Package 'corella'

March 24, 2025

Title Prepare, Manipulate and Check Data to Comply with Darwin Core Standard

Version 0.1.4

Description Helps users standardise data to the Darwin Core Standard, a global data standard to store, document, and share biodiversity data like species occurrence records. The package provides tools to manipulate data to conform with, and check validity against, the Darwin Core Standard. Using 'corella' allows users to verify that their data can be used to build 'Darwin Core Archives' using the 'galaxias' package.

Depends R (>= 4.3.0)

Imports cli, dplyr, glue, hms, lubridate, purrr, rlang, sf, stringr, tibble, tidyr, uuid

Suggests gt, knitr, nanoparquet, ozmaps, readr, rmarkdown, testthat (>= 3.0.0), withr

License GPL-3

URL https://corella.ala.org.au

BugReports https://github.com/AtlasOfLivingAustralia/corella/issues

Maintainer Dax Kellie <dax.kellie@csiro.au>

Encoding UTF-8

VignetteBuilder knitr

RoxygenNote 7.3.2

Config/testthat/edition 3

LazyData true

NeedsCompilation no

Author Dax Kellie [aut, cre], Shandiya Balasubramanium [aut], Martin Westgate [aut]

Repository CRAN

Date/Publication 2025-03-24 15:10:06 UTC

Contents

`basisOfRecord_values` *Accepted value functions*

Description

When creating a Darwin Core Archive, several fields have a vocabulary of acceptable values. These functions provide a vector of terms that can be used to fill or validate those fields.

Usage

```
basisOfRecord_values()

countryCode_values()
```

Value

A vector of accepted values for that use case.

See Also

`occurrence_terms()` or `event_terms()` for valid Darwin Core *terms* (i.e. column names).

Examples

```
# See all valid basis of record values
basisOfRecord_values()
```

`check_dataset` *Check a dataset for Darwin Core conformance*

Description

Run a test suite of checks to test whether a `data.frame` or `tibble` conforms to Darwin Core Standard.

While most users will only want to call `suggest_workflow()`, the underlying check functions are exported for detailed work, or for debugging. This function is useful for users experienced with Darwin Core Standard or for final dataset checks.

Usage

```
check_dataset(.df)
```

Arguments

`.df`	A tibble against which checks should be run

Details

`check_dataset()` is modelled after `devtools::test()`. It runs a series of checks, then supplies a summary of passed/failed checks and error messages.

Checks run by `check_dataset()` are the same that would be run automatically by various `set_` functions in a piped workflow. This function allows users with only minor expected updates to check their entire dataset without the need for `set_` functions.

Value

Invisibly returns the input data frame, but primarily called for the side-effect of running check functions on that input.

Examples

```
df <- tibble::tibble(
  scientificName = c("Crinia Signifera", "Crinia Signifera", "Litoria peronii"),
  latitude = c(-35.27, -35.24, -35.83),
  longitude = c(149.33, 149.34, 149.34),
  eventDate = c("2010-10-14", "2010-10-14", "2010-10-14"),
  status = c("present", "present", "present")
  )

# Run a test suite of checks for Darwin Core Standard conformance
```

```
# Checks are only run on columns with names that match Darwin Core terms
df |>
  check_dataset()
```

`composite_id` *Create unique identifier columns*

Description

A unique identifier is a pattern of words, letters and/or numbers that is unique to a single record within a dataset. Unique identifiers are useful because they identify individual observations, and make it possible to change, amend or delete observations over time. They also prevent accidental deletion when when more than one record contains the same information(and would otherwise be considered a duplicate).

The identifier functions in corella make it easier to generate columns with unique identifiers in a dataset. These functions can be used within `set_events()`, `set_occurrences()`, or (equivalently) `dplyr::mutate()`.

Usage

```
composite_id(..., sep = "-")

sequential_id(width)

random_id()
```

Arguments

`...`	Zero or more variable names from the tibble being mutated (unquoted), and/or zero or more `_id` functions, separated by commas.
`sep`	Character used to separate field values. Defaults to `"-"`
`width`	(Integer) how many characters should the resulting string be? Defaults to one plus the order of magnitude of the largest number.

Details

Generally speaking, it is better to use existing information from a dataset to generate identifiers. For this reason we recommend using `composite_id()` to aggregate existing fields, if no such composite is already present within the dataset. Composite IDs are more meaningful and stable; they are easier to check and harder to overwrite.

It is possible to call `sequential_id()` or `random_id()` within `composite_id()` to combine existing and new columns.

Value

An amended `tibble` containing a column with identifiers in the requested format.

Examples

```
df <- tibble::tibble(
  eventDate = paste0(rep(c(2020:2024), 3), "-01-01"),
  basisOfRecord = "humanObservation",
  site = rep(c("A01", "A02", "A03"), each = 5)
  )

# Add composite ID using a random ID, site name and eventDate
df |>
  set_occurrences(
    occurrenceID = composite_id(random_id(),
                                site,
                                eventDate)
    )

# Add composite ID using a sequential number, site name and eventDate
df |>
  set_occurrences(
    occurrenceID = composite_id(sequential_id(),
                                site,
                                eventDate)
    )
```

`country_codes` *Dataset of supported Country Codes*

Description

A `tibble` of ISO 3166-1 alpha-2 codes for countries, which are the accepted standard for supplying `countryCode` in Darwin Core Standard.

Usage

```
country_codes
```

Format

A `tibble` containing valid country codes (249 rows x 3 columns). Column descriptions are as follows:

name ISO 3166-1 alpha-2 code, pointing to its ISO 3166-2 article.

code English short name officially used by the ISO 3166 Maintenance Agency (ISO 3166/MA).

year Year when alpha-2 code was first officially assigned.

Source

Wikipedia.

See Also

`set_locality()` for assigning `countryCode` within a tibble; `countryCode_values()` to return valid codes as a vector.

`darwin_core_terms` *Dataset of supported Darwin Core terms*

Description

The Darwin Core Standard is maintained by Biodiversity Information Standards, previously known as the Taxonomic Databases Working Group and known by the acronym 'TDWG'. This `tibble` is the full list of supported terms, current as of 2024-12-10.

Users can use `occurrence_terms()` and `event_terms()` as convenience functions to access these terms.

Usage

```
darwin_core_terms
```

Format

A `tibble` containing valid Darwin Core Standard terms (206 rows x 6 columns). Column descriptions are as follows:

class TDWG group that a term belongs to.

term Column header names that can be used in Darwin Core

url Stable url to information describing the term.

definition Human-readable definition of the term.

comments Further information from TDWG.

examples Examples of how the field should be populated.

set_functions Function in corella that supports Darwin Core term.

Source

Slightly modified version of a table supplied by TDWG.

See Also

`occurrence_terms()` and `event_terms()` to get terms for use in `dplyr::select()`

occurrence_terms *Select support functions*

Description

When creating a Darwin Core archive, it is often useful to select only those fields that conform to the standard. These functions provide a vector of terms that can be used in combination with `dplyr::select()` and `dplyr::any_of()` to quickly select Darwin Core terms for the relevant data type (events, occurrences, media).

Usage

```
occurrence_terms()

event_terms()
```

Value

A vector of accepted (but not mandatory) values for that use case.

See Also

`basisOfRecord_values()` or `countryCode_values()` for valid entries *within* a field.

Examples

```
# Return a vector of accepted terms in an Occurrence-based dataset
occurrence_terms() |> head(10L) # first 10 terms

# Use this vector to filter a data frame
df <- tibble::tibble(
  name = c("Crinia Signifera", "Crinia Signifera", "Litoria peronii"),
  latitude = c(-35.27, -35.24, -35.83),
  longitude = c(149.33, 149.34, 149.34),
  eventDate = c("2010-10-14", "2010-10-14", "2010-10-14"),
  measurement1 = c(24.3, 24.9, 20.1), # example measurement column
  measurement2 = c(0.92, 1.03, 1.09)  # example measurement column
  )

df |>
  dplyr::select(any_of(occurrence_terms()))
```

set_abundance *Set, create or modify columns with abundance information*

Description

In some field methods, it is common to observe more than one individual per observation; to observe abundance using non-integer measures such as mass or area; or to seek individuals but not find them (abundance of zero). As these approaches use different Darwin Core terms, this function assists in specifying abundances to a `tibble` using Darwin Core Standard.

In practice this is no different from using `mutate()`, but gives some informative errors, and serves as a useful lookup for how columns with abundance information are represented in the Darwin Core Standard.

Usage

```
set_abundance(
  .df,
  individualCount = NULL,
  organismQuantity = NULL,
  organismQuantityType = NULL,
  .keep = "unused"
)
```

Arguments

`.df`	A `data.frame` or `tibble` that the column should be appended to.
`individualCount`	The number of individuals present
`organismQuantity`	A number or enumeration value for the quantity of organisms. Used together with `organismQuantityType` to provide context.
`organismQuantityType`	The type of quantification system used for `organismQuantity`.
`.keep`	Control which columns from .data are retained in the output. Note that unlike `dplyr::mutate()`, which defaults to `"all"` this defaults to `"unused"`; i.e. only keeps Darwin Core columns, and not those columns used to generate them.

Details

Examples of `organismQuantity` & `organismQuantityType` values:

- 27 (`organismQuantity`) individuals (`organismQuantityType`)
- 12.5 (`organismQuantity`) % biomass (`organismQuantityType`)
- r (`organismQuantity`) Braun-Blanquet Scale (`organismQuantityType`)
- many (`organismQuantity`) individuals (`organismQuantityType`)

Value

A `tibble` with the requested fields added/reformatted.

Examples

```
df <- tibble::tibble(
  scientificName = c("Cacatua (Licmetis) tenuirostris",
                     "Cacatua (Licmetis) tenuirostris",
                     "Cacatua (Licmetis) tenuirostris"),
  n_obs = c(1, 3, 4)
  )

df |>
  set_abundance(individualCount = n_obs)
```

`set_collection` *Set, create or modify columns with museum- or collection-specific information*

Description

Format fields that specify the collection or catalog number of a specimen or occurrence record to a `tibble` using Darwin Core Standard.

In practice this is no different from using `mutate()`, but gives some informative errors, and serves as a useful lookup for fields in the Darwin Core Standard.

Usage

```
set_collection(
  .df,
  datasetID = NULL,
  datasetName = NULL,
  catalogNumber = NULL,
  .keep = "unused"
)
```

Arguments

`.df`	A `data.frame` or `tibble` that the column should be appended to.
`datasetID`	An identifier for the set of data. May be a global unique identifier or an identifier specific to a collection or institution.
`datasetName`	The name identifying the data set from which the record was derived.
`catalogNumber`	A unique identifier for the record within the data set or collection.
`.keep`	Control which columns from .data are retained in the output. Note that unlike `dplyr::mutate()`, which defaults to `"all"` this defaults to `"unused"`; i.e. only keeps Darwin Core columns, and not those columns used to generate them.

Details

Examples of `datasetID` values:

- `b15d4952-7d20-46f1-8a3e-556a512b04c5`

Examples of `datasetName` values:

- `Grinnell Resurvey Mammals`
- `Lacey Ctenomys Recaptures`

Examples of `catalogNumber` values:

- `145732`
- `145732a`
- `2008.1334`
- `R-4313`

Value

A `tibble` with the requested fields added/reformatted.

Examples

```
df <- tibble::tibble(
  name = c("Crinia Signifera", "Crinia Signifera", "Litoria peronii"),
  eventDate = c("2010-10-14", "2010-10-14", "2010-10-14"),
  catalog_num = c("16789a", "16789c", "08742f"),
  dataset = c("Frog search", "Frog search", "Frog search")
  )

# Reformat columns to Darwin Core terms
df |>
  set_collection(
    catalogNumber = catalog_num,
    datasetName = dataset
    )
```

---

set_coordinates *Set, create or modify columns with spatial information*

Description

This function helps format standard location fields like latitude and longitude point coordinates to a `tibble` using Darwin Core Standard.

Usage

```
set_coordinates(
  .df,
  decimalLatitude = NULL,
  decimalLongitude = NULL,
  geodeticDatum = NULL,
  coordinateUncertaintyInMeters = NULL,
  coordinatePrecision = NULL,
  .keep = "unused"
)
```

Arguments

`.df`	A `data.frame` or `tibble` that the column should be appended to.
`decimalLatitude`	The latitude in decimal degrees.
`decimalLongitude`	The longitude in decimal degrees.
`geodeticDatum`	The datum or spatial reference system that coordinates are recorded against (usually "WGS84" or "EPSG:4326"). This is often known as the Coordinate Reference System (CRS). If your coordinates are from a GPS system, your data are already using WGS84.
`coordinateUncertaintyInMeters`	(numeric) Radius of the smallest circle that contains the whole location, given any possible measurement error. `coordinateUncertaintyInMeters` will typically be around `30` (metres) if recorded with a GPS after 2000, or `100` before that year.
`coordinatePrecision`	(numeric) The precision that `decimalLatitude` and `decimalLongitude` are supplied to. `coordinatePrecision` should be no less than 0.00001 if data were collected using GPS.
`.keep`	Control which columns from `.df` are retained in the output. Note that unlike `dplyr::mutate()`, which defaults to `"all"` this defaults to `"unused"`; i.e. only keeps Darwin Core columns, and not those columns used to generate them.

Details

In practice this is no different from using `mutate()`, but gives some informative errors, and serves as a useful lookup for how spatial columns are represented in the Darwin Core Standard.

Example values are:

- `geodeticDatum` should be a valid EPSG code

Value

A `tibble` with the requested columns added/reformatted.

See Also

set_locality() for provided text-based spatial information.

Examples

```
df <- tibble::tibble(
  scientificName = c("Crinia Signifera", "Crinia Signifera", "Litoria peronii"),
  latitude = c(-35.27, -35.24, -35.83),
  longitude = c(149.33, 149.34, 149.34),
  eventDate = c("2010-10-14", "2010-10-14", "2010-10-14")
  )

# Reformat columns to Darwin Core Standard terms
df |>
  set_coordinates(
    decimalLongitude = longitude,
    decimalLatitude = latitude
    )
```

---

set_coordinates_sf *Set, create or modify columns with* sf *spatial information*

Description

This function helps format standard location fields like longitude and latitude point coordinates to a tibble using Darwin Core Standard.

It differs from set_coordinates() by accepting sf geometry columns of class POINTas coordinates (rather than numeric lat/lon coordinates). The advantage of using an sf geometry is that the Coordinate Reference System (CRS) is automatically formatted into the required geodeticDatum column.

Usage

```
set_coordinates_sf(.df, geometry = NULL, .keep = "unused")
```

Arguments

.df	A data.frame or tibble that the column should be appended to.
geometry	The latitude/longitude coordinates as sf POINT class
.keep	Control which columns from .data are retained in the output. Note that unlike dplyr::mutate(), which defaults to "all" this defaults to "unused"; i.e. only keeps Darwin Core columns, and not those columns used to generate them.

Value

A tibble with the requested columns added/reformatted.

See Also

`set_coordinates()` for providing numeric coordinates, `set_locality()` for providing text-based spatial information.

Examples

```
df <- tibble::tibble(
  scientificName = c("Crinia Signifera", "Crinia Signifera", "Litoria peronii"),
  latitude = c(-35.27, -35.24, -35.83),
  longitude = c(149.33, 149.34, 149.34),
  eventDate = c("2010-10-14", "2010-10-14", "2010-10-14")
  ) |>
  sf::st_as_sf(coords = c("longitude", "latitude")) |>
  sf::st_set_crs(4326)

# Reformat columns to Darwin Core Standard terms.
# Coordinates and CRS are automatically detected and reformatted.
df |>
  set_coordinates_sf()
```

---

set_datetime *Set, create or modify columns with date and time information*

Description

This function helps format standard date/time columns in a `tibble` using Darwin Core Standard. Users should make use of the lubridate package to format their dates so corella can read them correctly.

In practice this is no different from using `mutate()`, but gives some informative errors, and serves as a useful lookup for how spatial fields are represented in the Darwin Core Standard.

Usage

```
set_datetime(
  .df,
  eventDate = NULL,
  year = NULL,
  month = NULL,
  day = NULL,
  eventTime = NULL,
  .keep = "unused",
  .messages = TRUE
)
```

Arguments

.df	A data.frame or tibble that the column should be appended to.
eventDate	The date or date + time that the observation/event occurred.
year	The year of the observation/event.
month	The month of the observation/event.
day	The day of the observation/event.
eventTime	The time of the event. Use this term for Event data. Date + time information for observations is accepted in eventDate.
.keep	Control which columns from .data are retained in the output. Note that unlike dplyr::mutate(), which defaults to "all" this defaults to "unused"; i.e. only keeps Darwin Core fields, and not those fields used to generate them.
.messages	(logical) Should informative messages be shown? Defaults to TRUE.

Details

Example values are:

- eventDate should be class Date or POSITct. We suggest using the lubridate package to define define your date format using functions like ymd(), mdy, dmy(), or if including date + time, ymd_hms(), ymd_hm(), or ymd_h().

Value

A tibble with the requested columns added/reformatted.

Examples

```
df <- tibble::tibble(
  name = c("Crinia Signifera", "Crinia Signifera", "Litoria peronii"),
  latitude = c(-35.27, -35.24, -35.83),
  longitude = c(149.33, 149.34, 149.34),
  date = c("2010-10-14", "2010-10-14", "2010-10-14"),
  time = c("10:08:12", "13:01:45", "14:02:33")
)

# Use the lubridate package to format date + time information
# eventDate accepts date + time
df |>
  set_datetime(
    eventDate = lubridate::ymd_hms(paste(date, time))
  )
```

`set_events` *Set, create or modify columns with Event information*

Description

Identify or format columns that contain information about an Event. An "Event" in Darwin Core Standard refers to an action that occurs at a place and time. Examples include:

- A specimen collecting event
- A survey or sampling event
- A camera trap image capture
- A marine trawl
- A camera trap deployment event
- A camera trap burst image event (with many images for one observation)

In practice this function is used no differently from `mutate()`, but gives users some informative errors, and serves as a useful lookup for fields in the Darwin Core Standard.

Usage

```
set_events(
  .df,
  eventID = NULL,
  eventType = NULL,
  parentEventID = NULL,
  .keep = "unused",
  .keep_composite = "all"
)
```

Arguments

`.df`	A `data.frame` or `tibble` that the column should be appended to.
`eventID`	A unique identifier for an individual Event.
`eventType`	The type of Event
`parentEventID`	The parent event under which one or more Events sit within.
`.keep`	Control which columns from `.df` are retained in the output. Note that unlike `dplyr::mutate()`, which defaults to `"all"` this defaults to `"unused"`; i.e. only keeps Darwin Core columns, and not those columns used to generate them.
`.keep_composite`	Control which columns from `.df` are kept when `composite_id()` is used to assign values to `eventID`, defaulting to `"all"`. This has a different default from `.keep` because composite identifiers often contain information that is valuable in other contexts, meaning that deleting these columns by default is typically unwise.

Details

Each Event requires a unique `eventID` and `eventType` (because there can be several types of Events in a single dataset), along with a `parentEventID` which specifies the level under which the current Event sits (e.g., An individual location's survey event ID, which is one of several survey locations on a specific day's set of surveys ie the parentEvent).

Examples of `eventID` values:

- `INBO:VIS:Ev:00009375`

Examples of `eventType` values:

- `Sample`
- `Observation`
- `Survey`
- `Site Visit`
- `Deployment` See more examples on dwc.tdwg.org

Examples of `parentEventID` `A1` (To identify the parent event in nested samples, each with their own `eventID` - `A1_1`, `A1_2`)

Value

A `tibble` with the requested fields added/reformatted.

Examples

```
# example Event dataframe
df <- tibble::tibble(
  site_code = c("AMA100", "AMA100", "AMH100"),
  scientificName = c("Crinia signifera", "Crinia signifera", "Crinia signifera"),
  latitude = c(-35.275, -35.274, -35.101),
  longitue = c(149.001, 149.004, 149.274),
  )

# Add event information
df |>
  set_events(
    eventID = composite_id(sequential_id(),
                           site_code,
                           year),
    eventType = "Survey"
    )
```

`set_individual_traits` *Set, create or modify columns with information of individual organisms*

Description

Format fields that contain measurements or attributes of individual organisms to a `tibble` using Darwin Core Standard. Fields include those that specify sex, life stage or condition. Individuals can be identified by an `individualID` if data contains resampling.

In practice this is no different from using `mutate()`, but gives some informative errors, and serves as a useful lookup for fields in the Darwin Core Standard.

Usage

```
set_individual_traits(
  .df,
  individualID = NULL,
  lifeStage = NULL,
  sex = NULL,
  vitality = NULL,
  reproductiveCondition = NULL,
  .keep = "unused"
)
```

Arguments

`.df`	A `data.frame` or `tibble` that the column should be appended to.
`individualID`	An identifier for an individual or named group of individual organisms represented in the Occurrence. Meant to accommodate resampling of the same individual or group for monitoring purposes. May be a global unique identifier or an identifier specific to a data set.
`lifeStage`	The age class or life stage of an organism at the time of occurrence.
`sex`	The sex of the biological individual.
`vitality`	An indication of whether an organism was alive or dead at the time of collection or observation.
`reproductiveCondition`	The reproductive condition of the biological individual.
`.keep`	Control which columns from .data are retained in the output. Note that unlike `dplyr::mutate()`, which defaults to `"all"` this defaults to `"unused"`; i.e. only keeps Darwin Core columns, and not those columns used to generate them.

Details

Examples of `lifeStage` values:

- `zygote`

- larva
- adult
- seedling
- flowering

Examples of vitality values:

- alive
- dead
- uncertain

Examples of reproductiveCondition values:

- non-reproductive
- pregnant
- in bloom
- fruit bearing

Value

A tibble with the requested fields added/reformatted.

See Also

set_scientific_name() for adding scientificName and authorship information.

Examples

```
df <- tibble::tibble(
  name = c("Crinia Signifera", "Crinia Signifera", "Litoria peronii"),
  latitude = c(-35.27, -35.24, -35.83),
  longitude = c(149.33, 149.34, 149.34),
  eventDate = c("2010-10-14", "2010-10-14", "2010-10-14"),
  id = c(4421, 4422, 3311),
  life_stage = c("juvenile", "adult", "adult")
  )

# Reformat columns to Darwin Core Standard
df |>
  set_individual_traits(
    individualID = id,
    lifeStage = life_stage
    )
```

`set_license` *Set, create or modify columns with license and rights information*

Description

Format fields that contain information on permissions for use, sharing or access to a record to a `tibble` using Darwin Core Standard.

In practice this function is no different from using `mutate()`, but gives some informative errors, and serves as a useful lookup for fields in the Darwin Core Standard.

Usage

```
set_license(
  .df,
  license = NULL,
  rightsHolder = NULL,
  accessRights = NULL,
  .keep = "unused"
)
```

Arguments

`.df`	A `data.frame` or `tibble` that the column should be appended to.
`license`	A legal document giving official permission to do something with the resource. Must be provided as a url to a valid license.
`rightsHolder`	Person or organisation owning or managing rights to resource.
`accessRights`	Access or restrictions based on privacy or security.
`.keep`	Control which columns from .data are retained in the output. Note that unlike `dplyr::mutate()`, which defaults to `"all"` this defaults to `"unused"`; i.e. only keeps Darwin Core columns, and not those columns used to generate them.

Details

Examples of `license` values:

- `http://creativecommons.org/publicdomain/zero/1.0/legalcode`
- `http://creativecommons.org/licenses/by/4.0/legalcode`
- `CC0`
- `CC-BY-NC 4.0 (Int)`

Examples of `rightsHolder` values:

- `The Regents of the University of California`

Examples of `accessRights` values:

- `not-for-profit use only` (string example)
- `https://www.fieldmuseum.org/field-museum-natural-history-conditions-and-suggested-norms-use-col` (URI example)

Value

A `tibble` with the requested fields added/reformatted.

See Also

`set_observer()` for adding observer information.

Examples

```
df <- tibble::tibble(
  name = c("Crinia Signifera", "Crinia Signifera", "Litoria peronii"),
  latitude = c(-35.27, -35.24, -35.83),
  longitude = c(149.33, 149.34, 149.34),
  eventDate = c("2010-10-14", "2010-10-14", "2010-10-14"),
  attributed_license = c("CC-BY-NC 4.0 (Int)", "CC-BY-NC 4.0 (Int)", "CC-BY-NC 4.0 (Int)")
  )

# Reformat columns to Darwin Core Standard
df |>
  set_license(
    license = attributed_license
    )
```

---

set_locality *Set, create or modify columns with locality information*

Description

Locality information refers to a description of a place, rather than a spatial coordinate. This function helps to format columns with locality information to a `tibble` using Darwin Core Standard.

In practice this is used no differently from `mutate()`, but gives some informative errors, and serves as a useful lookup for fields in the Darwin Core Standard.

Usage

```
set_locality(
  .df,
  continent = NULL,
  country = NULL,
  countryCode = NULL,
  stateProvince = NULL,
  locality = NULL,
  .keep = "unused"
)
```

Arguments

.df	A data.frame or tibble that the column should be appended to.
continent	(string) Valid continent. See details.
country	Valid country name. See country_codes.
countryCode	Valid country code. See country_codes.
stateProvince	A sub-national region.
locality	A specific description of a location or place.
.keep	Control which columns from .data are retained in the output. Note that unlike dplyr::mutate(), which defaults to "all" this defaults to "unused"; i.e. only keeps Darwin Core columns, and not those columns used to generate them.

Details

Values of continent should be one of "Africa", "Antarctica", "Asia", "Europe", "North America", "Oceania" or "South America".

countryCode should be supplied according to the ISO 3166-1 ALPHA-2 standard, as per TDWG advice. Examples of countryCode:

- AUS
- NZ
- BRA

Examples of locality:

- Bariloche, 25 km NNE via Ruta Nacional 40 (=Ruta 237)
- Queets Rainforest, Olympic National Park

Value

A tibble with the requested columns added/reformatted.

See Also

set_coordinates() for numeric spatial data.

Examples

```
df <- tibble::tibble(
  scientificName = c("Crinia Signifera", "Crinia Signifera", "Litoria peronii"),
  latitude = c(-35.27, -35.24, -35.83),
  longitude = c(149.33, 149.34, 149.34),
  eventDate = c("2010-10-14", "2010-10-14", "2010-10-14"),
  countryCode = c("AU", "AU", "AU"),
  state = c("New South Wales", "New South Wales", "New South Wales"),
  locality = c("Melville Caves", "Melville Caves", "Bryans Swamp about 3km away")
)

# Reformat columns to Darwin Core Standard terms
```

```
df |>
  set_locality(
    countryCode = countryCode,
    stateProvince = state,
    locality = locality
  )

# Columns with valid Darwin Core terms as names are automatically detected
# and checked. This will do the same as above.
df |>
  set_locality(
    stateProvince = state
  )
```

`set_measurements` *Convert columns with measurement data for an individual or event to Darwin Core standard*

Description

[Experimental] This function is a work in progress, and should be used with caution.

In raw collected data, many types of information can be captured in one column. For example, the column name `LMA_g.m2` contains the measured trait (Leaf Mass per Area, LMA) and the unit of measurement (grams per meter squared, g/m2), and recorded in that column are the values themselves. In Darwin Core, these different types of information must be separated into multiple columns so that they can be ingested correctly and aggregated with sources of data accurately.

This function converts information preserved in a single measurement column into multiple columns (`measurementID`, `measurementUnit`, and `measurementType`) as per Darwin Core standard.

Usage

```
set_measurements(.df, cols = NULL, unit = NULL, type = NULL, .keep = "unused")
```

Arguments

`.df`	a `data.frame` or `tibble` that the column should be appended to.
`cols`	vector of column names to be included as 'measurements'. Unquoted.
`unit`	vector of strings giving units for each variable
`type`	vector of strings giving a description for each variable
`.keep`	Control which columns from .data are retained in the output. Note that unlike `dplyr::mutate()`, which defaults to `"all"` this defaults to `"unused"`; i.e. only keeps Darwin Core fields, and not those fields used to generate them.

Details

Columns are nested in a single column `measurementOrFact` that contains Darwin Core Standard measurement fields. By nesting three measurement columns within the `measurementOrFact` column, nested measurement columns can be converted to long format (one row per measurement, per occurrence) while the original data frame remains organised by one row per occurrence. Data can be unnested into long format using `tidyr::unnest()`.

Value

A tibble with the requested fields added.

Examples

```
library(tidyr)

# Example data of plant species observations and measurements
df <- tibble::tibble(
  Site = c("Adelaide River", "Adelaide River", "AgnesBanks"),
  Species = c("Corymbia latifolia", "Banksia aemula", "Acacia aneura"),
  Latitude = c(-13.04, -13.04, -33.60),
  Longitude = c(131.07, 131.07, 150.72),
  LMA_g.m2 = c(NA, 180.07, 159.01),
  LeafN_area_g.m2 = c(1.100, 0.913, 2.960)
)

# Reformat columns to Darwin Core Standard
# Measurement columns are reformatted and nested in column `measurementOrFact`
df_dwc <- df |>
  set_measurements(
    cols = c(LMA_g.m2,
             LeafN_area_g.m2),
    unit = c("g/m2",
             "g/m2"),
    type = c("leaf mass per area",
             "leaf nitrogen per area")
  )

df_dwc

# Unnest to view full long format data frame
df_dwc |>
  tidyr::unnest(measurementOrFact)
```

`set_observer` *Set, create or modify columns with information of who made an observation*

Description

Format fields that contain information about who made a specific observation of an organism to a `tibble` using Darwin Core Standard.

In practice this is no different from using `mutate()`, but gives some informative errors, and serves as a useful lookup for fields in the Darwin Core Standard.

Usage

```
set_observer(.df, recordedBy = NULL, recordedByID = NULL, .keep = "unused")
```

Arguments

`.df`	A `data.frame` or `tibble` that the column should be appended to.
`recordedBy`	Names of people, groups, or organizations responsible for recording the original occurrence. The primary collector or observer should be listed first.
`recordedByID`	The globally unique identifier for the person, people, groups, or organizations responsible for recording the original occurrence.
`.keep`	Control which columns from .data are retained in the output. Note that unlike `dplyr::mutate()`, which defaults to `"all"` this defaults to `"unused"`; i.e. only keeps Darwin Core columns, and not those columns used to generate them.

Details

Examples of `recordedBy` values:

- `José E. Crespo`

Examples of `recordedByID` values:

- `c("https://orcid.org/0000-0002-1825-0097", "https://orcid.org/0000-0002-1825-0098")`

Value

A `tibble` with the requested fields added/reformatted.

Examples

```
df <- tibble::tibble(
  name = c("Crinia Signifera", "Crinia Signifera", "Litoria peronii"),
  latitude = c(-35.27, -35.24, -35.83),
  longitude = c(149.33, 149.34, 149.34),
  eventDate = c("2010-10-14", "2010-10-14", "2010-10-14"),
  observer = c("David Attenborough", "David Attenborough", "David Attenborough")
  )

# Reformat columns to Darwin Core terms
df |>
  set_observer(
    recordedBy = observer
    )
```

`set_occurrences` *Set, create or modify columns with occurrence-specific information*

Description

Format fields uniquely identify each occurrence record and specify the type of record. `occurrenceID` and `basisOfRecord` are necessary fields of information for occurrence records, and should be appended to a data set to conform to Darwin Core Standard prior to submission.

In practice this is no different from using `mutate()`, but gives some informative errors, and serves as a useful lookup for fields in the Darwin Core Standard.

Usage

```
set_occurrences(
  .df,
  occurrenceID = NULL,
  basisOfRecord = NULL,
  occurrenceStatus = NULL,
  .keep = "unused",
  .keep_composite = "all",
  .messages = TRUE
)
```

Arguments

`.df`	a `data.frame` or `tibble` that the column should be appended to.
`occurrenceID`	A character string. Every occurrence should have an `occurrenceID` entry. Ideally IDs should be persistent to avoid being lost in future updates. They should also be unique, both within the dataset, and (ideally) across all other datasets.
`basisOfRecord`	Record type. Only accepts `camelCase`, for consistency with field names. Accepted `basisOfRecord` values are one of: • `"humanObservation"`, `"machineObservation"`, `"livingSpecimen"`, `"preservedSpecimen"`, `"fossilSpecimen"`, `"materialCitation"`
`occurrenceStatus`	Either `"present"` or `"absent"`.
`.keep`	Control which columns from `.df` are retained in the output. Note that unlike `dplyr::mutate()`, which defaults to `"all"` this defaults to `"unused"`; i.e. only keeps Darwin Core columns, and not those columns used to generate them.
`.keep_composite`	Control which columns from `.df` are kept when `composite_id()` is used to assign values to `occurrenceID`, defaulting to `"all"`. This has a different default from `.keep` because composite identifiers often contain information that is valuable in other contexts, meaning that deleting these columns by default is typically unwise.
`.messages`	Logical: Should progress message be shown? Defaults to `TRUE`.

Details

Examples of occurrenceID values:

- 000866d2-c177-4648-a200-ead4007051b9
- http://arctos.database.museum/guid/MSB:Mamm:233627

Accepted basisOfRecord values are one of:

- "humanObservation", "machineObservation", "livingSpecimen", "preservedSpecimen", "fossilSpecimen", "materialCitation"

Value

A tibble with the requested columns added/reformatted.

See Also

basisOfRecord_values() for accepted values for the basisOfRecord field'; random_id(), composite_id() or sequential_id() for formatting ID columns; set_abundance() for occurrence-level counts.

Examples

```
df <- tibble::tibble(
  scientificName = c("Crinia Signifera", "Crinia Signifera", "Litoria peronii"),
  latitude = c(-35.27, -35.24, -35.83),
  longitude = c(149.33, 149.34, 149.34),
  eventDate = c("2010-10-14", "2010-10-14", "2010-10-14")
  )

# Add occurrence information
df |>
  set_occurrences(
    occurrenceID = composite_id(random_id(), eventDate), # add composite ID
    basisOfRecord = "humanObservation"
    )
```

---

set_scientific_name *Set, create or modify columns with scientific name & authorship information*

Description

Format the field scientificName, the lowest identified taxonomic name of an occurrence, along with the rank and authorship of the provided name to a tibble using Darwin Core Standard.

Usage

```
set_scientific_name(
  .df,
  scientificName = NULL,
  scientificNameAuthorship = NULL,
  taxonRank = NULL,
  .keep = "unused"
)
```

Arguments

.df	A data.frame or tibble that the column should be appended to.
scientificName	The full scientific name in the lower level taxonomic rank that can be determined.
scientificNameAuthorship	The authorship information for scientificName.
taxonRank	The taxonomic rank of scientificName.
.keep	Control which columns from .data are retained in the output. Note that unlike dplyr::mutate(), which defaults to "all" this defaults to "unused"; i.e. only keeps Darwin Core columns, and not those columns used to generate them.

Details

In practice this function is used no differently from mutate(), but gives users some informative errors, and serves as a useful lookup for accepted column names in the Darwin Core Standard.

Examples of scientificName values (we specify the rank in parentheses, but users should not include this information):

- Coleoptera (order)
- Vespertilionidae (family)
- Manis (genus)
- Ctenomys sociabilis (genus + specificEpithet)
- Ambystoma tigrinum diaboli (genus + specificEpithet + infraspecificEpithet)

Examples of scientificNameAuthorship:

- (Györfi, 1952)
- R. A. Graham
- (Martinovský) Tzvelev

Examples of taxonRank:

- order
- genus
- subspecies
- infraspecies

Value

A tibble with the requested columns added/reformatted.

See Also

`set_taxonomy()` for taxonomic name information.

Examples

```
df <- tibble::tibble(
  name = c("Crinia Signifera", "Crinia Signifera", "Litoria peronii"),
  latitude = c(-35.27, -35.24, -35.83),
  longitude = c(149.33, 149.34, 149.34),
  eventDate = c("2010-10-14", "2010-10-14", "2010-10-14")
  )

# Reformat columns to Darwin Core Standard terms
df |>
  set_scientific_name(
    scientificName = name
    )
```

---

set_taxonomy *Set, create or modify columns with taxonomic information*

Description

Format fields that contain taxonomic name information from kingdom to species, as well as the common/vernacular name, to a `tibble` using Darwin Core Standard.

In practice this is no different from using `mutate()`, but gives some informative errors, and serves as a useful lookup for accepted column names in the Darwin Core Standard.

Usage

```
set_taxonomy(
  .df,
  kingdom = NULL,
  phylum = NULL,
  class = NULL,
  order = NULL,
  family = NULL,
  genus = NULL,
  specificEpithet = NULL,
  vernacularName = NULL,
  .keep = "unused"
)
```

Arguments

.df	A data.frame or tibble that the column should be appended to.
kingdom	The kingdom name of identified taxon.
phylum	The phylum name of identified taxon.
class	The class name of identified taxon.
order	The order name of identified taxon.
family	The family name of identified taxon.
genus	The genus name of the identified taxon.
specificEpithet	The name of the first species or species epithet of the scientificName. See documentation
vernacularName	The common or vernacular name of the identified taxon.
.keep	Control which columns from .data are retained in the output. Note that unlike dplyr::mutate(), which defaults to "all" this defaults to "unused"; i.e. only keeps Darwin Core columns, and not those columns used to generate them.

Details

Examples of specificEphithet:

- If scientificName is Abies concolor, the specificEpithet is concolor.
- If scientificName is Semisulcospira gottschei, the specificEpithet is gottschei.

Value

A tibble with the requested columns added/reformatted.

See Also

set_scientific_name() for adding scientificName and authorship information.

Examples

```
df <- tibble::tibble(
  scientificName = c("Crinia Signifera", "Crinia Signifera", "Litoria peronii"),
  fam = c("Myobatrachidae", "Myobatrachidae", "Hylidae"),
  ord = c("Anura", "Anura", "Anura"),
  latitude = c(-35.27, -35.24, -35.83),
  longitude = c(149.33, 149.34, 149.34),
  eventDate = c("2010-10-14", "2010-10-14", "2010-10-14")
  )

# Reformat columns to Darwin Core terms
df |>
  set_scientific_name(
    scientificName = scientificName
    ) |>
```

```
  set_taxonomy(
    family = fam,
    order = ord
    )
```

`suggest_workflow` *Suggest a workflow to make data comply with Darwin Core Standard*

Description

Checks whether a `data.frame` or `tibble` conforms to Darwin Core Standard and suggests how to standardise a data frame that is not standardised to minimum Darwin Core requirements. This is intended as users' go-to function for figuring out how to get started standardising their data.

Output provides a summary to users about which column names match valid Darwin Core terms, the minimum required column names/terms (and which ones are missing), and a suggested workflow to add any missing terms.

Usage

```
suggest_workflow(.df)
```

Arguments

`.df`	A `data.frame`/`tibble` against which checks should be run

Value

Invisibly returns the input `data.frame`/`tibble`, but primarily called for the side-effect of running check functions on that input.

Examples

```
df <- tibble::tibble(
  scientificName = c("Callocephalon fimbriatum", "Eolophus roseicapilla"),
  latitude = c(-35.310, "-35.273"), # deliberate error for demonstration purposes
  longitude = c(149.125, 149.133),
  eventDate = c("14-01-2023", "15-01-2023"),
  status = c("present", "present")
)

# Summarise whether your data conforms to Darwin Core Standard.
# See a suggested workflow to amend or add missing information.
df |>
  suggest_workflow()
```

Index