

Package ‘rextor’

September 10, 2026

Title Prepare 'WEXTOR' Data

Version 1.2.0

Depends R (>= 4.1.0)

Description Facilitate data preparation for data collected on 'WEXTOR' <<https://wextor.eu>>, created by Reips and Neuhaus (2002) <[doi:10.3758/bf03195449](https://doi.org/10.3758/bf03195449)>. Perform plausibility and other checks and make use of cool color palettes and themes for data visualization.

License GPL (>= 3)

Date 2026-09-09

Encoding UTF-8

Imports dplyr, readr, ggplot2

Suggests knitr, rmarkdown, rlang, scales, tibble

VignetteBuilder knitr

URL <https://github.com/iscience-kn/rextor>,
<https://iscience-kn.github.io/rextor/>

BugReports <https://github.com/iscience-kn/rextor/issues>

Config/roxygen2/version 8.0.0

RoxygenNote 7.3.2

NeedsCompilation no

Author Annika Tave Overlander [aut, cre] (ORCID:
<<https://orcid.org/0009-0006-8373-4086>>),
Ulf-Dietrich Reips [ths, cph] (ORCID:
<<https://orcid.org/0000-0002-1566-4745>>)

Maintainer Annika Tave Overlander <annika-tave.overlander@uni.kn>

Repository CRAN

Date/Publication 2026-09-10 15:30:10 UTC

Contents

ask_rex	2
BiFiX_data_raw	3
check_ok	3
fake_wextor_log	4
getmode	5
getperc	6
informed_consent	6
ip_check	7
namepref	8
namepref0	9
path_to_file	9
plausicheck	10
read_WEXTOR	11
read_WEXTOR_log	12
removepref	14
scale_rextor	14
serious_check	15
sess_length_check	16
theme_wob	17
tidy_WEXTOR_log	18
Index	20

ask_rex	<i>Ask Rex</i>
---------	----------------

Description

This function lets you ask dino Rex for help. You can try to ask for specific functions and topics or just use the function as-is for some motivational words of encouragement. :)

Usage

```
ask_rex(topic = "?")
```

Arguments

topic	What you want to know about; either leave it blank for motivation or specify 'ttest', 'mean'... Input must be in quotes!
-------	--

Value

Returns advice and explanations on R; sometimes Rex asks follow-up questions

Examples

```
# Get motivation
ask_rex()

# Get specific information
ask_rex("mean")
ask_rex(topic = "ttest")
```

BiFiX_data_raw	<i>BiFiX - Big Five raw data</i>
----------------	----------------------------------

Description

Raw data from the BiFiX study

Format

A data frame with 760 rows and 81 columns:

\$validation_var validation variable

.wx.1.id ID

.wx.3.experimental_condition Experimental condition

big_... Big Five items (A - Agreeableness, C - Conscientiousness, E - Extraversion, N - Neuroticism, O - Openness) ...

Source

<https://github.com/iscience-kn/BiFiX>

check_ok	<i>Check whether a plausibility condition is satisfied</i>
----------	--

Description

Returns a logical vector indicating whether a given check column equals "ok". If the column does not exist in the data, all cases are treated as passing.

Usage

```
check_ok(data, col)
```

Arguments

<code>data</code>	A data frame containing plausibility check variables.
<code>col</code>	A character string specifying the column name to check for "ok" entries (case-sensitive).

Details

Used inside the `plausicheck()` function.

Value

A logical vector of length `nrow(data)`.

Examples

```
example_df <- data.frame(check_ip = c("ok", "nope", "ok", "OK"))
check_ok(example_df, "check_ip")
```

<code>fake_wextor_log</code>	<i>Fake WEXTOR log</i>
------------------------------	------------------------

Description

Fake example of what a WEXTOR log file looks like for demonstration. In most cases, data will be exported by using "Get Data" on WEXTOR. But sometimes, when there are doubts about data quality or any kinds of anomalies when analyzing data from an online study, it can make sense to check the server log file.

Format

A log file containing fake data from an example study

1. **process_id** – Apache worker/process id (%P)
2. **date** – request date, mm/dd/yy
3. **time** – request time, HH:MM:SS
4. **connection_status** – Apache connection status (%c: "+" keep-alive, "-" closed, "X" aborted)
5. **client_ip** – requesting IP address (%h)
6. **url** – requested path, host/researcher/study/page (%U)
7. **bytes_sent** – response size in bytes (%B)
8. **user_agent** – browser/client user-agent string
9. **referer** – the URL the request came from
10. **time_taken_s** – time to serve the request, in whole seconds (%T)
11. **query_string** – the raw GET query string (%q): the WEXTOR variables submitted together with this request

12. **duration_ms** – server response time in milliseconds. This field isn't part of WEXTOR's documented log format and its exact meaning is inferred from the data – treat it with mild caution and verify against your own server setup if it matters for your analysis.
13. **query** – the listed GET query: the WEXTOR variables submitted together with this request that can later be brought into the actual study-dataframe

Source

Generated by Claude AI to resemble a real WEXTOR log file

getmode	<i>Get the mode of a vector or variable</i>
---------	---

Description

The mode is the number that appears the most often in a vector or variable. This measure is especially important for nominal variables since the mean or median cannot meaningfully be reported.

Usage

```
getmode(v)
```

Arguments

v	a vector of variable in dataframe
---	-----------------------------------

Value

Returns the mode of v, i.e., the value occurring most often

Examples

```
vec <- c("A", "A", "B")  
getmode(vec)
```

getperc	<i>Get percentage of a value</i>
---------	----------------------------------

Description

Get percentage of a value

Usage

```
getperc(data, var, val)
```

Arguments

data	A dataframe
var	A variable in the dataframe, written as a character in quotes, e.g. "var"
val	A value of variable var of which to get the percentage, also in quotes

Value

Returns the percentage of value val in relation to all values of var

Examples

```
getperc(data.frame(a = c("duck", "duck", "goose")),
  "a",
  "duck")

# Also works when val is indexed (alphabetically)
getperc(data.frame(a = c("duck", "duck", "goose")),
  "a",
  2)
```

informed_consent	<i>Informed Consent Filter</i>
------------------	--------------------------------

Description

One of the best practices in data collection is including an informed consent question. With this function you can automatically filter out rows where participants did not explicitly consent to their data being used.

Usage

```
informed_consent(dataframe, varname = "informedconsent")
```

Arguments

dataframe	A dataframe of data collected with 'WEXTOR'
varname	character. Variable name of informed consent variable. Defaults to "informed-consent".

Value

Returns data with cases filtered out that did not check the informed consent box

Examples

```
data <- data.frame(informedconsent = c("checked", "not checked", "checked"))
informed_only <- informed_consent(data)
```

ip_check	<i>Check for duplicate IP addresses</i>
----------	---

Description

ip_check() identifies potential duplicate participations in a dataset by checking for repeated IP addresses. Cases with duplicated IPs are flagged as "possible duplicate", while unique IPs are labeled "ok". Keep in mind that in some cases, duplicate IP addresses should not lead to exclusion, for example when different participants (such as siblings or flatmates) plausibly use the same device.

Usage

```
ip_check(dataframe)
```

Arguments

dataframe	A data frame containing the study data. Must include a variable named ip representing participants' IP addresses.
-----------	---

Details

If WEXTOR-style prefixes (i.e. .wx.) are detected in the variable names, they are removed prior to performing the check.

The function uses duplicated() to flag repeated IP addresses. Only subsequent occurrences are marked as duplicates; the first instance of each IP is treated as valid.

Value

A data frame with an additional column check_ip, indicating whether each case has a unique IP ("ok") or is a potential duplicate ("possible duplicate").

Examples

```
data <- read_WEXTOR(path_to_file("BiFiX_data_raw.csv"))
# The example data does not contain real IPs (data protection), so we will use simulate ones
data$ip <- sample(1:1000, nrow(data), replace = TRUE)

new_data <- ip_check(data)
```

namepref

Add variable name prefix (with substitution)

Description

This function strips the old prefix from the variable names of the dataframe. It then adds the new prefix where the old one was and changes the dataframe's names to the new ones.

Usage

```
namepref(dataframe, pref_old, pref_new)
```

Arguments

dataframe	A dataframe
pref_old	Old prefix character that some or all variables in the dataframe have
pref_new	New prefix character to replace the old one

Value

Returns dataframe with those variables renamed that start with the old prefix.

Examples

```
bla <- tibble::tibble(x_ar = 1:5, y_ar = 6:10)
blo <- namepref(bla, "x_", "z_")
names(bla)
names(blo)
```

namepref0	<i>Add variable name prefix (without substitution)</i>
-----------	--

Description

This function adds the new prefix and changes the dataframe's names to the new ones.

Usage

```
namepref0(dataframe, pref_new)
```

Arguments

dataframe	A dataframe
pref_new	New prefix character to add to all variable names in the dataframe

Value

Returns dataframe with all variables renamed to start with the new prefix.

Examples

```
bla <- tibble::tibble(x_ar = 1:5, y_ar = 6:10)
blo <- namepref0(bla, "var")
names(bla)
names(blo)
```

path_to_file	<i>Get file path to example data file</i>
--------------	---

Description

This function makes the example data from the BiFiX study (submitted for publication in BRM) easy to access.

Usage

```
path_to_file(path = NULL)
```

Arguments

path	Name of file in quotes with extension; "BiFiX_data_raw.csv" will work. If NULL, the example file will be listed.
------	--

Value

Gives the path to the example file included in the package. Meant for use in `read_WEXTOR()`.

Source

This function is adapted from `palmerpenguins` (which is adapted from `readxl::readxl_example()`).

Examples

```
path_to_file()
path_to_file("BiFiX_data_raw.csv")
head(read.csv(path_to_file("BiFiX_data_raw.csv")))
```

plausicheck	<i>Plausibility check on WEXTOR study data</i>
-------------	--

Description

`plausicheck()` performs basic plausibility checks on a study dataset to identify potentially invalid or suspicious participation. The function can check whether participants visited a minimum number of pages, whether the recorded session length appears plausible, and whether IP addresses indicate duplicate participation.

Usage

```
plausicheck(dataframe, min_pages, check_sess_length = TRUE, check_ip = TRUE)
```

Arguments

dataframe	A data frame containing the study data (needs to contain variables <code>page_trail</code> for the trail of webpages in the study & <code>session_length</code> for the overall time each participant spent on the study (if session length is to be checked) and <code>ip</code> for participants' IP addresses (if IP is to be checked)).
min_pages	Numeric. The minimum number of pages a participant must have visited in the study for their participation to be considered plausible.
check_sess_length	Logical. Should the session length plausibility check be performed? Defaults to TRUE.
check_ip	Logical. Should the IP address plausibility check be performed? Defaults to TRUE.

Details

If WEXTOR prefixes are detected in the variable names, they are removed before the plausibility checks are applied.

Value

A data frame with additional plausibility check variables. The final variable `check_plausibility` indicates whether all selected checks were passed ("all ok") or whether the case should be excluded ("exclude"). Keep in mind that researchers are advised to make sure that the "exclude"-cases were correctly identified and are indeed of poorer data quality to avoid unnecessary data loss.

Examples

```
data <- read_WEXTOR(path_to_file("BiFiX_data_raw.csv"))
# The example data does not contain real IPs (data protection), so we will use simulated ones
data$ip <- sample(1:1000, nrow(data), replace = TRUE)

plausi_data <- plausicheck(dataframe = data,
  min_pages = 6,
  check_sess_length = TRUE,
  check_ip = TRUE)
```

read_WEXTOR

*Read 'WEXTOR' Data***Description**

With this function, you can easily read in your 'WEXTOR' generated data. By default, it will be exported as a "CSV" file, which stands for "comma-separated values". R has many available options of reading in this type of data, which can make it hard to navigate which one to use - especially for beginners.

Usage

```
read_WEXTOR(filepath, keep_validation = TRUE)
```

Arguments

<code>filepath</code>	Location of the WEXTOR CSV file on your computer.
<code>keep_validation</code>	Should the validation variable from WEXTOR be kept? TRUE by default.

Details

In the `read_WEXTOR()` function, you just need to provide the `filepath`, i.e. the location of the CSV data file that you downloaded from 'WEXTOR'. If you want, you can already decide whether to keep the first column containing a so-called validation variable, but you don't have to. With no other input, `read_WEXTOR` will keep the variable for you. You can explicitly set `keep_validation` to `FALSE` if you like your data neat and do not need this extra measure. The usage is explained in the examples down below using the open source BiFiX data with psychological Big Five personality traits.

This function prepares the 'WEXTOR' data so that it is readable by both R and you as a human. It will give you your original data and also make the start and end time of each participation easier to read and work with later (by default, 'WEXTOR' will return these values as date and time separately, cluttering your dataset).

Value

The WEXTOR data as an R data object.

Examples

```
data <- read_WEXTOR(path_to_file("BiFiX_data_raw.csv"))

# If you don't need the validation variable, try
data_noval<-read_WEXTOR(path_to_file("BiFiX_data_raw.csv"),
                        keep_validation = FALSE)
```

read_WEXTOR_log

Read a raw WEXTOR server log file

Description

Reads a raw .txt access log downloaded from a WEXTOR study (<https://wextor.eu>) into a data frame with one row per logged HTTP request, i.e. one row per page or asset served during the study. This can then be brought into a *tidy* (one participant per row) format using `tidy_WEXTOR_log`. If you suspect there may be double entries or other issues with the data, it can help to check the original log file. The tidy version also gets a new variable added that flags potential anomalies like a double IP address.

The log format WEXTOR writes is a tab-separated custom Apache LogFormat equivalent to: "%P %{m/%d/%y}t %{T}t %c %h %U %B User-Agent Referer %T %q", with one further, undocumented field appended after %q that appears to be the server's response time in milliseconds. Concretely, every line in the file has 12 tab-separated fields:

1. **process_id** – Apache worker/process id (%P)
2. **date** – request date, mm/dd/yy
3. **time** – request time, HH:MM:SS
4. **connection_status** – Apache connection status (%c: "+" keep-alive, "-" closed, "X" aborted)
5. **client_ip** – requesting IP address (%h)
6. **url** – requested path, host/researcher/study/page (%U)
7. **bytes_sent** – response size in bytes (%B)
8. **user_agent** – browser/client user-agent string
9. **referer** – the URL the request came from
10. **time_taken_s** – time to serve the request, in whole seconds (%T)

11. **query_string** – the raw GET query string (%q): the WEXTOR variables submitted together with this request
12. **duration_ms** – server response time in milliseconds. This field isn't part of WEXTOR's documented log format and its exact meaning is inferred from the data – treat it with mild caution and verify against your own server setup if it matters for your analysis.

date and time are combined into one timestamp column, and url is split into host, study_path and page for convenience. query_string is additionally parsed into individual name/value pairs and kept as a list-column, query – one small tibble per request, one column per WEXTOR variable that was submitted with it.

Note that WEXTOR logs each request's query string as only the variables submitted *from the previous page* (plus id), not the full cumulative set collected so far – so a single row is not a full participant record. To get one row per participant, use `tidy_WEXTOR_log()` from this package.

Repeated keys within one query string (which happens for checkbox variables WEXTOR logs once per click) are resolved by keeping the *last* value, i.e. the final state at submission.

Usage

```
read_WEXTOR_log(filepath, tz = "UTC", drop_assets = FALSE)
```

Arguments

filepath	Path to a WEXTOR log .txt file.
tz	Time zone to assign to the parsed timestamp. WEXTOR logs the server's local time, not necessarily UTC. Default "UTC"; set this to your server's actual time zone if absolute times matter.
drop_assets	If TRUE, drop requests for static assets (css/js/images/fonts) and keep only the study's own HTML pages. Default FALSE.

Value

A [tibble](#) with one row per logged HTTP request and the columns described above (plus host, study_path, page, timestamp, and the query list-column). Lines that don't split into exactly 12 tab-separated fields (e.g. a truncated trailing line) are dropped with a warning.

Examples

```
log <- read_WEXTOR_log(path_to_file("fake_wextor_log.txt"))
```

removepref	<i>Remove 'WEXTOR' variable name prefixes (automatic)</i>
------------	---

Description

This function strips the old prefix from the server variable names of the dataframe. It identifies such variables that start with ".wx.#." where # stands for any number, or simply ".wx." and then removes these prefixes for more easily legible and usable variable names in your data!

Usage

```
removepref(dataframe)
```

Arguments

dataframe	A dataframe, usefully containing variables that have the server prefix ".wx.#."
-----------	---

Value

Returns dataframe with those variables renamed that start with the old prefix.

Examples

```
bli <- tibble::tibble(.wx.1.ar = 1:5, y_ar = 6:10, .wx.z = 11:15)
blu <- removepref(bli)
names(bli)
names(blu)
```

scale_rextor	<i>Coloring Scale for Data Visualization</i>
--------------	--

Description

This scale is very versatile and powerful. It offers several color palettes to choose from and automatically recognizes whether to use a discrete or continuous scale depending on the data type of the mapping object. It can also be used for both color and fill aesthetics by defining aesthetic accordingly directly inside the function! All scales feature 12 unique colors to fit most data needs - if you have more than 12 categories to show, you might want to reconsider if all of them need plotting. Data Viz should always convey a message and simplify.

Usage

```
scale_rextor(pal = "cute", direction = 1, aesthetic = "color", ...)
```

Arguments

pal	Color palette. Use either numbers from 1 to 9 or one of the names: 'arby', 'cute', 'wextor', 'dualtone', 'colorful', 'protanomaly', 'forest', 'hadestown', 'neon'
direction	Direction of the color palette. Keep blank or use 1 for the default direction or use -1 to reverse the colors.
aesthetic	Which aesthetic to use the scale on; either color or fill, defaults to color.
...	Other common scale parameters.

Value

ggplot plot object

Examples

```
library(ggplot2)
ggplot(iris, aes(Species, Petal.Width, color = Petal.Width)) +
  geom_jitter() +
  theme_wob() +
  scale_rextor()

data <- data.frame(Time = rep(c("Time 1", "Time 2", "Time 3", "Time 4",
                                "Time 5", "Time 6", "Time 7", "Time 8"),
                              each = 2),
                    cont = 1:16,
                    Value = c(8, 5, 10, 3, 8, 5, 10, 3, 8, 5, 10, 3, 8, 5, 10, 3))

ggplot(data, aes(x = Time, y = Value, color = Time, fill = Time)) +
  geom_boxplot(alpha = 0.95) +
  theme_wob() +
  scale_rextor(pal = "neon") +
  scale_rextor(pal = "neon", aesthetic = "fill")

ggplot(data, aes(x = cont, y = Value, color = cont, fill = cont)) +
  geom_jitter(alpha = 0.8, size = 8) +
  theme_minimal() +
  scale_rextor(pal = "wextor") +
  scale_rextor(pal = "wextor", aesthetic = "fill")
```

serious_check

Filter data frame by seriousness check

Description

Filter your data frame by your item for the "seriousness check". If the data comes from 'WEXTOR' with no previous changes to the automatically created variable, you will be able to accept all defaults in this function and simply input the data frame.

Usage

```
serious_check(dataframe, varname = "seriousness", keep = "participate")
```

Arguments

dataframe	Data which contains a variable for the seriousness check.
varname	Variable for the seriousness check. Defaults to "seriousness".
keep	Values from the variable to keep. Defaults to "participate"

Value

Filtered data

Examples

```
data <- read_WEXTOR(path_to_file("BiFiX_data_raw.csv"))
serious_only <- serious_check(data)
```

sess_length_check	<i>Session Length Check</i>
-------------------	-----------------------------

Description

This function takes a dataframe as input as well as the minimum number of webpages in your web experiment or study, that a participant should have seen in order to be considered a complete (or plausible) participation. It flags the most extreme 10% of complete participations in the data as very slow or very fast.

Usage

```
sess_length_check(dataframe, min_pages = 6)
```

Arguments

dataframe	A dataframe containing WEXTOR data (needs to contain variables page_trail for the trail of webpages in the study and session_length for the overall time each participant spent on the study)
min_pages	Numeric. The minimum number of pages a participant must have visited in the study for their participation to be considered plausible. Defaults to 6.

Value

A dataframe with added variables sess_length_clean for only completed session lengths, otherwise NA and check_sess_length for a flag of the most extreme 10% of time needed to finish participation

Examples

```
data <- read_WEXTOR(path_to_file("BiFiX_data_raw.csv"))
new_data <- sess_length_check(data)
```

theme_wob	<i>Theme White on Black</i>
-----------	-----------------------------

Description

This theme option is built to be used with a ggplot2 plot. It offers several options for a dark theme, i.e. dark background with light writing.

Usage

```
theme_wob(max_font_size = 14, contrast = "high")
```

Arguments

max_font_size	Maximum font size, used for the plot title (other font sizes are automatically adjusted accordingly).
contrast	Character to indicate the choice of color palette depending on the desired contrast. Defaults to "high". Should be one of "max", "high", "medium", "low", "pink" or "rex".

Value

Plot theme.

Examples

```
# Load the data
data <- read_WEXTOR(path_to_file("BiFiX_data_raw.csv"))

# Create any plot
library(ggplot2)
ggplot(data, aes(age, color = gender, fill = gender)) +
  geom_density(alpha = .5) +
  theme_wob(contrast = "rex")
```

tidy_WEXTOR_log	<i>Tidy a WEXTOR log into one row per participant</i>
-----------------	---

Description

Takes the tibble returned by `read_WEXTOR_log` and collapses it into 'tidy' format: one row per participant. Every WEXTOR variable submitted anywhere in the study becomes its own column, keyed by a single unique id per row and the resulting data frame contains the commonly relevant paradata variables.

Internally this happens:

1. For each participant, their (usually many) rows are collapsed into one by taking, for every column, the first non-missing value in reverse chronological order by default – i.e. the *last* value submitted/logged for that column. This matters if a participant navigated back and re-submitted a page: the latest value wins, consistent with how `read_WEXTOR_log` itself resolves repeated keys within a single request. Set `duplicates = "first"` to keep the earliest value instead (e.g. the participant's very first timestamp/client_ip). Empty-string answers ("", submitted but left blank) are turned into NA in the final output, same as a variable that was never submitted at all.
2. Participants who share an IP address with at least one other participant are flagged, in a `anomaly_flag` column added at the very end, containing "double IP address" if so, NA otherwise. This is a common sign of one person completing the study more than once from the same computer/network – worth a manual check before treating those rows as independent observations. It only catches an exact IP match; it won't catch someone who switched networks between attempts.

Usage

```
tidy_WEXTOR_log(data, id_col = "id", duplicates = c("last", "first"))
```

Arguments

<code>data</code>	A tibble as returned by <code>read_WEXTOR_log</code> , i.e. containing <code>query</code> , <code>timestamp</code> , <code>client_ip</code> , <code>user_agent</code> and <code>duration_ms</code> columns.
<code>id_col</code>	Name of the participant identifier variable inside query. Defaults to "id", WEXTOR's standard variable name.
<code>duplicates</code>	How to resolve a participant having more than one non-missing value logged for the same column. "last" (default) keeps the most recently submitted/logged value; "first" keeps the earliest.

Value

A tibble with one row per unique participant id (sorted numerically when every id looks numeric, alphabetically otherwise), columns `id`, `timestamp`, `client_ip`, `user_agent` and `duration_ms` (each keeping its original type from data), one column per WEXTOR variable collected anywhere in the study (character; convert types such as SUS/UEQ items to numeric downstream as needed), and a final `anomaly_flag` column as described above.

Examples

```
log <- read_WEXTOR_log(path_to_file("fake_wextor_log.txt"))
data <- tidy_WEXTOR_log(log)

# anything worth a manual look:
dplyr::filter(data, !is.na(anomaly_flag))
```

Index

ask_rex, [2](#)

BiFiX_data_raw, [3](#)

check_ok, [3](#)

fake_wextor_log, [4](#)

getmode, [5](#)

getperc, [6](#)

informed_consent, [6](#)

ip_check, [7](#)

namepref, [8](#)

namepref0, [9](#)

path_to_file, [9](#)

plausicheck, [10](#)

read_WEXTOR, [11](#)

read_WEXTOR(), [11](#)

read_WEXTOR_log, [12](#), [18](#)

removepref, [14](#)

scale_rextor, [14](#)

serious_check, [15](#)

sess_length_check, [16](#)

theme_wob, [17](#)

tibble, [13](#)

tidy_WEXTOR_log, [12](#), [18](#)

tidy_WEXTOR_log(), [13](#)