Package: wcUtils (via r-universe)

November 3, 2024

Title Utilities to Access the Wildlife Computers Data Portal API

Version 0.1

Description This package provides basic access and capabilities for interacting with and downloading data from the Wildilfe Computers Data Portal API. Where appropriate functions to specify data types and tidy the data are employed.

Depends R (>= 3.1.1)

License CC0 | file LICENSE

LazyData true

Imports lubridate, readr, magrittr, tidyr, dplyr, httr, digest, XML, xml2, jsonlite, data.table, janitor, fs, rlang, glue

RoxygenNote 7.2.3

Suggests testthat

Repository https://ocean-tracking-network.r-universe.dev

RemoteUrl https://github.com/noaa-afsc/wcUtils

RemoteRef HEAD

RemoteSha 150d1047e569a8ef4c367bd374b47e6bc16312b0

Contents

`as_ecdf` *Create An ECDF List*

Description

This creates a standardized list for holding ECDF data and sets the class to ‘ecdf‘. The S3 class, *wcECDF* consists of three named objects: ‘type‘, ‘percent_time‘, and ‘ecdf‘. The ‘type‘ object is the same as that specified for the ‘type‘ parameter. The ‘percent_time‘ object holds the proportion of the summary period the tag spent within the ‘shallow‘ or ‘deep‘ portion of the water column. Lastly, the ‘ecdf‘ object is a data.frame with a column, ‘ecd_prop‘, that specifies the proportional values for the ECDF and a column, ‘depth_break‘, that reports the corresponding depth value (in meters).

Usage

```
as_ecdf(type, ecdf)
```

Arguments

`type`	this is either ‘shallow‘ or ‘deep‘
`ecdf`	a data.frame

Value

A list with three named objects (‘type‘,‘percent_time‘,‘ecdf‘) of class ‘wcECDF‘

`combine_ecdf` *Combine Two ‘wcECDF‘ Objects Into a Single ECDF*

Description

Combine Two ‘wcECDF‘ Objects Into a Single ECDF

Usage

```
combine_ecdf(ecdf1, ecdf2, return = "pdf")
```

Arguments

`ecdf1`	object of class ‘wcECDF‘; typically the *shallow* region
`ecdf2`	object of class ‘wcECDF‘; typically the *deep* region

Value

four-column data.frame with columns ‘ecd_prop‘, ‘pdf‘, ‘prob‘, and ‘depth_break‘

`drop_ecd_duplicates` *Drop duplicates within an ECDF*

Description

Drop duplicates within an ECDF

Usage

```
drop_ecd_duplicates(e)
```

Arguments

`e`

`fixCSV` *Tidy a comma separated value (CSV) file*

Description

Tidies up a Comma Separated Value (CSV) file, ensuring that each row of the table in the file contains the same number of commas, and no empty rows are left below the table.

Usage

```
fixCSV(file, skip = 0, overwrite = FALSE)
```

Arguments

`file`	character: the name of the CSV file to be 'fixed'.
`skip`	integer: the number of lines in the CSV file to skip before the header row of the table. The skipped lines are copied directly to the output file unchanged. The default is `skip=0`, implying that the header row is the first row of the CSV file.
`overwrite`	logical: Write output to a separate, 'FIXED' file (`overwrite=FALSE`, the default), or overwrite the original file (`overwrite=TRUE`)? If `overwrite=TRUE`, the original file is copied to a `.BAK` file before being overwritten.

Details

`fixCSV` tidies up a Comma Separated Value (CSV) file to ensure that the CSV file contains a strictly rectangular block of data for input into R (ignoring any preliminary comment rows via the `skip=` argument).

CSV formatted files are a plain text file format for tabular data, in which cell entries in the same row of a table are separated by commas. When such files are exported from other applications such as spreadsheet software, the software has to decide whether any empty cells to the right-hand side of, or below, the table or spreadsheet should be represented by trailing commas in the CSV file. Such decisions can result in a 'ragged' table in the CSV file, in which some rows contain fewer commas ('short rows') or more commas ('long rows') than others, or where empty rows below the table are included as comma-only rows in the CSV file.

While R's `read.table` and related functions can sensibly extend short rows as needed, ragged tables in a CSV file can still result in errors, unwanted empty rows (below the table) or unwanted columns (to the right of the table) when the data is loaded into R.

`fixCSV` reads in a specified CSV file and removes or adds commas to rows, to ensure that each row in the body of the table contains the same number of cells as the header row of the table. Any empty rows below the table are also removed. The resulting table is then written back to file, either to a new file with 'FIXED' added to the filename (argument `overwrite=FALSE`, the default) or overwriting the original file (`overwrite=TRUE` - the original file is copied to a `.BAK` file before being overwritten).

Note that:

- The table of data in the CSV file *must* contain a header row of the correct length, since this row is used to determine the correct number of columns for the table. Note: if this header row is too short, then subsequent rows will be truncated to match the length of the header, so beware. Misspecification of the `skip=` argument (see below) can similarly lead to such corruption of the 'fixed' file.
- In the header row, any trailing commas representing empty cells to the right of the (non-empty) header entries are first removed before determining the correct number of columns for the table. Thus the length of the header row (and hence the assumed width of the entire table) is determined by the *right-most non-empty cell* in the header row.
- `fixCSV` does not remove empty cells, rows or columns within the interior (or on the left side) of the table - it is concerned only with the right and bottom boundaries of the table.
- A `skip=` argument is included to tell `fixCSV` to ignore the specified number of comment rows preceding the header row. Such rows are simply copied over into the output file unchanged. The default for this parameter is `skip=0`, so that the first row in the data file is assumed to be the header row. As noted above, misspecification of this argument can seriously corrupt the output.
- `fixCSV` can overwrite your data file(s) (via `overwrite=TRUE`), and althought it makes a backup of your original file, you should still make sure that you have a separate backup of your data file in a safe place before using this function! The author of this code takes no responsibility for any data loss or corruption as a result of the use of this routine...

Author(s)

Alexander Zwart (alec.zwart at csiro.au)

Examples

```
## Not run:

## Assuming CSV file 'alleleDataFile.csv' exists in the current
## directory.  The following overwrites the CSV file - make sure
## you have a backup!

fixCSV("alleleDataFile.csv",overwrite=TRUE)


## End(Not run)
```

`format_bins` *format bin labels*

Description

format bin labels

Usage

```
format_bins(bins)
```

Arguments

bins	a vector of bin labels to be formated

Value

a vector of formated bin labels

---

make_names *Expanded make.names function for creating consistent column names*

Description

Expanded make.names function for creating consistent column names

Usage

```
make_names(x)
```

Arguments

x	data frame with columns to be renamed

Value

a data frame

---

read_allmsg *Parse a *-All.csv file into a proper data.frame*

Description

Parse a *-All.csv file into a proper data.frame

Usage

```
read_allmsg(allmsg_file, to_lower = TRUE, fix_csv = FALSE)
```

Arguments

allmsg_file	file path or file connection to a *-All.csv file
to_lower	whether to convert the column names to lower case
fix_csv	whether to attemtp to fix any comma, csv issues

Value

a data frame

`read_behav` *Parse a *-Behavior.csv file into a proper data.frame*

Description

Parse a *-Behavior.csv file into a proper data.frame

Usage

```
read_behav(behav_file, to_lower = TRUE, fix_csv = FALSE)
```

Arguments

`behav_file`	file path or file connection to a *-Behavior.csv file
`to_lower`	whether to convert the column names to lower case
`fix_csv`	whether to attempt to fix any comma, csv issues

Value

a data frame

`read_ecdf` *Read and Tidy ECDF Data File*

Description

This function is the workhorse of the ‘wcECDF‘ package. All ECDF data are stored within a ‘*-ECDHistos.csv‘ file that is output from either the Wildlife Computers Data Portal or DAP processing software. The function presumes the ‘*-ECDHistos.csv‘ data file is provided as-is from these sources and has not been edited. The resulting output is a nested ‘tibble‘ that adheres to tidy data principles and includes new columns (‘shallow_ecdf‘, ‘deep_ecdf‘, ‘full_ecdf‘, and ‘full_pdf‘.

Usage

```
read_ecdf(ecdf_csv)
```

Arguments

`ecdf_csv`	file path for the ‘*-ECDHistos.csv‘

Details

In addition to *tidying* up the original data into a more workable *long* format, this function calculates four new columns.

shallow_ecdf The ‘shallow_ecdf‘ column is a list-col that contains nested S3 objects of class ‘wcECDF‘ representing the portion of the water column defined as ‘shallow‘.

deep_ecdf The ‘deep_ecdf‘ column is a list-col that contains nested S3 objects of class ‘wcECDF‘ representing the portion of the water column defined as ‘deep‘.

full_ecdf The combined ECDF for both shallow and deep regions. The resulting ECDF is weighted based on the reported proportion of time spent within each region.

full_pdf The ‘full_ecdf‘ is transformed into a probability density function and two columns are returned: ‘pdf‘ and ‘prob‘. The later represents the probability the tag spent time at a given depth.

Value

A nested tibble

`read_fastGPS` *Parse a *-FastGPS.csv files into a proper data.frame*

Description

Parse a *-FastGPS.csv files into a proper data.frame

Usage

```
read_fastGPS(gps_file, to_lower = TRUE, fix_csv = FALSE)
```

Arguments

`gps_file`	file path or file connection to a *-FastGPS.csv file
`to_lower`	whether to convert the column names to lower case
`fix_csv`	whether to attemtp to fix any comma, csv issues

Value

a data frame

read_histos *Parse a *-Histos.csv files into a proper data.frame*

Description

Parse a *-Histos.csv files into a proper data.frame

Usage

```
read_histos(
  histo_file,
  to_lower = TRUE,
  dt_fmt = "%H:%M:%S %d-%b-%Y",
  fix_csv = FALSE
)
```

Arguments

histo_file	file path or file connection to a *-Histos.csv file
to_lower	whether to convert the column names to lower case
dt_fmt	format for the Date column
fix_csv	whether to attemtp to fix any comma, csv issues

Value

a list of two data frames

read_locs *Parse a *-Locations.csv files into a proper data.frame*

Description

Parse a *-Locations.csv files into a proper data.frame

Usage

```
read_locs(loc_file, fix_csv = FALSE)
```

Arguments

loc_file	file path or file connection to a *-Locations.csv file
fix_csv	whether to attemtp to fix any comma, csv issues

Value

a data frame

smooth_ecdf *Create smoothed ECDF*

Description

Create smoothed ECDF

Usage

```
smooth_ecdf(ecdf, bin.width)
```

Arguments

```
bin.width
```

tidyDiveDepths *Apply 'tidy' data principles to the dive-depth histogram data stream*

Description

`tidyDiveDepths` returns a 'tidy'd' data frame of dive depth data

Usage

```
tidyDiveDepths(histos)
```

Arguments

`histos`	a list returned from `read_histos`

Details

The histogram data stream is provided in a 'wide' format (each row represents a time period and the observed values are provided in 1 to 72 'bin' columns). This format can be difficult to work with in R and other data analysis platforms (e.g. database tables), so we use the `tidyr` and `dplyr` packages to manipulate the data into a more flexible, 'narrow' format. This results in a data structure where every row represents a single observation.

This is implemented, here, with the dive depth data. For dive depth data, the tag records the maximum depth experienced during a qualifying dive and tallies those dives into user-sepcified depth bins and user-specified time bins. This, unlike with timeline data, requires some knowledge of these user-specified bins. As long as the user has uploaded a configuration/report file to the Wildlife Computers Data Portal, then the *-Histos.csv file provides information on the dive depth bins. If the bin information is not available, the function will produce a warning and output files with generic 'Bin' labels.

Value

a data frame with tidy, narrow data structure and actual dive depths bin limits (when provided)

`tidyDiveDurations` *Apply 'tidy' data principles to the dive-duration histogram data stream*

Description

`tidyDiveDurations` returns a 'tidy'd' data frame of dive duration data

Usage

```
tidyDiveDurations(histos)
```

Arguments

`histos`	a list returned from `read_histos`

Details

The histogram data stream is provided in a 'wide' format (each row represents a time period and the observed values are provided in 1 to 72 'bin' columns). This format can be difficult to work with in R and other data analysis platforms (e.g. database tables), so we use the `tidyr` and `dplyr` packages to manipulate the data into a more flexible, 'narrow' format. This results in a data structure where every row represents a single observation.

This is implemented, here, with the dive duration data. For dive uration data, the tag records the duration in seconds of a qualifying dive and tallies those durationso user-sepcified duration bins and user-specified time bins. This, unlike with timeline data, requires some knowledge of these user-specified bins. As long as the user has uploaded a configuration/report file to the Wildlife Computers Data Portal, then the *-Histos.csv file provides information on the dive durations. If the bin information is not available, the function will produce a warning and output files with generic 'Bin' labels.

Value

a data frame with tidy, narrow data structure and actual dive duration bin limits (when provided)

tidyTimeAtDepth *Apply 'tidy' data principles to the time-at-depth histogram data stream*

Description

`tidyTimeAtDepth` returns a 'tidy'd' data frame of time-at-depth data

Usage

```
tidyTimeAtDepth(histos)
```

Arguments

`histos`	a list returned from `read_histos`

Details

The histogram data stream is provided in a 'wide' format (each row represents a time period and the observed values are provided in 1 to 72 'bin' columns). This format can be difficult to work with in R and other data analysis platforms (e.g. database tables), so we use the `tidyr` and `dplyr` packages to manipulate the data into a more flexible, 'narrow' format. This results in a data structure where every row represents a single observation.

This is implemented, here, with the time-at-depth data. For time-at-depth data, the tag records the portion of time the tag spent within user-defined dive depth bins. This, unlike with timeline data, requires some knowledge of these user-specified bins. As long as the user has uploaded a configuration/report file to the Wildlife Computers Data Portal, then the *-Histos.csv file provides information on the time-at-depth bins. If the bin information is not available, the function will produce a warning and output files with generic 'Bin' labels.

Value

a tibble with tidy, narrow data structure and actual time-at-depth bin limits (when provided)

tidyTimelines *Apply 'tidy' data principles to the timeline histogram data stream*

Description

`tidyTimelines` returns a 'tidy'd' data frame of timeline data

Usage

```
tidyTimelines(histos, row_min = 1)
```

Arguments

histos	a list returned from read_histos
row_min	user defined minimum timeline records required; row_min = 2 will return NULL unless at least 1 records are found.

Details

The histogram data stream is provided in a 'wide' format (each row represents a time period and the observed values are provided in 1 to 72 'bin' columns). This format can be difficult to work with in R and other data analysis platforms (e.g. database tables), so we use the tidyr and dplyr packages to manipulate the data into a more flexible, 'narrow' format. This results in a data structure where every row represents a single observation.

This is implemented, here, with the timeline data. For timeline data, tag 'dryness' is provided as either a percentage of each hour the tag was dry or as a binary (1 or 0) value representing whether a tag was dry for a majority of a given 20-minute period. For both of these situations, the values for the 'bin' columns are predictable and we can, in addition to tidying the data structure, also turn the bin values into actual time periods.

Value

a data frame with tidy, narrow data structure and actual time periods in place of bins

---

to_pdf *Transform a 'wcECDF' Object Into a Probability Density Function*

Description

Transform a 'wcECDF' Object Into a Probability Density Function

Usage

```
to_pdf(ecdf)
```

Arguments

ecdf	two-column data.frame with 'ecdf_prop' and 'depth_break'

Value

a two-column data.frame with 'ecdf_prop' and 'depth_break'

wcGetDeployID — *Return a vector of data portal unique IDs associated with a given DeployID*

Description

wcGetDeployID returns a vector of data portal unique ID(s)

Usage

```
wcGetDeployID(xml_content, deployid = NULL)
```

Arguments

xml_content	XML content/data returned from wcPOST (with 'action=get_deployments')
deployid	valid deployid character (required)

Details

This function presumes a DeployID has been setup for deployments on the Wildife Computers Data Portal. The vector of deployment ID(s) returned will be a subset that match the deployid character in the function call. The list returned will also include a simple data frame with summary information one can use to determine the appropriate id.

Value

a list with ids (a vector of deployment ids) and a df (data frame of deployment summaries)

wcGetDownload — *Retrieve data from Wildlife Computers Data Portal*

Description

wcGetDownload will return a list of data frames containing deployment data

Usage

```
wcGetDownload(
  id,
  wc.key = Sys.getenv("WCACCESSKEY"),
  wc.secret = Sys.getenv("WCSECRETKEY"),
  keyfile = NULL,
  tidy = TRUE
)
```

Arguments

id	a single character representing a data portal unique deployment identifier
tidy	whether to tidy the histogram data stream and create a timelines output

Details

The Wildlife Computers Data Portal will return deployment data in the form of a zipped file with various comma-separated files and other accessory files. The *.csv files correspond to particular data streams. This function, currently, focuses on the locations, behavior, histograms, timelines, status, and messages data streams.

For most of the files, the data are read in with `read.csv` and, other than a few steps to set the data types, the data are provided 'as is'. The one exception is the histogram data stream. Here, we use the `tidyr` and `dplyr` package to 'tidy' the data into a more appropriate data structure. For now, this is only implemented with timeline data and the 'tidy'd' data is provided within the list element `$timelines`

Value

a list of data frames with up to 6 named elemnts (locations, behavior, histograms, status,timelines,messages)

---

wcGetIDs *Return a vector of deployment IDs*

Description

`wcGetIDs` returns a vector of deployment IDs

Usage

```
wcGetIDs(xml_content, xpath = NULL)
```

Arguments

xml_content	XML content/data returned from wcPOST (with 'action=get_deployments')
xpath	additional customization possible by passing an xpath statement

Details

Each deployment in the Wildlife Computers Data Portal is identified by a unique alpha-numeric value. This function searches the XML response data and extracts those IDs

Value

returns a vector of deployment IDs

wcGetProjectIDs *Return a vector of deployment IDs associated with a given Project*

Description

wcGetProjectIDs returns a vector of deployment IDs

Usage

```
wcGetProjectIDs(xml_content, project = NULL)
```

Arguments

xml_content	XML content/data returned from wcPOST (with 'action=get_deployments')
project	valid project name (required)

Details

This function presumes a custom label,'Project', has been setup for deployments on the Wildife Computers Data Portal. The vector of deployment IDs returned will be a subset that match the project name provided in the function call.

Value

returns a vector of deployment IDs

wcGetPttID *Return a vector of deployment ID associated with a given PTT*

Description

wcGetPttID returns a vector of deployment ID(s)

Usage

```
wcGetPttID(xml_content, ptt = NULL)
```

Arguments

xml_content	XML content/data returned from wcPOST (with 'action=get_deployments')
ptt	valid ptt integer (required)

Details

This function presumes a PTT has been setup for deployments on the Wildife Computers Data Portal. The vector of deployment ID(s) returned will be a subset that match the ptt integer provided in the function call. The list returned will also include a simple data frame with summary information one can use to determine the appropriate id.

Value

a list with ids (a vector of deployment ids) and a df (data frame of deployment summaries)

---

wcGetRecentIDs *Return a vector of deployment IDs with new data in the last* n *days*

Description

`wcGetProjectIDs` returns a vector of deployment IDs

Usage

```
wcGetRecentIDs(xml_content, days = 14)
```

Arguments

`xml_content`	XML content/data returned from wcPOST (with 'action=get_deployments')
`days`	integer value specifying the time window from `now()` in days

Details

This returns a subset of deployment IDs with new data available on the portal within the last n days. The default value is for 14 days

Value

returns a vector of deployment IDs

wcGetZip *Retrieve a single zip file from Wildlife Computers Data Portal*

Description

wcGetZip will return a path to a downloaded zip file

Usage

```
wcGetZip(
  id,
  wc.key = Sys.getenv("WCACCESSKEY"),
  wc.secret = Sys.getenv("WCACCESSKEY"),
  keyfile = NULL
)
```

Arguments

wc.key	public access key (default retrieves from option value set in .Renviron)
wc.secret	secret access key (default retrieves from option value set in .Renviron)
keyfile	path to a json formatted keyfile with WCACCESSKEY and wcSecretKey

Details

The Wildlife Computers Data Portal will return deployment data in the form of a zipped file with various comma-separated files and other accessory files.

Value

a path to the zip file

wcPOST *Send POST to Wildlife Computers Data Portal API*

Description

wcPOST returns a response from a POST to the API

Usage

```
wcPOST(
  wc.key = Sys.getenv("WCACCESSKEY"),
  wc.secret = Sys.getenv("WCSECRETKEY"),
  keyfile = NULL,
  params = "action=get_deployments"
)
```

Arguments

wc.key	public access key (default retrieves from option value set in .Renviron)
wc.secret	secret access key (default retrieves from option value set in .Renviron)
keyfile	path to a json formatted keyfile with WCACCESSKEY and WCSECRETKEY
params	POST message (default returns a list of deployments)

Details

This function provides basic access to the API via POST. The params value contains the string to include in the body of the POST. The default action is to return a list of deployments associated with your account . Most users will likely not call this function directly, but instead, rely on other helper/wrapper functions within the package.

The Wildlife Computers Data Portal API uses a form of keyed-hash message authentication code for secure access. Your 'Access Key' and 'Secret Key' can be obtained from the data portal website (Account Settings > Web Services Security). For security reasons you should NOT include the keys as plain text in any scripts. Instead, include the key values as within your .Renviron.

Value

an httr response object is returned. Content of the response can be obtained with the httr::content() function.

Setting key values within .Renviron

This option is a preferred option for storing keys, passwords and other sensitive values. Your .Renviron should be secured via OS security/permissions (e.g. on Linux/OS X .Renviron is stored within the home directory which is only accessible by an authorized user. However, you should not share your .Renviron or include your .Renviron in version control (e.g. git) if you use this option. An alternative is to read values from a different file in the home directory or to use OS level environment variables.

preformatted WCACCESSKEY = 'E4iZhsfdje7590JDNR/VARTEZyhfwb84485X5Xw86ow=' WCSECRETKEY = 'WIRJFYhfjdsuSEqKoE7WSDvXUHzVP0pHDJSscmeA7fw='

---

wcUtils *wcUtils: A package for import/export of Wildlife Computers tag data*

Description

wcUtils provides functionality for working with data from select (SPLASH, SPOT) Wildlife Computers satellite telemetry tags. The package relies on data files produced by the Wildlife Computers DAP program or downloaded from the Wildlife Computers Data Portal. Data are read into R

wcUtils functions

- read_behav
- read_histos
- read_locs
- tidyDiveDepths
- tidyDiveDurations
- tidyTimeAtDepth
- tidyTimelines
- wcGetDownload
- wcGetIDs
- wcGetProjectIDs
- wcGetRecentIDs
- wcPOST

Index