~~~~~~

2.2.3.5 *sensor_value* and *sensor_unit*

Change the column names from *'Sensor Value'* and *'Sensor Unit'* to *sensor_value* and *sensor_unit*.

```
#change column name
names(dtc)[match(c("Sensor Value", "Sensor Unit"), names(dtc))] <-
                 c("sensor_value", "sensor_unit")
```

~~~~~~

data.table tip: Use *setnames* to change multiple column names.

```
setnames(dtc, c("Sensor Value", "Sensor Unit"),
              c("sensor_value", "sensor_unit"))
```

~~~~~~

```
str(dtc)
#> 'data.frame':    69703 obs. of  19 variables:
#> $ receiver_sn : chr "109924" ...
#> $ detection_timestamp_utc: POSIXct, format: "2011-05-08 05:38:32" ...
#> $ Receiver : chr "VR2W-109924" ...
#> $ Transmitter : chr "A69-9002-4043" ...
#> $ Transmitter Name : logi NA ...
#> $ Transmitter Serial : logi NA ...
#> $ sensor_value : int 5 7 ...
#> $ sensor_unit : chr "ADC" ...
#> $ Station Name : logi NA ...
#> $ Latitude : int 0 0 ...
#> $ Longitude : int 0 0 ...
#> $ glatos_array : chr "DWM" ...
#> $ station : chr "DWM-001" ...
#> $ deploy_lat : num 45.7 ...
#> $ deploy_long : num -84.5 ...
#> $ deploy_date_time : POSIXct, format: "2011-04-11 20:30:00" ...
#> $ recover_date_time : POSIXct, format: "2011-07-08 17:11:00" ...
#> $ transmitter_codespace : chr "A69-9002" ...
#> $ transmitter_id : chr "4043" ...
```

2.2.3.6 *animal_id*

The *animal_id* column was not included in the VUE database and will need to come from another source. If no tags were re-used, then a simple solution might be to create a new column and assign it the values of *Transmitter* column. In this example, however, we will make a new data frame containing *animal* data and merge it with detection data.

The code below shows a simple left join on *transmitter_codespace* and *transmitter_id*, which assigns the same receiver location data to all detection records of each transmitter without time consideration.

```r
#make an example animal (fish) data frame
fsh <- data.frame(
        animal_id = c("1", "4", "7", "128"),
        tag_code_space = "A69-1601",
        tag_id_code = c("439", "442", "445", "442"),
        common_name = "Sea Lamprey",
        release_date_time = as.POSIXct(c("2011-05-05 12:00",
                                         "2011-05-05 12:00",
                                         "2011-05-06 12:00",
                                         "2011-06-08 12:00"),
                                       tz = "UTC"),
        recapture_date_time = as.POSIXct(c(NA, "2011-05-26 15:00", NA, NA),
                                         tz = "UTC"),
        stringsAsFactors = FALSE)

#simple left join on codespace and id
dtc <- merge(dtc, fsh, by.x = c("transmitter_codespace", "transmitter_id"),
                       by.y = c("tag_code_space", "tag_id_code"),
                       all.x = TRUE)
```

Two limitations of this simple join are that it:

- is inadequate if any transmitter was deployed more than once.
- includes detections that occurred before transmitter deployment (animal release) and after transmitter recovery (animal recapture).

Note that one tag (`tag_id_code = 442`) was re-used, but we did not account for this in the above merge (a simple left join). So we now need to subset to omit detections that occurred before release or after recapture.

```r
#count rows before subset
nrow(dtc)
#> [1] 69711

#subset detections to include only those between release and recapture
# or after release if never recaptured
dtc <- with(dtc, dtc[detection_timestamp_utc >= release_date_time &
                     (detection_timestamp_utc <= recapture_date_time |
                        is.na(recapture_date_time)) , ])
```

> ~~~~~~
>
> **data.table tip**
> Use *between* to query records or evaluate statements by intervals.
>
> ```r
> dtc <- dtc[between(detection_timestamp_utc, release_date_time,
>                    recapture_date_time) | is.na(recapture_date_time), ]
> ```
>
> ~~~~~~

```r
#count rows after subset
nrow(dtc)
#> [1] 69703
```

We should now have a detection dataset that will meet the requirements of *glatos* functions.

3 Receiver location data

3.1 Requirements

glatos functions that accept receiver location data as input will typically require a *data.frame* with one or more of the following columns:

- deploy_lat
- deploy_long
- deploy_date_time
- recover_date_time

Some functions will also require at least one categorical column to identify location (or group of locations). These can be specified by the user, but examples of such columns in a GLATOS standard receiver locations file are:

- glatos_array
- station
- glatos_project_receiver

For definitions of any of the above fields, see the Data Requirements vignette) and function-specific help files (e.g., `?abacus_plot`).

Any *data.frame* that contains the above columns (in the correct formats) should be compatible with all *glatos* functions that accept receiver data as input. Use of the data loading function *read_glatos_receivers* will ensure that these columns are present and formatted correctly, but can only be used on data in GLATOS format. Data in other formats will need to be loaded using other functions (e.g., *read.csv*, *fread*, etc.) and carefully checked for compatibility with *glatos* functions (see Other formats - CSV file exported from a VUE database).

3.2 Examples

3.2.1 Loading GLATOS data (entire network)

The *read_glatos_receivers* function reads in receiver location data obtained from the GLATOS Data Portal and checks that the data meet requirements of *glatos* functions. Data are read using *fread* in the *data.table* package, timestamps are formatted as class *POSIXct*.

We will get the path to the *sample_receivers.csv* (example included in the *glatos* package) using *system.file*, then read the data using *read_glatos_receivers*, and view the structure of the result.

```
#get path to example receiver_locations file
rec_file <- system.file("extdata", "sample_receivers.csv",
                        package = "glatos")

#read sample_receivers.csv using 'read_glatos_receivers'
rcv <- read_glatos_receivers(rec_file)

#view structure
str(rcv)
#> Classes 'glatos_receivers' and 'data.frame': 898 obs. of  23 variables:
#> $ station : chr "WHT-009" ...
#> $ glatos_array : chr "WHT" ...
#> $ station_no : chr "9" ...
#> $ consecutive_deploy_no: int 1 2 ...
#> $ intend_lat : num NA NA ...
#> $ intend_long : num NA NA ...
#> $ deploy_lat : num 43.7 ...
```

```
#> $ deploy_long : num -82.5 ...
#> $ recover_lat : num NA NA ...
#> $ recover_long : num NA NA ...
#> $ deploy_date_time : POSIXct, format: "2010-09-22 18:05:00" ...
#> $ recover_date_time : POSIXct, format: "2012-08-15 16:52:00" ...
#> $ bottom_depth : num NA NA ...
#> $ riser_length : num NA NA ...
#> $ instrument_depth : num NA NA ...
#> $ ins_model_no : chr "VR2W" ...
#> $ glatos_ins_frequency : int 69 69 ...
#> $ ins_serial_no : chr "109450" ...
#> $ deployed_by : chr "" ...
#> $ comments : chr "" ...
#> $ glatos_seasonal : chr "NO" ...
#> $ glatos_project : chr "HECWL" ...
#> $ glatos_vps : chr "NO" ...
```

The result is an object with 23 columns (including the required columns described above) and two classes: *glatos_receivers* and *data.frame*. The *glatos_receivers* class label indicates that the data set was created using a *glatos* load function and therefore should work with any *glatos* function that accepts receiver data as input.

3.2.2 Loading GLATOS data (single project workbook)

The *read_glatos_workbook* function reads in receiver location data from a standard GLATOS project workbook (*.xlsm file) and checks that the data meet requirements of *glatos* functions. Data are read using *readWorkbook* in the *openxlsx* package, timestamps are formatted as class *POSIXct*.

We will get the path to the *walleye_workbook.xlsm* (example included in the *glatos* package) using *system.file*, then read the data using *read_glatos_workbook*, and view the structure of the result.

```
#get path to example walleye_workbook.xlsm file
wb_file <- system.file("extdata", "walleye_workbook.xlsm",
                        package = "glatos")

#read walleye_workbook.xlsm using 'read_glatos_workbook'
wb <- read_glatos_workbook(wb_file)

#view structure
class(wb)
#> [1] "glatos_workbook" "list"
names(wb)
#> [1] "metadata"  "animals"   "receivers"
```

The result is a *list* (also a *glatos_workbook*) object with three elements containing data about the project and the data file (*metadata*), the fish that were tagged and released (*animals*), and the receivers (*receivers*). The *receivers* element is actually the result of merging two sheets in the source file: *deployments* and *recoveries*. Next, we will extract the receiver element from the workbook object and view its structure.

```
#extract receivers element from workbook list
rcv2 <- wb[["receivers"]]

#view structure
str(rcv2)
#> Classes 'glatos_receivers' and 'data.frame': 1039 obs. of  41 variables:
#> $ glatos_array : chr "BBI" ...
#> $ glatos_project : chr "HECWL" ...
```

```
#> $ station_no : chr "5" ...
#> $ consecutive_deploy_no : num 1 1 ...
#> $ ins_serial_no : chr "109493" ...
#> $ otn_array : chr NA ...
#> $ mooring_drop_dead_date : Date, format: NA ...
#> $ intend_lat : chr NA ...
#> $ intend_long : chr NA ...
#> $ otn_mission_id : chr NA ...
#> $ deploy_date_time : POSIXct, format: "2010-09-14 15:58:00" ...
#> $ deploy_lat : num 45.7 ...
#> $ deploy_long : num -84.4 ...
#> $ bottom_depth : num NA NA ...
#> $ riser_length : num NA NA ...
#> $ instrument_depth : num NA NA ...
#> $ checwlk_complete_time : chr NA ...
#> $ status_in : chr NA ...
#> $ ins_model_no : chr "VR2W" ...
#> $ glatos_ins_frequency : chr "69" ...
#> $ rcv_modem_address : chr NA ...
#> $ sync_date_time : POSIXct, format: NA ...
#> $ memory_erased_at_deploy : chr NA ...
#> $ rcv_battery_install_date : Date, format: NA ...
#> $ rcv_expected_battery_life: chr NA ...
#> $ rcv_voltage_at_deploy : chr NA ...
#> $ rcv_tilt_after_deploy : chr NA ...
#> $ deployed_by : chr NA ...
#> $ comments : chr NA ...
#> $ glatos_seasonal : chr "NO" ...
#> $ glatos_vps : chr "NO" ...
#> $ ar_confirm : chr NA ...
#> $ data_downloaded : chr NA ...
#> $ ins_model_number : chr NA ...
#> $ recovered : chr NA ...
#> $ recover_date_time : POSIXct, format: "2011-09-16 18:50:00" ...
#> $ recover_lat : num NA NA ...
#> $ recover_long : num NA NA ...
#> $ location_description : chr "Bois Blanc Island (East line)" ...
#> $ water_body : chr "Lake Huron" ...
#> $ glatos_region : chr "Lake Huron" ...
```

The result contains 41 columns and two classes: *glatos_receivers* and *data.frame*. Despite some differences between the structure of this project-specific data object and the network-level data object loaded in the previous example, both have been minimally modified to meet the requirements of any *glatos* function that accepts receiver data as input.

3.2.3 Other formats

Receiver location data in any format than one of the GLATOS standards will need to be:

1. loaded into R using some other function (e.g., *read_csv*, *fread*, etc) and
2. modified to ensure that all requirements of the desired function are met.

This vignette does not include an example of receiver location data loaded from CSV because the methods would be very similar to those described above. For example, you might step through each required column described in the Data Requirements vignette), check that each column meets *glatos* requirements, and modify accordingly using methods described above for detection data from a CSV file exported from VUE (see Other formats - CSV file exported from a VUE database)

4 Animal tagging and biological data

4.1 Requirements

There are currently no *glatos* functions that require animal tagging and biological data other than those columns present in the required *detection* data. Therefore, there are no formal requirements of such data in the package. Nonetheless, the *read_glatos_workbook* function can be used to facilitate loading animal tagging and biological data from a standard GLATOS project workbook (*.xlsm file) into an R session.

Use of the data loading function *read_glatos_workbook* will ensure that animal data are loaded efficiently and consistently among users, but can only be used on data in GLATOS format. Data in other formats will need to be loaded using other functions (e.g., *read.csv*, *fread*, etc.). Although there are currently no *glatos* requirements of animal data, any future requirements might be expected to be consistent with the *glatos_animals* class.

4.2 Examples

4.2.1 Loading GLATOS data (single project workbook)

The *read_glatos_workbook* function reads animal tagging and biological data from a standard GLATOS project workbook (*.xlsm file; *tagging* sheet). Data are read using *readWorkbook* in the *openxlsx* package, timestamps are formatted as class *POSIXct*.

We will again use data from the same *walleye_workbook.xlsm* example file used in the previous section (see data loading steps above), but will extract the *animals* element and view its structure.

```
#extract animals element from workbook list
fsh <- wb[["animals"]]

#view structure
str(fsh)
#> Classes 'glatos_animals' and 'data.frame':   543 obs. of  55 variables:
#> $ animal_id : chr "120" ...
#> $ tag_type : chr NA ...
#> $ tag_manufacturer : chr "VEMCO" ...
#> $ tag_model : chr "V16-4x" ...
#> $ tag_serial_number : chr "1106553" ...
#> $ tag_id_code : chr "32024" ...
#> $ tag_code_space : chr "A69-9001" ...
#> $ tag_implant_type : chr "internal" ...
#> $ tag_activation_date : Date, format: NA ...
#> $ est_tag_life : chr "1338" ...
#> $ tagger : chr NA ...
#> $ tag_owner_pi : chr NA ...
#> $ tag_owner_organization : chr NA ...
#> $ common_name_e : chr "walleye" ...
#> $ scientific_name : chr "Sander vitreus" ...
#> $ capture_location : chr "Maumee River" ...
#> $ capture_latitude : num 41.6 ...
#> $ capture_longitude : num -83.6 ...
#> $ wild_or_hatchery : chr NA ...
#> $ stock : chr NA ...
```

```
#> $ length : num 0.627 0.706 ...
#> $ weight : num NA NA ...
#> $ length_type : chr "total" ...
#> $ age : chr "7" ...
#> $ sex : chr "F" ...
#> $ dna_sample_taken : chr NA ...
#> $ treatment_type : chr NA ...
#> $ release_group : chr NA ...
#> $ release_location : chr "Maumee" ...
#> $ release_latitude : num 41.6 ...
#> $ release_longitude : num -83.6 ...
#> $ utc_release_date_time : POSIXct, format: "2011-03-28 04:00:00" ...
#> $ capture_depth : num NA NA ...
#> $ temperature_change : num NA NA ...
#> $ holding_temperature : num NA NA ...
#> $ surgery_location : chr "Maumee" ...
#> $ date_of_surgery : Date, format: NA ...
#> $ surgery_latitude : num 43.6 ...
#> $ surgery_longitude : num -84.2 ...
#> $ sedative : chr NA ...
#> $ sedative_concentration : chr NA ...
#> $ anaesthetic : chr NA ...
#> $ buffer : chr NA ...
#> $ anaesthetic_concentration : chr NA ...
#> $ buffer_concentration_in_anaesthetic : chr NA ...
#> $ anesthetic_concentration_in_recirculation: chr NA ...
#> $ buffer_concentration_in_recirculation : chr NA ...
#> $ dissolved_oxygen : chr NA ...
#> $ comments : chr NA ...
#> $ glatos_project : chr "HECWL" ...
#> $ glatos_external_tag_id1 : chr "5017" ...
#> $ glatos_external_tag_id2 : chr "5016" ...
#> $ glatos_tag_recovered : chr "NO" ...
#> $ glatos_caught_date : Date, format: NA ...
#> $ glatos_reward : chr NA ...
```

The result contains 55 columns and two classes: *glatos_animals* and *data.frame*.

4.2.2 Other formats

Receiver location data in any format than one of the GLATOS standards will need to be:

1. loaded into R using some other function (e.g., *read_csv*, *fread*, etc) and
2. modified to ensure that all requirements of the desired function are met.

This vignette does not include an example of animal tagging and biological data loaded from CSV because the methods would be very similar to those described above. Moreover, there are currently no *glatos* functions that require animal tagging and biological data other than those present in *glatos_detections* data. Although the *glatos* package currently does not contain any specific requirements of animal tagging and biological data, future requirements might be expected to resemble key columns of *glatos_animals* objects.

5 Transmitter specification data

5.1 Requirements

There are currently no *glatos* functions that require transmitter specification data. Therefore, there are no formal requirements of such data in the package. Nonetheless, the *read_vemco_tag_specs* function can be used to facilitate loading transmitter specification data from a standard VEMCO tag spec (*.xls) file provided to tag purchasers from VEMCO.

Use of the data loading function *read_vemco_tag_specs* will ensure that transmitter specificaiton data are loaded efficiently and consistently among users, but can only be used on data in VEMCO standard format. Data in other formats will need to be loaded using other functions (e.g., *read.csv*, *fread*, etc.). Although there are currently no *glatos* requirements of transmitter specification data, any future requirements might be expected to be consistent with the output of *read_vemco_tag_specs*.

5.2 Examples

5.2.1 Loading data from a VEMCO tag specs file

The *read_vemco_tag_specs* function reads transmitter specification data from a standard VEMCO tag specs (*.xls file). Data are read using *read_excel* in the *readxl* package.

We will get the path to the *lamprey_tag_specs.xls* (example included in the *glatos* package) using *system.file*, then read the data using *read_vemco_tag_specs*, and view the structure of the result.

```
#get path to example lamprey_tag_specs.xls file
spec_file <- system.file("extdata", "lamprey_tag_specs.xls",
                         package = "glatos")

#read lamprey_tag_specs.xls using 'read_vemco_tag_specs'
my_tags <- read_vemco_tag_specs(spec_file, file_format = "vemco_xls")

#view structure
class(my_tags)
#> [1] "list"
names(my_tags)
#> [1] "specs"    "schedule"
```

The result is a *list* object with two elements containing data about the transmitter specifications (*specs*) and the operating schedule (*schedule*). Next, we will view the structure of each element, starting with *specs*.

```
#view structure of specs element
str(my_tags$specs)
#> 'data.frame':    4 obs. of  18 variables:
#> $ sales_order : chr "11189" ...
#> $ serial_number : chr "1109712" ...
#> $ manufacturer : chr "Vemco" ...
#> $ model : chr "V9-2x-069k-1" ...
#> $ id_count : num 1 1 ...
#> $ code_space : chr "A69-1601" ...
#> $ id_code : chr "1362" ...
#> $ n_steps : int 1 1 ...
#> $ sensor_type : chr NA ...
#> $ sensor_range : chr NA ...
#> $ sensor_units : chr NA ...
#> $ sensor_slope : num NA NA ...
#> $ sensor_intercept : num NA NA ...
```

```
#> $ accel_algorithm : chr NA ...
#> $ accel_sample_rate : num NA NA ...
#> $ sensor_transmit_ratio: num NA NA ...
#> $ est_battery_life_days: num NA NA ...
#> $ battery_life_stat : chr "95%" ...
```

The result contains 18 columns of transmitter characteristics that do not change over time.

```
#view structure of schedule element
str(my_tags$schedule)
#> 'data.frame':    16 obs. of  11 variables:
#> $ serial_number : chr "1109712" ...
#> $ code_space : chr "A69-1601" ...
#> $ id_code : chr "1362" ...
#> $ step : int 1 2 ...
#> $ next_step : chr "2" ...
#> $ status : chr "ON" ...
#> $ duration_days : num 101 365 ...
#> $ power : chr "H" ...
#> $ min_delay_secs : num 40 NA ...
#> $ max_delay_secs : num 80 NA ...
#> $ accel_on_time_secs: num NA NA ...
```

The result contains 11 columns of transmitter characteristics that change over time. These may be used to estimate the operating characteristics (e.g., *power*, *min_delay*, *max_delay*, etc.) on a specific date following activation or release.

5.2.2 Other formats

Transmitter specification and schedule data in any format than one of the GLATOS standards will need to be:

1. loaded into R using some other function (e.g., *read_csv*, *fread*, etc) and
2. modified to ensure that all requirements of the desired function are met.

This vignette does not include an example of transmitter specification and schedule data loaded from CSV because the methods would be very similar to those described above. Moreover, there are currently no *glatos* functions that require transmitter specification and schedule data other than those present in *glatos_detections* data. Although the *glatos* package currently does not contain any specific requirements of these data, future requirements might be expected to resemble key columns of the output of *read_vemco_tag_specs*.