Package: VasGBIF (via r-universe)

October 22, 2026

Title Fast and Easy Compilation of Vascular Plants Occurrence Records from 'GBIF'

Version 3.7.1

Date 2026-10-11

Type Package

Maintainer Yuxuan Wang <wyx1743@gmail.com>

URL https://github.com/wyx619/VasGBIF (devel), https://wyx619.github.io/VasGBIF/ (documentation)

BugReports https://github.com/wyx619/VasGBIF/issues

Description 'GBIF' <https://www.gbif.org/> hosts over 500 million occurrence records of tracheophyte (vascular plants). However, dealing with them typically involves lots of workflows and time, posing significant challenges for researchers. To address these limitations, we present 'VasGBIF', a fast and easy 'R' package to parse and merge enormous tracheophyte records.

License GPL (>= 3.0)

Encoding UTF-8

Roxygen list(markdown = TRUE)

LazyData true

LazyDataCompression xz

Depends R (>= 4.1),data.table,bit64

Imports dplyr,stringi,terra (>= 1.8-10),CoordinateCleaner,TNRS,geohashTools,mapview,foreach,doParallel

Suggests knitr, leaflet, rmarkdown, rnaturalearth, sf, rgbif, letsR, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

Config/Needs/website pkgdown, knitr, rmarkdown, rgbif, letsR, sf, rnaturalearth, leaflet

Config/pak/sysreqs libabsl-dev cmake libfontconfig1-dev libfreetype6-dev libfribidi-dev libgdal-dev gdal-bin libgeos-dev make libharfbuzz-dev libicu-dev libpng-dev libuv1-dev libxml2-dev libssl-dev libproj-dev libsqlite3-dev libudunits2-dev zlib1g-dev

Repository https://wyx619.r-universe.dev

Date/Publication 2026-10-11 04:36:24 UTC

RemoteUrl https://github.com/wyx619/vasgbif

RemoteRef HEAD

RemoteSha ef82ba6edbb4befb3369b04004139c46195800f8

Contents

check_taxon . . . 2
customized_filter . . . 5
detect_native_coord . . . 9
detect_native_country . . . 11
Distributions . . . 13
EnumOccurrenceIssue . . . 14
export_records . . . 15
import_records . . . 16
Level3maping . . . 18
map_records . . . 20
par_clean_coordinates . . . 22
print.CoordinateRefined . . . 25
print.customFiltered . . . 25
print.import . . . 26
print.nativeDetected . . . 26
print.occ_taxa . . . 27
set_threads . . . 28
VasGBIF . . . 29
WGSRPD3 . . . 34
WorldLandMap . . . 34

Index **36**

`check_taxon` *Resolve taxon names via the Taxonomic Name Resolution Service*

Description

Submits plant scientific names from a GBIF occurrence table to the Taxonomic Name Resolution Service ('TNRS') and resolves them against the World Checklist of Vascular Plants ('WCVP') or World Flora Online ('WFO'), depending on `sources` (default `"wcvp"`). Spelling errors are corrected, variant spellings are standardised, and synonyms are converted to their currently accepted names. Only records that meet the `accuracy` threshold, have an accepted or synonym `Taxonomic_status`, and resolve to an accepted name at species rank or below are returned in `occ_taxa_checked`.

Usage

```
check_taxon(
  occ_import = NA,
  accuracy = 0.85,
  sources = "wcvp",
  timeout_minutes = 20
)
```

Arguments

`occ_import` An `"import"` `data.table` returned by `import_records()`, containing at least the columns `gbifID`, `scientificName`, and `taxonRank`.

`accuracy` Numeric scalar between `0` and `1`. Minimum 'TNRS' match score for a resolution to be accepted. Defaults to `0.85`. Records whose `Overall_score` falls below this threshold are excluded from `occ_taxa_checked`. Passed directly to `TNRS::TNRS()`.

`sources` Character vector naming the taxonomic sources to resolve against. Defaults to `"wcvp"` (World Checklist of Vascular Plants). `"wfo"` (World Flora Online) is also accepted. Multiple sources can be combined, e.g. `c("wcvp", "wfo")`; the 'TNRS' API then returns the best match across all selected sources and the `Source` column records which source provided it. Passed directly to `TNRS::TNRS()`.

`timeout_minutes` Numeric scalar. Timeout per 'TNRS' chunk attempt in minutes. Defaults to `20`. If an attempt exceeds the timeout it is abandoned and retried.

Details

Taxon rank filtering:

Only records whose `taxonRank` is one of `"SPECIES"`, `"VARIETY"`, `"SUBSPECIES"`, or `"FORM"` are submitted to 'TNRS'. Records at genus rank or above are excluded from the query and are absent from both `summary` and `occ_taxa_checked`.

This check filters on the `taxonRank` reported by GBIF for the submitted record. It is distinct from the output-side check on `Accepted_name_rank` described under "Output filtering", which filters on the rank of the *resolved* accepted name returned by 'TNRS'.

Chunked submission:

Unique names are submitted in chunks of up to 4,000 to respect 'TNRS' API limits. Each name is assigned a row identifier before chunking, so the identifiers stay unique across the whole query. Results from all chunks are combined with `data.table::rbindlist()` and then de-duplicated by `ID`, guarding against occasional duplicate rows returned by the API.

Joining results back to occurrences:

'TNRS' does not echo submitted names verbatim: commas are replaced by spaces and diacritics are normalised, so `Name_submitted` in the response may differ from the string that was sent. Results are therefore joined back to the occurrence table through the row identifier echoed in the `ID` column, which maps each result to the exact `scientificName` submitted.

Output filtering:

After merging 'TNRS' results back into the occurrence table, only records satisfying **all** of the following conditions are kept in `occ_taxa_checked`:

- `Overall_score` >= `accuracy`
- `Taxonomic_status` is `"Accepted"` or `"Synonym"`
- `Accepted_name_rank` is neither empty nor `"genus"`

The rank condition excludes records whose best match only reached a genus-level or unranked accepted name, even when the score and status would pass. Records that fail any condition (unresolved names, low-confidence matches, names with uncertain status, or genus-level resolutions) are present in `summary` for manual review but absent from `occ_taxa_checked`.

Retry logic:

Each chunk is attempted up to three times. An attempt that exceeds `timeout_minutes` or that returns no rows is abandoned and retried, with a five-second pause between attempts, guarding against transient network failures. If all attempts fail for a chunk, the function stops with an informative error.

Value

An object of class `"occ_taxa"`, implemented as a named list with three elements:

- `occ_taxa_checked`: a `data.table` of occurrence records that passed the `accuracy` threshold and have an `"Accepted"` or `"Synonym"` `Taxonomic_status`. Columns from `occ_import` (`gbifID`, `scientificName`) are joined with the following 'TNRS' result columns: `Overall_score`, `Taxonomic_status`, `Accepted_name`, `Accepted_species`, `Accepted_name_id`, `Accepted_name_rank`, `Accepted_family`, `Source`.
- `summary`: a `data.table` of unique 'TNRS' resolution results (one row per submitted name), useful for reviewing match quality and identifying names that require manual attention. `scientificName` holds the string as submitted, while `Name_submitted` holds the possibly rewritten form echoed by 'TNRS'; comparing the two exposes names the API altered.
- `runtime`: a `difftime` object recording the total elapsed time.

References

Boyle, B. et al. (2013). The taxonomic name resolution service: an online tool for automated standardisation of plant names. *BMC Bioinformatics*, 14, 16. doi:10.1186/147121051416

Govaerts, R. et al. (2021). The World Checklist of Vascular Plants, a continuously updated resource for exploring global plant diversity. *Scientific Data*, 8, 215. doi:10.1038/s41597021009976

See Also

- `import_records()` for the first step that produces the `occ_import` input.
- `TNRS::TNRS()` for the underlying name resolution function.
- `print.occ_taxa()` for a compact summary of the result.

Examples

```
gbif_file <- system.file(
  "extdata",
  "0003386-260721160103020.zip",
  package = "VasGBIF"
)
occ <- import_records(path = gbif_file)
taxa_checked <- check_taxon(occ_import = occ, accuracy = 0.85)
head(taxa_checked$summary, 10)
nrow(taxa_checked$occ_taxa_checked)
```

`customized_filter` *Apply custom quality filters to occurrence records*

Description

Joins the outputs of the VasGBIF import and taxonomic-resolution steps into a single occurrence table, then progressively removes records according to a user-selected set of quality rules.

Usage

```
customized_filter(
  occ_import = NA,
  taxa_checked = NA,
  filter_countryCode = TRUE,
  filter_coordinateUncertainty = 10000,
  filter_date = FALSE,
  filter_identifiedBy = FALSE,
  filter_recordedBy = FALSE,
  filter_gbif_issues_max = 5
)
```

Arguments

`occ_import` An `"import"` `data.table` returned by `import_records()`. `gbifID` and `issue` are always required; every other column is required only by the rule that reads it, so a column backing a disabled rule may be absent. The columns are `decimalLatitude` and `countryCode` for `filter_countryCode`, `coordinateUncertaintyInMeters` for `filter_coordinateUncertainty`, `eventDate`, `month`, `year`, and `day` for `filter_date`, `identifiedBy`, and `recordedBy`.

`taxa_checked` An `"occ_taxa"` object returned by `check_taxon()`.

`filter_countryCode` Logical scalar. If `TRUE` (default), removes records with no usable geographic information: `decimalLatitude` is `NA` **and** `countryCode` is `NA` or empty. Records with either a coordinate or a country code are kept. When enabled, a `filter_countryCode` logical column is added to `occ_marked`; see *Per-rule verdict columns*.

`filter_coordinateUncertainty`
: Non-negative numeric scalar. Removes records whose `coordinateUncertaintyInMeters` is strictly greater than the threshold. Defaults to `10000`. Records with `NA` or empty `coordinateUncertaintyInMeters` are always kept - they do not participate in this rule. Pass `NULL`, `NA`, or `''` to disable the rule. The value is validated when the function is called, before any work is done. When the rule is enabled, a `filter_coordinateUncertainty` logical column is added to `occ_marked`.

`filter_date`
: Logical scalar. If `TRUE`, removes records for which all four date components (`eventDate`, `month`, `year`, `day`) are missing or empty. Defaults to `FALSE`.

`filter_identifiedBy`
: Logical scalar. If `TRUE`, removes records whose `identifiedBy` is missing or contains no named identifier (see *Collector junk detection*). Defaults to `FALSE`.

`filter_recordedBy`
: Logical scalar. If `TRUE`, removes records whose `recordedBy` is missing or contains no named collector. Defaults to `FALSE`.

`filter_gbif_issues_max`
: Non-negative numeric scalar. Removes records flagged with more GBIF issues than the threshold (`gbif_issues` > `filter_gbif_issues_max`). Defaults to `5`. Pass `NULL`, `NA`, or `''` to disable the rule. The value is validated when the function is called, before any work is done.

  Each of `filter_date`, `filter_identifiedBy`, `filter_recordedBy`, and `filter_gbif_issues_max` adds a logical column named after the argument to `occ_marked` when it is enabled; see *Per-rule verdict columns*.

Details

Joining the inputs:

The two inputs are joined by `gbifID`. For memory efficiency the joins are performed **in place** on a single defensive copy of `occ_import`: `copy()` is made once and each join adds columns via `:=`, instead of materialising a fresh full-width table per join. The caller's `occ_import` is never modified. Rows are not deleted during the join or the rules; every criterion accumulates into a logical mask and a single row subset is applied at the end. Peak memory is therefore close to the input plus one working copy, regardless of how many rules are enabled.

`check_taxon()` already removes records that fail the `accuracy` threshold, whose `Taxonomic_status` is neither `"Accepted"` nor `"Synonym"`, or that resolve to a genus-level or unranked accepted name, so `occ_taxa_checked` contains only fully resolved records. Rows absent from `occ_taxa_checked` are dropped via the keep mask instead of carrying `NA` taxonomy through the rest of the pipeline. The number removed is recorded in `summary` under the rule name `taxon_resolved`.

`extract_issues()` is called internally on `occ_import` and returns exactly one row per imported record, so the issue join is one-to-one and cannot change the row count. The function verifies this and stops if any record lacks an issue count. The raw `issue` column is removed from `occ_import` and replaced by `gbif_issues`, the per-record issue count.

Filter rules:

Every applied test inspects the full table on its own terms rather than only the records left by the preceding tests, and its verdict is stored as a logical column named after the test. `TRUE` means the record passed that test. By default `countryCode`, `coordinateUncertainty`, and

gbif_issues_max are enabled; date, identifiedBy, and recordedBy are disabled. Numeric thresholds are validated before any work is done, so an invalid value fails immediately.

The always-on taxon_resolved test is the first one applied. It is not a selectable rule: check_taxon() only retains records whose name resolved, so records absent from taxa_checked$occ_taxa_checked receive NA Accepted_name at the join and fail this test. It always contributes a filter_taxon_resolved column.

- countryCode: drop records with NA latitude **and** NA/empty countryCode.
- coordinateUncertainty: drop records with coordinateUncertaintyInMeters > threshold; records with NA or empty uncertainty are kept.
- date: drop records with eventDate, month, year, **and** day all missing.
- identifiedBy / recordedBy: drop records whose value is junk (see below).
- gbif_issues_max: drop records with gbif_issues > threshold. The one-to-one issue join guarantees every record carries an issue count, so no record is exempt from this rule.

Per-rule verdict columns:

Each applied test contributes a logical column named after it: filter_taxon_resolved (always) plus filter_countryCode, filter_coordinateUncertainty, filter_date, filter_identifiedBy, filter_recordedBy, and filter_gbif_issues_max for the selected rules. TRUE marks a record that passed that test. A disabled rule contributes no column, so the output width depends on which rules were selected. The columns are carried by occ_marked, where they identify which tests rejected each excluded record; occ_filtered holds only occurrence and joined columns.

Because each rule is evaluated independently, one record can fail several tests at once and is then counted by each of them. The summary table makes this explicit:

- failed is the number of records that test rejects on its own terms; these counts overlap and sum to more than the number of records removed.
- only_failed_here is the number of records that test rejects and no other test does; these counts are disjoint but incomplete, since a record failing several tests is credited to none of them.
- dropped and remaining remain the sequential attribution: each removed record is credited to the first enabled test that rejected it, so dropped sums exactly to the number of records removed.

The split of the output reflects the same distinction. occ_filtered holds the records that passed every applied test, with their occurrence and joined columns and no verdict columns; occ_marked holds the rest, reduced to gbifID and the verdict columns. Because every applied test has a column, each marked record carries at least one FALSE and shows exactly which tests rejected it - a record excluded only because its taxon name did not resolve has FALSE in filter_taxon_resolved alone.

Collector junk detection:

The identifiedBy and recordedBy rules are identical in strictness. A value is treated as junk - and the record removed - when it is missing or empty, or when it matches a curated set of "no named person" patterns while containing **no name separator**.

The keyword list covers English (unknown, anonymous, unnamed, unidentified, unrecorded, incognito), other languages (desconocido, desconhecido, anonimo and its accented Spanish and Portuguese variants, sin nombre, sem nome, inconnu, anonyme, unbekannt, anonym, and four Chinese terms meaning "unknown", "unnamed", "anonymous" and "no details"), and whole-value patterns such as s.n., n/a, et al., and no collector.

Name separators (`,`, `;`, `&`, `+`, and, plus the full-width comma, full-width semicolon, and ideographic enumeration comma used in CJK text) protect values that mix a keyword with a real name, e.g. `"Unknown; Jongmans WJ"` or `"Collector(s): Eric Sundell, unknown"` are kept because a named person is present.

Word-boundary matching means the Chinese keywords only match standalone values; a longer phrase that merely begins with one of them (for instance the Chinese for "unknown collector") is not removed. This is deliberately conservative: without boundaries, a real name that happens to contain a keyword could be wrongly dropped.

Value

An object of class `"customFiltered"`, implemented as a named list with three elements:

- `occ_filtered`: the records that passed every applied test, as a `data.table` with the occurrence and joined columns only - no verdict columns.
- `summary`: a `data.table` with columns `rule`, `dropped`, `remaining`, `failed`, and `only_failed_here`, giving for each applied step (the `taxon_resolved` join plus each enabled rule) how many records it removed, how many remained, and how many it rejects under the independent and exclusive readings described in *Per-rule verdict columns*.
- `occ_marked`: the records excluded by at least one applied test, reduced to `gbifID` and one verdict column per applied test (see *Per-rule verdict columns*). Every such record fails at least one of them, so its verdict columns contain at least one `FALSE`. Together with `occ_filtered`, this table accounts for every record in `occ_import`.

A `print()` method for class `"customFiltered"` displays how many records were kept and excluded, and the per-rule summary.

See Also

- `import_records()` and `check_taxon()` for the two inputs.
- `extract_issues()` for the issue count behind the `gbif_issues_max` rule.

Examples

```
gbif_file <- system.file(
  "extdata",
  "0003386-260721160103020.zip",
  package = "VasGBIF"
)
occ <- import_records(path = gbif_file)
taxa_checked <- check_taxon(occ_import = occ, accuracy = 0.85)

filtered <- customized_filter(
  occ_import = occ,
  taxa_checked = taxa_checked,
  filter_date = TRUE,
  filter_identifiedBy = TRUE,
  filter_recordedBy = TRUE
)
filtered
```

```
filtered$summary

# The kept records carry the occurrence and joined columns only:
names(filtered$occ_filtered)

# The excluded records carry `gbifID` and one verdict column per applied
# test, so each shows exactly which tests rejected it:
names(filtered$occ_marked)
head(filtered$occ_marked)

# Disable the coordinate-uncertainty rule (NULL / NA / '' all work). The
# rule contributes no verdict column:
filtered_loose <- customized_filter(
  occ_import = occ,
  taxa_checked = taxa_checked,
  filter_coordinateUncertainty = NULL
)
"filter_coordinateUncertainty" %in% names(filtered_loose$occ_marked)
```

`detect_native_coord` *Detect native status from WGSRPD distributions*

Description

Assigns a native status classification to each occurrence record by matching it against WCVP distribution data (the internal `Distributions` dataset) via WGSRPD Level 3 areas. Classification uses only the spatial stage: the records in `input` are overlaid on the WGSRPD Level 3 polygon map (via `terra::extract()`) to assign an area code to each record. That area code is looked up in a distribution table classified from the WCVP flags (`introduced`, `extinct`, `location_doubtful`) with the following priority:

1. If `location_doubtful == 1`, the area is classified as `"location_doubtful"` regardless of other flags.
2. Otherwise, if `introduced == 1`, the area is `"introduced"`.
3. Otherwise, if `extinct == 1`, the area is `"extinct"`.
4. If all three flags are `0`, the area is `"native"`.
5. Any remaining case defaults to `"unknown"`.

A record that falls in several areas at once is assigned the most preferred status (`"native"` first). Unresolved records may be buffered (`buffer_km`) so coastal points just outside a polygon can still be matched; buffered hits are ranked below exact ones.

Every record must carry a usable coordinate. Records without one are **not** classified here; classify them with `detect_native_country()`, which matches them through their `countryCode` without using geometry.

Usage

```
detect_native_coord(
  input = NA,
  species = "Accepted_name",
  longitude = "decimalLongitude",
  latitude = "decimalLatitude",
  buffer_km = 10,
  buffer_chunk_size = 2000
)
```

Arguments

`input` A table holding one occurrence record per row, with the species and coordinate columns named by `species`, `longitude` and `latitude`. Anything `data.table::as.data.table()` can convert is accepted, so a `data.frame`, `data.table`, tibble or list of equal-length columns all work. Typically the `CoordinateCleaned` table of a `CoordinateRefined` object returned by `par_clean_coordinates()`. Every record must carry a non-missing coordinate.

`species`, `longitude`, `latitude` Names of the columns in `input` holding the species name and the coordinates. Default to the GBIF field names `"Accepted_name"`, `"decimalLongitude"` and `"decimalLatitude"`.

`buffer_km` Numeric scalar. Width of the spatial buffer in km applied to records the exact spatial match left unresolved. `0` disables the buffer. Defaults to `10`.

`buffer_chunk_size` Numeric scalar. Maximum number of records buffered in one chunk, keeping the relate matrix small. Defaults to `2000`.

Details

Coordinate reference system. Both the occurrence points and the internal `WGSRPD3` polygons are assumed to be in longitude/latitude (EPSG:4326); the function asserts this on the polygon side. `buffer_km` is applied as metres via `terra::buffer()`'s geodesic buffer, so it keeps the same meaning at every latitude.

Input. Any table-like object that `data.table::as.data.table()` can convert is accepted - a `data.frame`, a `data.table`, a tibble, or a list of equal-length columns - provided it carries the species and coordinate columns named by `species`, `longitude` and `latitude`. It does not have to come from `par_clean_coordinates()`. Pass the validated-coordinate table (for example `refined_coordinates$Coordina` so the records that failed coordinate validation are left to `detect_native_country()` instead of being classified by geometry. If `input` has no `gbifID`, one is created as a character sequence number. The column-name arguments are independent of the data: a column that happens to share a name with one of them (for example a `species` column) does not interfere, and `species = "species"` is a valid way to select it.

Value

A `nativeDetected` object - a `data.table` subclass with one row per input record (every row of `input`), keyed by `gbifID`. Every column of the input records is retained unchanged, with three

classification columns appended:

- `LEVEL3_COD`: the assigned WGSRPD Level 3 area code, or `NA` if the record could not be matched
- `native_status`: one of `"native"`, `"introduced"`, `"extinct"`, `"location_doubtful"`, or `"unknown"`
- `native_status_source`: how the status was inferred. `"exact"` is a direct spatial match, `"buffered"` a match obtained through the geodesic buffer, and `"unmatched"` a record that matched no area at all.

The intermediate matching columns used internally (taxon keys, candidate areas, match ranks) are not returned. Because the record columns are carried through, the result holds a second copy of the input data: for large inputs, the input table can be dropped once the classification is in hand.

See Also

`detect_native_country()` for records without coordinates, `print.nativeDetected()` for a compact summary of the result.

Examples

```
# Classify the records with validated coordinates. `refined_coordinates`
# comes from `par_clean_coordinates()`, whose example creates it when run
# first.
native_coord <- detect_native_coord(refined_coordinates$CoordinateCleaned)
native_coord
```

`detect_native_country` *Detect native status from country codes*

Description

Assigns a native status classification to occurrence records by matching their country code against WCVP distribution data (the internal `Distributions` dataset) via WGSRPD Level 3 areas. The country code is mapped to candidate Level 3 areas by the `Level3maping` table; no geometry is used. The same flag priority as `detect_native_coord()` applies:

1. If `location_doubtful == 1`, the area is classified as `"location_doubtful"` regardless of other flags.
2. Otherwise, if `introduced == 1`, the area is `"introduced"`.
3. Otherwise, if `extinct == 1`, the area is `"extinct"`.
4. If all three flags are `0`, the area is `"native"`.
5. Any remaining case defaults to `"unknown"`.

This stage needs no geometry, so it classifies every record handed to it, including records that lack coordinates entirely. Records with validated coordinates are better classified with `detect_native_coord()`, whose spatial match is more precise.

Usage

```
detect_native_country(
  input = NA,
  species = "Accepted_name",
  country = "countryCode"
)
```

Arguments

`input` A table holding one occurrence record per row, with the species and country-code columns named by `species` and `country`. Anything `data.table::as.data.table()` can convert is accepted. Typically the `CoordinateProblematic` table of a `CoordinateRefined` object returned by `par_clean_coordinates()`. Missing coordinates are allowed: no geometry is used.

`species, country` Names of the columns in `input` holding the species name and the country code. Default to the GBIF field names `"Accepted_name"` and `"countryCode"`.

Details

`L3 ISOcode` in `Level3maping` is reproduced as published and is **not** a complete or one-to-one concordance. A single ISO code usually maps to several Level 3 units (for example `CN` maps to eight), so a country code identifies a *set* of candidate areas; a record is assigned the most preferred status among the areas its taxon occurs in (`"native"` first).

Every input record is returned. A record whose country code is missing or empty, or maps to no Level 3 area, is kept with `native_status_source = "unmatched"`; a record whose country maps to areas but whose taxon has no distribution entry there is `"no_entry"`.

Input. Any table-like object that `data.table::as.data.table()` can convert is accepted - a `data.frame`, a `data.table`, a tibble, or a list of equal-length columns - provided it carries the species and country-code columns named by `species` and `country`. No coordinate column is required. Pass the problematic-coordinate table (for example `refined_coordinates$CoordinateProblematic`). If `input` has no `gbifID`, one is created as a character sequence number. The column-name arguments are independent of the data: a column that happens to share a name with one of them (for example a `species` column) does not interfere, and `species = "species"` is a valid way to select it.

Value

A `nativeDetected` object - a `data.table` subclass with one row per input record (every row of `input`), keyed by `gbifID`. Every column of the input records is retained unchanged, with three classification columns appended:

- `LEVEL3_COD`: the assigned WGSRPD Level 3 area code, or `NA` if the record could not be matched
- `native_status`: one of `"native"`, `"introduced"`, `"extinct"`, `"location_doubtful"`, or `"unknown"`

- `native_status_source`: `"country"` for a mapped hit; `"no_entry"` when the country mapped to areas but the taxon had no distribution entry there; `"unmatched"` when the record had no usable country code.

The intermediate matching columns used internally (taxon keys, candidate areas, match ranks) are not returned.

See Also

`detect_native_coord()` for records with validated coordinates, `print.nativeDetected()` for a compact summary of the result.

Examples

```
# Classify the records that failed coordinate validation. `refined_coordinates`
# comes from `par_clean_coordinates()`, whose example creates it when run
# first.
native_country <- detect_native_country(
  refined_coordinates$CoordinateProblematic
)
native_country
```

`Distributions` *The World Checklist of Vascular Plants Distributions*

Description

A processed dataset containing the distribution data from the WCVP, mapped to the Biodiversity Information Standards (TDWG) World Geographical Scheme for Recording Plant Distributions (WGSRPD)

Usage

```
Distributions
```

Format

A data.table with 1,647,045 rows and 6 variables:

plant_name_id World Checklist of Vascular Plants (WCVP) identifier

taxon_name Concatenation of genus with species and, where applicable, infraspecific epithets to make a binomial or trinomial name

area_code_l3 WGSRPD Level 3 code

introduced 0 if native; 1 if introduced

extinct 1 if extinct; 0 if extant

location_doubtful 1 if doubtful; 0 otherwise

Source

https://sftp.kew.org/pub/data-repositories/WCVP/wcvp.zip

References

Govaerts, R., Nic Lughadha, E., Black, N. et al. The World Checklist of Vascular Plants, a continuously updated resource for exploring global plant diversity. *Sci Data* 8, 215 (2021). doi:10.1038/s41597021009976

EnumOccurrenceIssue *Enumeration of GBIF occurrence issues*

Description

An enumeration of validation rules for single occurrence records.

Usage

```
EnumOccurrenceIssue
```

Format

A data.table with 69 rows and 9 columns

constant GBIF issue constant

description GBIF issue description

definition Our definition for classifying geographic issues

type Type issue

priority Impact of the issue for the use of geospatial information

score Impact, in number, of the issue for the use of geospatial information

selection_score Value used to calculate the quality of the geospatial information according to the classification of the issue

reasoning Reasoning of the impact of the theme for the use of geospatial information

notes Notes

Details

There are many things that can go wrong and we continuously encounter unexpected data. In order to help us and publishers improve the data, we flag records with various issues that we have encountered. This is also very useful for data consumers as you can include these issues as filters in occurrence searches. Not all issues indicate bad data. Some merely flag the fact that GBIF has altered values during processing. On the details page of any occurrence record you will see the list of issues in the notice at the bottom.

Source

- GBIF Infrastructure: Data processing
- An enumeration of validation rules for single occurrence records

---

export_records — *Export classified records to compressed CSV files*

Description

Writes the results of `detect_native_coord()` and/or `detect_native_country()` to disk as gzip-compressed CSV files. Each classification supplied produces two files: all of its classified records and the native subset.

Usage

```
export_records(
  native_detected_coord = NA,
  native_detected_country = NA,
  export_path = NA
)
```

Arguments

`native_detected_coord`
: A `nativeDetected` object returned by `detect_native_coord()`, containing records with validated coordinates. It must carry `gbifID`, `native_status`, `decimalLongitude` and `decimalLatitude`, and none of its coordinates may be missing. Optional, but at least one of `native_detected_coord` and `native_detected_country` must be supplied.

`native_detected_country`
: A `nativeDetected` object returned by `detect_native_country()`, containing the records classified through their `countryCode`. It must carry `gbifID` and `native_status`; missing coordinates are allowed. Optional, but at least one of the two classification arguments must be supplied.

`export_path`
: Directory where the compressed CSV files should be written.

Details

Each classification supplied is written as its own pair of files, so the two sets never overwrite each other:

- `all_records_coord.csv.gz`: every record classified by `detect_native_coord()`, with all its columns and its native status
- `native_records_coord.csv.gz`: the subset of those records classified as `"native"`
- `all_records_country.csv.gz`: every record classified by `detect_native_country()`, with all its columns and its native status

- native_records_country.csv.gz: the subset of those records classified as "native"

Only the pair belonging to a supplied argument is written, so supplying one argument leaves the files of the other untouched. The two sets are exported separately rather than bound together: this function never writes a combined table.

The two all_records_* files are likewise not a partition of a single input and cannot be row-bound with each other. They carry different columns - the coordinate result comes from CoordinateCleaned, whereas the country-code result comes from CoordinateProblematic, which keeps its per-test verdict columns - and detect_native_country() classifies only the records it receives with a usable countryCode.

Both arguments already carry every column of the input records, so no join is performed here: each table is written straight from the object.

native_detected_coord must carry coordinates for every record: it is the spatial output of detect_native_coord(), which only classifies records with validated coordinates. The function stops if decimalLongitude or decimalLatitude is missing for any of its records, which catches the two arguments being swapped. native_detected_country carries no such requirement, because detect_native_country() classifies the records that failed coordinate validation - including records whose coordinates are entirely absent.

Files are written with fwrite(encoding = "UTF-8"). export_path is validated before writing: if it is not a single character path or exists as a file (not a directory), the function stops with an error; if it does not exist, a warning is emitted and the directory is created automatically.

Value

Called for its side effect of writing files to export_path. Returns NULL invisibly.

See Also

detect_native_coord(), detect_native_country(), par_clean_coordinates()

---

import_records *Import GBIF occurrence records*

Description

Reads a GBIF occurrence download and returns a data.table of the fields required by the 'VasGBIF' workflow.

Usage

```
import_records(path = "", tempdir = NULL, remove_tempfile = TRUE)
```

Arguments

`path` Character scalar. Path to a GBIF occurrence download: either a 'SIMPLE_CSV' or 'DWCA' ZIP archive, or an extracted tab-separated occurrence table. An archive must contain a tab-separated occurrence data file, named `occurrence.txt` when the archive contains more than one member (as in a Darwin Core Archive).

`tempdir` Character scalar or `NULL`. Directory into which the ZIP archive is extracted. Ignored when `path` is not a ZIP file.

- `NULL` (default): a unique subdirectory is created inside the system temporary directory via `base::tempfile()`, and it is deleted on exit unless `remove_tempfile = FALSE`.
- A user-supplied path: if the directory does not exist it is created (recursively). If it already exists and contains files, a warning is issued because those files may be overwritten. The directory is never deleted; on exit only the extracted occurrence file is unlinked, and even that is skipped when `remove_tempfile = FALSE`.

`remove_tempfile` Logical scalar. Controls whether the files this call produced are deleted when the function exits (including after an error). Defaults to `TRUE`.

- With `tempdir = NULL`, the extraction directory is created by this call and holds nothing else, so the directory is removed whole.
- With a user-supplied `tempdir`, only the extracted occurrence file is unlinked. The directory itself and every unrelated file in it are left untouched.
- `FALSE` keeps the extracted file in `ex_path` and reports its location.

Details

Two kinds of input are accepted, chosen by the file extension:

- A ZIP archive in 'SIMPLE_CSV' or Darwin Core Archive ('DWCA') format. The occurrence data file is extracted and read from there.
- An already extracted occurrence table in any other file, e.g. `.csv`, `.txt` or `.tsv`. The extension is not used to infer the format: the file must be tab-separated regardless.

In both cases the first column must be named `gbifID`; anything else stops with an error. This also catches comma-separated files, whose header then arrives as a single column.

An archive is extracted into a dedicated directory rather than next to the ZIP file, so a ZIP stored in `inst/extdata` is never modified.

The function performs the following steps:

- Validates that `path` is a non-empty single character string.
- Resolves the file to read. For a `.zip` path the archive members are listed: a 'SIMPLE_CSV' download holds a single data file which is extracted by name, while a 'DWCA' holds several members (typically `meta.xml`, `occurrence.txt`, and extension files) and its `occurrence.txt` core file is assumed. For any other path the file itself is read directly.
- Reads the header of the resolved file with `data.table::fread()` using `nrows = 0`, and requires its first column to be named `gbifID`. Only the header is parsed, so this check is nearly free and rejects unusable files before the full read.

- Reads the tab-separated, UTF-8 occurrence file with `data.table::fread()`, selecting only the GBIF fields used by VasGBIF.
- Coerces `gbifID` to character.

Value

A `data.table` of class `"import"` containing the selected occurrence fields with Darwin Core / GBIF column names. The `gbifID` column is always character.

References

GBIF.org (23 July 2026) GBIF Occurrence Download doi:10.15468/dl.nt5exp

See Also

- `print.import()` for a one-line record count.
- `data.table::fread()` for delimited-file import.
- `unzip()` for ZIP archive handling.
- GBIF download formats for the difference between 'SIMPLE_CSV' and 'DWCA' downloads.

Examples

```
gbif_file <- system.file(
  "extdata",
  "0003386-260721160103020.zip",
  package = "VasGBIF"
)
occ <- import_records(path = gbif_file)
occ_import <- extract_gbif_issues(occ)
head(occ_import$summary, 5)
```

`Level3maping` *WGSRPD Level 3 area codes, names and country concordance*

Description

The tabular component of the Biodiversity Information Standards (TDWG) World Geographical Scheme for Recording Plant Distributions (WGSRPD): one row per Level 3 unit ("Botanical Country"), giving its code, name, parent Level 2 region, and - where the source standard supplies one - the ISO 3166-1 alpha-2 code of the country it belongs to.

Usage

```
Level3maping
```

Format

A data.table with 369 rows and 6 variables:

L3 code Three-letter WGSRPD Level 3 code, e.g. `"AGE"`. Unique; joins to `LEVEL3_COD` in WGSRPD3 and to `area_code_l3` in Distributions.

L3 area Name of the Level 3 unit, e.g. `"Argentina Northeast"`. Boundaries follow political borders or coastlines, but the units are botanical rather than political: large countries are subdivided, and small ones may be grouped.

L2 code Code of the parent Level 2 (regional) unit. Stored as text with a comma decimal mark, e.g. `"85,00"`, as in the source file.

L3 ISOcode ISO 3166-1 alpha-2 country code, or `""` when the source standard gives none. See Details before using this column to map country codes onto areas.

Ed2status Amendment flag carried over verbatim from the source table. Empty for 366 areas; `"N"` for Belarus (BLR) and Suriname (SUR), `"T"` for the Chagos Archipelago (CGS). The source standard should be consulted for the meaning of these codes.

Notes Free-text note from the source table. Populated for three areas only: alternative spellings for Belarus and Suriname, and a Level 2 code remark for the Chagos Archipelago.

Details

Level 3 is the level at which the World Checklist of Vascular Plants records distributions, so `L3 code` is the key shared with Distributions and with the polygon map in WGSRPD3. All 369 codes are present in `WGSRPD3`.
`L3 ISOcode` is reproduced as published and is **not** a complete or one-to-one concordance. Three properties matter when using it to translate an occurrence record's `countryCode` into candidate Level 3 areas:

- **Many areas share one country code.** A single ISO code usually maps to several Level 3 units: `US` to 51, `RU` to 21, `CA` to 13, `CN` and `AU` to 8 each. Only 138 of the represented countries map to exactly one area, so an ISO code generally identifies a *set* of areas, not one area.
- **42 areas carry no ISO code at all**, among them the mainland units for France (FRA), Italy (ITA), Spain (SPA), Austria (AUT), Belgium (BGM), Ireland (IRE), Ukraine (UKR) and Korea (KOR). Because some offshore units of the same countries *do* carry a code, a naive lookup silently returns the islands alone: `"FR"` resolves to Corse, `"IT"` to Sardegna, and `"ES"` to the Baleares and Canary Islands. This yields a plausible but wrong area rather than no answer.
- **One code is inconsistent with current usage.** Belarus (BLR) is listed under ISO `RU`, reflecting the standard's 2001 vintage; records coded `BY` therefore find no area, and records coded `RU` gain Belarus as a candidate.

Source

- tblLevel3.txt
- World Geographical Scheme for Recording Plant Distributions (WGSRPD)

References

R. K. Brummitt. 2001. World Geographical Scheme for Recording Plant Distributions, Edition 2. Hunt Institute for Botanical Documentation, Carnegie Mellon University (Pittsburgh). http://rs.tdwg.org/wgsrpd/doc/data/

See Also

WGSRPD3 for the matching polygons, Distributions for the distribution records keyed on these codes.

`map_records` *Visualize refined records on interactive maps*

Description

Renders refined occurrence records on interactive maps for spatial exploration and quality assessment. Records are deduplicated with geohashes and color-coded by their native status.

Usage

```
map_records(
  native_detected_coord = NA,
  species = "all",
  precision = 3,
  cex = 3
)
```

Arguments

`native_detected_coord`
: A `nativeDetected` object returned by `detect_native_coord()`, containing records with validated coordinates.

`species`
: Either `"all"` (the default) to map every species, or a single species name to restrict the map to that species. The name is matched exactly against the `Accepted_name` column, so `"Saxifraga hirculus"` maps the records of that species. Records whose `native_status` is `"unknown"` are excluded either way.

`precision`
: Positive integer controlling the spatial resolution of geohash-based deduplication. Higher values produce finer-grained cells. For reference, precision values of 4, 3, and 2 represent approximately 20 km, 156 km, and 1,250 km resolution, respectively. Defaults to `3`.

`cex`
: Numeric value controlling the point size of occurrence records on the map. Defaults to `3`.

Details

The function works in four steps:

- **Record selection:** Reads the classified records from `detect_native_coord()`, keeping those whose `native_status` is not `"unknown"`, optionally restricted to a single species named by `species`.
- **Geohash deduplication:** Encodes coordinates at the requested precision and retains one representative record per species, geohash cell, and native status.
- **Interactive visualization:** Builds a multi-layer map with records color-coded by `native_status`.
- **Basemap selection:** Provides OpenStreetMap, Esri World Imagery, and Stadia Stamen Watercolor basemaps.

Record selection:

Both the classification and the coordinates are read from `native_detected_coord`, which carries every column of the input records. Records with `native_status = "unknown"` are excluded. When `species` names a species rather than `"all"`, the selection is narrowed further to the records whose `Accepted_name` equals that name; the comparison is exact, so the spelling must match the column value. A name that selects nothing is an error, because an empty map is read as "this species has no native records". Records with missing longitude or latitude are excluded as a guard; `detect_native_coord()` only classifies records with validated coordinates, so none are expected, and inputs that carry missing coordinates (such as the output of `detect_native_country()`) are rejected outright.

Geohash deduplication:

Geohash encoding converts latitude-longitude pairs into alphanumeric strings representing grid cells of varying sizes. The function groups records by species name, geohash cell, and native status, then retains the first record from each group. This reduces visual overplotting while preserving the broad spatial distribution pattern, which is useful for densely sampled regions.

Map layers and interactivity:

The generated map includes:

- A color-coded legend based on `native_status` categories.
- Popups displaying record attributes such as GBIF ID, GBIF issues, and taxon name.
- Toggleable basemap layers for different visualization contexts.
- Point transparency set to `alpha.regions = 0.6` to improve density perception.

Value

A `mapview` interactive map object displaying refined occurrence records color-coded by `native_status`. The map contains a native-status legend, three switchable basemap layers, and clickable popups with record metadata.

See Also

`detect_native_coord()` for the object consumed by this function; `gh_encode()` for geohash encoding and `mapView()` for interactive map construction.

Examples

```
# Every species in the classification
map_records(
  native_detected_coord = native_detected_coord,
  precision = 3,
  cex = 3
)

# A single species
map_records(
  native_detected_coord = native_detected_coord,
  species = "Saxifraga hirculus"
)
```

---

par_clean_coordinates *Validate coordinates of filtered occurrence records*

Description

Validates the coordinates of occurrence records with CoordinateCleaner and splits them into coordinate-clean and problematic tables. Any table with a taxon column and complete coordinate columns is accepted, so the function does not depend on the output of `customized_filter()`.

CoordinateCleaner::clean_coordinates checks run in parallel to flag common spatial issues such as centroids, capitals, and marine records.

Records that pass all requested tests are returned in `CoordinateCleaned`; records that fail one or more tests, along with records lacking coordinates, are returned in `CoordinateProblematic`. The latter carries one logical column per test applied, so a flagged record can be traced to the specific test(s) it failed. Native-status classification is a separate step: `detect_native_coord()` classifies the records with validated coordinates, and `detect_native_country()` the coordinate-less records (extracted from `CoordinateProblematic`).

Usage

```
par_clean_coordinates(
  input = NA,
  species = "Accepted_name_id",
  longitude = "decimalLongitude",
  latitude = "decimalLatitude",
  threads = 4,
 tests = c("capitals", "centroids", "equal", "gbif", "institutions", "outliers", "seas",
    "zeros")
)
```

Arguments

`input` A table of occurrence records, with coordinates that are complete, i.e. neither longitude nor latitude may be `NA`; records missing either are returned in `CoordinateProblematic` without being validated. Typically `filtered$occ_filtered`, the table returned by `customized_filter()`. If the table has no `gbifID` column, one is created as a character sequence number over the input rows.

`species` Name of the column giving the taxon each record belongs to, used to group records for the `outliers` test. Defaults to `"Accepted_name_id"`, the column produced by `customized_filter()`.

`longitude, latitude` Names of the longitude and latitude columns. Default to `"decimalLongitude"` and `"decimalLatitude"`, the names produced by `customized_filter()`.

`threads` Number of threads to use for coordinate validation, passed to `set_threads()`. Use an integer `>= 1` for an absolute count, or a value between `0` and `1` for a proportion of available cores. The default is `4`.

`tests` Character vector of CoordinateCleaner validation tests to apply. Choose one or more of `"capitals"`, `"centroids"`, `"equal"`, `"gbif"`, `"institutions"`, `"outliers"`, `"seas"`, and `"zeros"`. The default uses all tests. Each test adds its own verdict column to `CoordinateProblematic`; see *Verdict columns*.

Details

Coordinate validation:

Coordinate validation is performed with CoordinateCleaner. Available tests are:

- `capitals`: records at country capital coordinates
- `centroids`: records at country or province centroids
- `equal`: records with identical latitude and longitude values
- `gbif`: records matching known GBIF geospatial issues
- `institutions`: records at known herbarium or museum coordinates
- `outliers`: geographic outliers within a species range
- `seas`: records located in marine areas for terrestrial species
- `zeros`: records at coordinates `(0, 0)`

The `seas` test is evaluated against the bundled `WorldLandMap` land polygons buffered by 5 km. The buffer absorbs records that a coarse coastline places just offshore, which would otherwise be reported as sea records; it is applied because the map is a low-resolution outline, not because those records are expected to be at sea.

Parallel processing:

Records with complete coordinates are chunked across the requested number of workers and validated with `foreach` and `doParallel`. The worker count is capped to the number of species and of records, so no worker is left idle.

Chunks are built from species, not from consecutive rows: every record of a species is kept in one chunk, and whole species are assigned to the chunk that currently holds the fewest records, which keeps chunk sizes close to equal. Records of a species must not be split across workers because the `outliers` test judges each record against the distribution of its own species, so splitting them would change which records are flagged.

Verdict columns:

`CoordinateProblematic` gains one logical column per test listed in `tests`, named after the corresponding CoordinateCleaner flag (`.cap` for `capitals`, `.cen` for `centroids`, `.equ` for `equal`, `.gbf` for `gbif`, `.inst` for `institutions`, `.otl` for `outliers`, `.sea` for `seas`, and `.zer` for `zeros`). `TRUE` means the record passed that test, `FALSE` means it failed it, and records that never reached validation keep `NA` throughout, which is how coordinate-less records can be told apart from flagged ones. These columns are appended after the occurrence columns. `CoordinateCleaned` holds records that passed every test and keeps the occurrence columns only.

Empty input:

If no records have complete coordinates, validation is skipped. An empty `CoordinateCleaned` table is returned, while records lacking coordinates are placed in `CoordinateProblematic`, with every verdict column set to `NA`.

Value

A `CoordinateRefined` object (list) with three elements:

- `CoordinateCleaned`: a `data.table` of records that passed all requested coordinate tests (complete data with valid coordinates)
- `CoordinateProblematic`: a `data.table` containing (1) records that failed one or more coordinate tests, and (2) records lacking complete coordinates (missing latitude or longitude). The two kinds of record are told apart by the verdict columns described in *Verdict columns*: they are all `NA` for the coordinate-less records and for the other records at least one is `FALSE`, the columns for the remaining tests being `TRUE`
- `runtime`: the elapsed execution time

Use `detect_native_coord()` to classify the records with validated coordinates, and `detect_native_country()` for the coordinate-less records from `CoordinateProblematic`.

References

- Zizka, A., Silvestro, D., Andermann, T., Azevedo, J., Duarte Ritter, C., Edler, D., Farooq, H., Herdean, A., Ariza, M., Scharn, R., Svantesson, S., Wengstrom, N., Vitecek, S., & Antonelli, A. (2019). CoordinateCleaner: Standardized cleaning of occurrence records from biological collection databases. *Methods in Ecology and Evolution*, 10(5), 744-751. doi:10.1111/2041-210X.13152

See Also

`CoordinateCleaner::clean_coordinates()`, `customized_filter()`, `detect_native_coord()`, `detect_native_country()`, `export_records()`, `print.CoordinateRefined()`, `set_threads()`

Examples

```
cleaned_coordinates <- par_clean_coordinates(filtered$occ_filtered, threads = 4)

cleaned_coordinates
# the test each flagged record failed
cleaned_coordinates$CoordinateProblematic[.otl == FALSE]
```

```
# the column names can be mapped explicitly instead
par_clean_coordinates(
  filtered$occ_filtered,
  species = "Accepted_name_id",
  longitude = "decimalLongitude",
  latitude = "decimalLatitude"
)
```

---

print.CoordinateRefined *Print a* `CoordinateRefined` *object*

Description

Displays a compact summary of a coordinate-refinement result: the number of records in each output table and the elapsed runtime. The records themselves are not shown; use `head()` or `View()` to inspect them.

Usage

```
## S3 method for class 'CoordinateRefined'
print(x, ...)
```

Arguments

`x`	An object of class `"CoordinateRefined"` returned by `par_clean_coordinates()`.
`...`	Additional arguments (unused, retained for S3 compatibility).

Value

Invisibly returns `x`.

---

print.customFiltered *Print a* `customFiltered` *object*

Description

Displays a concise summary of the filtering result: the number of records before and after filtering, how many were excluded, and the per-rule table (`dropped`, `remaining`, `failed`, and `only_failed_here`). The excluded records are available in `x$occ_marked`.

Usage

```
## S3 method for class 'customFiltered'
print(x, ...)
```

Arguments

x	An object of class "customFiltered" returned by customized_filter().
...	Additional arguments (unused, retained for S3 compatibility).

Value

Invisibly returns x.

---

print.import *Print an* import *object*

Description

Displays a one-line summary of an imported GBIF download: the number of occurrence records.

Usage

```
## S3 method for class 'import'
print(x, ...)
```

Arguments

x	An object of class "import" returned by import_records().
...	Additional arguments (unused, retained for S3 compatibility).

Value

Invisibly returns x.

---

print.nativeDetected *Print a* nativeDetected *object*

Description

Displays a compact summary of a native status classification: the number of records, and the counts by native_status and by native_status_source. The records themselves are not shown; use head() or View() to inspect them.

Usage

```
## S3 method for class 'nativeDetected'
print(x, ...)
```

Arguments

x	An object of class "nativeDetected" returned by detect_native_coord() or detect_native_country().
...	Additional arguments (unused, retained for S3 compatibility).

Value

Invisibly returns x.

---

print.occ_taxa *Print an* occ_taxa *object*

Description

Displays a summary of a taxonomic resolution result based on occ_taxa_checked: the number of records, the number of distinct scientificName values, a table of distinct name counts (scientificName, Accepted_name, Accepted_species), the Source breakdown, and the top three Accepted_name values by record count, followed by the elapsed runtime. The records themselves are not shown; use head() or View() to inspect them.

Usage

```
## S3 method for class 'occ_taxa'
print(x, ...)
```

Arguments

x	An object of class "occ_taxa" returned by check_taxon().
...	Additional arguments (unused, retained for S3 compatibility).

Value

Invisibly returns x.

set_threads — *Normalize the number of worker threads*

Description

Converts either an absolute thread count or a proportion of available CPU cores into an integer number of worker threads. The number of available cores is determined by `parallel::detectCores()`.

Usage

```
set_threads(x)
```

Arguments

`x`	A numeric scalar. Values greater than or equal to `1` specify an absolute number of threads and are rounded to the nearest integer. Values strictly between `0` and `1` specify a proportion of available cores; the resulting number of threads is also rounded to the nearest integer.

Details

Absolute thread counts exceeding the number of cores reported by `parallel::detectCores()` are silently capped to that limit with a message. Values less than or equal to zero and non-numeric inputs produce an error.

This function is also used by `par_clean_coordinates()` to normalize its `threads` argument.

Value

An integer-like numeric scalar giving the normalized number of worker threads. A message reports the selected and available thread counts.

See Also

`parallel::detectCores()`, `par_clean_coordinates()`

Examples

```
set_threads(1)
set_threads(0.5)
```

VasGBIF *Fast and Easy Compilation of Vascular Plants Occurrence Records from GBIF*

Description

GBIF hosts over 500 million tracheophyte (vascular plant) occurrence records, but processing them typically involves complex workflows and substantial computational overhead. **VasGBIF** is a fast, plant-optimised R package that parses, validates, and consolidates GBIF occurrence records into analysis-ready datasets.

VasGBIF integrates taxonomic resolution via the TNRS, spatial validation via CoordinateCleaner, and native-range annotation against WCVP and WGSRPD within a single high-performance pipeline. Its vectorised design and selective parallelisation let it process one million records on a laptop in under 15 minutes without specialised hardware.

Details

Pipeline overview:

VasGBIF provides a reproducible workflow organised into seven sequential steps. Each step progressively filters records through taxonomic, quality, and coordinate checks, transforming GBIF occurrence records into analysis-ready datasets.

1. **Import Records** - `import_records()`: reads a GBIF occurrence download ZIP ('SIMPLE_CSV' or Darwin Core Archive), extracts the occurrence table, and returns an `"import"` data.table of the fields required by the workflow. No records are filtered at this step - all diagnostic flags are preserved for later quality scoring.
2. **Check Taxon Name** - `check_taxon()`: submits species- and infraspecific-rank names to the Taxonomic Name Resolution Service (TNRS; Boyle et al. 2013) for resolution against the World Checklist of Vascular Plants (WCVP) or World Flora Online (WFO). Synonyms are resolved to accepted names; records that fail the match-score threshold or lack an accepted/synonym status are excluded from the downstream table and reported in the `summary` for manual review.
3. **Custom Filter** - `customized_filter()`: joins the imported records with the resolved taxonomy and the parsed issue counts, then applies the enabled filter rules (country code, coordinate uncertainty, GBIF issue count, event date, collector and identifier fields) to retain only high-quality records. Every rule is independently toggleable, and each step is recorded in a per-rule audit table (see *Flexible and fluent custom filter system*).
4. **Clean Coordinates** - `par_clean_coordinates()`: validates coordinates with `CoordinateCleaner::clean_coordin` (Zizka et al. 2019) to flag spatial errors such as centroids, capitals, marine coordinates, and zero coordinates, splitting records into cleaned and problematic tables. Records that pass all tests go to `CoordinateCleaned`; those that fail any test or lack complete coordinates go to `CoordinateProblematic`. Validation is parallelized across user-specified threads.
5. **Detect Native Status** - `detect_native_coord()` and `detect_native_country()`: match each record against WCVP distribution data (the internal `Distributions` dataset) via WGSRPD Level 3 areas to classify it as native, introduced, extinct, location_doubtful, or unknown. `detect_native_coord()` classifies the records with validated coordinates by overlaying

them on the WGSRPD map; `detect_native_country()` classifies the remaining records - including coordinateless ones - through their country codes (see *Precise native-status detection system*).

6. **Map Records** - `map_records()`: renders refined records on interactive maps via `mapView()`, with geohash-based decluttering to reduce visual overlap. Records are colour-coded by native status, the map can cover every species or a single one named by `species`, and multiple basemap layers are supported (OpenStreetMap, Esri World Imagery, and others).
7. **Export Records** - `export_records()`: writes the classified records to disk as gzip-compressed CSV files, one pair of files per classification supplied: the records classified from validated coordinates and those classified through country codes. Each pair holds all classified records and the native subset.

Precise native-status detection system:

`detect_native_coord()` and `detect_native_country()` are the analytical core of VasGBIF: together they assign a native, introduced, extinct, location_doubtful, or unknown classification to every occurrence by matching the record's identification and position against authoritative WCVP distribution data (the internal `Distributions` dataset) organised by WGSRPD Level 3 areas. Classification is split across two functions so that the most precise available evidence always wins: records with validated coordinates are matched spatially by `detect_native_coord()`, and the remaining records (coordinateless ones and those that failed validation) are matched through their country code by `detect_native_country()`.

Both functions accept any table that carries the columns they need, under the names given by their `species`, `longitude`, `latitude`, and `country` arguments; neither depends on a particular upstream object, and both return every input row.

Spatial classification. `detect_native_coord()` overlays records with validated coordinates on the WGSRPD Level 3 polygon map with `terra::extract()` - a single vectorised call that assigns every point its area code in compiled code. Each area code is looked up in a distribution table classified from the WCVP flags (`introduced`, `extinct`, `location_doubtful`) with a fixed priority:

1. `location_doubtful` wins over every other flag;
2. otherwise `introduced`;
3. otherwise `extinct`;
4. otherwise, when no flag is set, the area is `native`;
5. any remaining case is `unknown`.

A record that falls in several areas at once receives the most preferred status (`native` first), never an arbitrary one. Coastal points that fall just outside a polygon are still matched through a geodesic buffer (`buffer_km`, applied in metres by `terra::buffer()` so its meaning is identical at every latitude); buffered hits always rank below exact ones, so a genuine in-polygon match is never displaced by a buffered candidate. The buffer pass is chunked (`buffer_chunk_size`) to keep the relate matrix small. Every record must carry a usable coordinate, so the records that failed validation are left to the country-code pass instead.

Country-code classification. `detect_native_country()` matches the remaining records through `countryCode` mapped to WGSRPD Level 3 areas by the `Level3maping` table. This includes both coordinateless records and records that failed coordinate validation tests. No geometry is used, so this pass is nearly free.

Every classification records how it was obtained in `native_status_source` - `exact` for a direct spatial match and `buffered` for one obtained through the geodesic buffer, `country` for a country-

code match, no_entry when the country mapped but the taxon has no distribution entry there, and unmatched when no usable key exists - so the entire decision chain is auditable.

Precision without a speed penalty. Hybrid markers are normalised so Alnus x pubescens matches the multiplication-sign variant recorded in the distributions; and neither the spatial overlay nor the distribution lookup ever iterates record-by-record in R. Both functions return a nativeDetected table keyed by gbifID that retains every column of the input records and appends LEVEL3_COD, native_status, and native_status_source; the results feed directly into export_records() and map_records() without any join back to the coordinate table.

Flexible and fluent custom filter system:

customized_filter() turns the raw download into an analysis-ready occurrence table. It joins the imported records with the resolved taxonomy and, through extract_issues(), the per-record issue count, then walks a user-selected set of quality rules - one vectorised data.table pass per rule - with every step audited.

Fluent rule control. Each rule is an independently toggleable argument. Three rules are on by default (countryCode, coordinateUncertainty <= 10000 m, gbif_issues_max <= 5), giving sensible out-of-the-box behaviour; date, identifiedBy, and recordedBy are opt-in, so no information is discarded without an explicit choice. Numeric thresholds share one uniform "off" convention - NULL, NA, or '' - so any rule can be disabled without restructuring the call.

Auditable pipeline. Every step - the taxon_resolved join as well as each enabled rule - is logged in the returned summary table (rule, dropped, remaining, failed, only_failed_here), making the effect of each decision visible and reproducible. Because every rule is evaluated on its own terms rather than only on the records left by the preceding rules, failed counts overlap while dropped credits each removed record to the first rule that rejected it. The joins are deliberately strict: the taxonomic join drops unresolved names instead of carrying NA taxonomy forward, and the one-to-one issue join is verified at runtime, stopping if any record lacks an issue count.

Careful collector and identifier detection. The identifiedBy and recordedBy rules remove only values that contain no named person: missing or empty values, a curated multilingual keyword list (unknown, anonymous, desconocido, inconnu, unbekannt, plus Chinese terms for "unknown", "unnamed", "anonymous" and "no details"), and whole-value patterns such as s.n. and n/a. Name separators (,, ;, &, +, and, and their full-width CJK equivalents) protect values that mix a keyword with a real name, and word-boundary matching keeps the CJK keywords from splitting genuine names - deliberately conservative so that real records are never dropped.

The output customFiltered object carries every downstream column in occ_filtered, the excluded records with one verdict column per applied test in occ_marked, and a per-rule summary that exposes the whole filtering decision chain at a glance.

Quick start:

The built-in example dataset lets you run the full pipeline without a GBIF download:

```
library(VasGBIF)


gbif_file <- system.file(
  "extdata", "0003386-260721160103020.zip",
  package = "VasGBIF"
)
```

```
occ_import <- import_records(path = gbif_file)


taxa_checked <- check_taxon(occ_import = occ_import, accuracy = 0.85)


filtered <- customized_filter(
  occ_import = occ_import,
  taxa_checked = taxa_checked
)


refined_coordinates <- par_clean_coordinates(
  filtered$occ_filtered,
  threads = 4
)

native_detected_coord <- detect_native_coord(
  refined_coordinates$CoordinateCleaned
)


native_detected_country <- detect_native_country(
  refined_coordinates$CoordinateProblematic
)

# every species
map_records(
  native_detected_coord = native_detected_coord,
  precision = 3,
  cex = 3
)

# a single species
map_records(
  native_detected_coord = native_detected_coord,
  species = "Saxifraga hirculus"
)

export_records(
  native_detected_coord = native_detected_coord,
  native_detected_country = native_detected_country,
  export_path = getwd()
)
```

Performance:

VasGBIF achieves its speed through several architectural choices:

- **C/C++ backends**: core operations are delegated to data.table, stringi, and terra - packages

written in C/C++ that bypass R's per-iteration interpretive overhead.

- **Vectorisation over explicit loops**: operations such as issue-flag detection in `extract_issues()` and native-status lookups in `detect_native_coord()` and `detect_native_country()` process entire columns in compiled calls rather than iterating in R.
- **SIMD exploitation**: vectorised routines in stringi and `terra::extract()` allow the compiler to emit SIMD instructions (AVX, AVX-512) that process multiple data elements per CPU cycle.
- **Memory-efficient design**: data.table's in-place modification `:=` avoids unnecessary copies, and contiguous memory access patterns improve CPU cache utilisation.
- **Selective parallelisation**: `par_clean_coordinates()` splits the dataset across workers via foreach and doParallel, combining vectorised processing within each chunk with parallel execution across chunks. Chunks are built from whole species rather than from consecutive rows, because the `outliers` test judges each record against the distribution of its own species; keeping every record of a species in one chunk is what makes the parallel result match the serial one.

On a standard laptop, VasGBIF compiles one million occurrence records within 15 minutes.

References

1. Appelhans, Tim, Florian Detsch, Christoph Reudenbach, and Stefan Woellauer. 2023. "Mapview: Interactive Viewing of Spatial Data in R." https://CRAN.R-project.org/package=mapview.
2. Boyle, Brad, Nicole Hopkins, Zhenyuan Lu, Juan Antonio Raygoza Garay, Dmitry Mozzherin, Tony Rees, Naim Matasci, et al. 2013. "The Taxonomic Name Resolution Service: An Online Tool for Automated Standardization of Plant Names." *BMC Bioinformatics* 14 (1): 16. doi:10.1186/147121051416.
3. Chirico, Michael. 2023. "geohashTools: Tools for Working with Geohashes." https://CRAN.R-project.org/package=geohashTools.
4. Govaerts, R. et al. (2021). The World Checklist of Vascular Plants, a continuously updated resource for exploring global plant diversity. *Scientific Data*, 8, 215. doi:10.1038/s41597021-009976
5. Zizka, Alexander, Daniele Silvestro, Tobias Andermann, Josue Azevedo, Camila Duarte Ritter, Daniel Edler, Harith Farooq, et al. 2019. "CoordinateCleaner: Standardized Cleaning of Occurrence Records from Biological Collection Databases." Edited by Tiago Quental. *Methods in Ecology and Evolution* 10 (5): 744-51. doi:10.1111/2041210X.13152.
6. Vilela, Bruno, and Fabricio Villalobos. 2015. "letsR: A New R Package for Data Handling and Analysis in Macroecology." Edited by Timothee Poisot. *Methods in Ecology and Evolution* 6 (10): 1229-34. doi:10.1111/2041210x.12401.

Examples

```
# Three vignettes walk through the entire VasGBIF workflow, from
# data acquisition to final export:
vignette("GetRecords",   package = "VasGBIF")  # search & download GBIF data
vignette("Example",      package = "VasGBIF")  # quick-start with built-in data
vignette("Application",  package = "VasGBIF")  # real-world full-pipeline walk-through
```

WGSRPD3 — *Biodiversity Information Standards (TDWG) World Geographical Scheme for Recording Plant Distributions (WGSRPD)*

Description

Spatial data for WGSRPD Level 3

Usage

```
WGSRPD3
```

Format

An 'sf' object with 369 rows and 5 variables:

LEVEL3_NAM Region name

LEVEL3_COD Region code

LEVEL2_COD Level 2 code

LEVEL1_COD Level 1 code (continent)

geometry sf geometry

Source

- level3

References

Govaerts, R., Nic Lughadha, E., Black, N. et al. The World Checklist of Vascular Plants, a continuously updated resource for exploring global plant diversity. *Sci Data* 8, 215 (2021). doi:10.1038/s41597021009976

WorldLandMap — *A simple features object of the world land map*

Description

An integrated land map

Usage

```
WorldLandMap
```

Format

A simple features object

featurecla Land area

scalerank Scale rank of the land feature

min_zoom Minimum zoom level at which the feature is displayed

geometry Geometry unit

Source

```
rnaturalearth::ne_download(scale = 110,type = 'land',category = 'physical',returnclass
= "sf")
```

See Also

`rnaturalearth::ne_download()`

Index

∗ **datasets**
- Distributions, 13
- EnumOccurrenceIssue, 14
- Level3maping, 18
- WGSRPD3, 34
- WorldLandMap, 34

∗ **package**
- VasGBIF, 29

base::tempfile(), *17*

check_taxon, 2
check_taxon(), *5–8*, *27*, *29*
CoordinateCleaner, *29*
CoordinateCleaner::clean_coordinates, *22*
CoordinateCleaner::clean_coordinates(), *24*, *29*
customized_filter, 5
customized_filter(), *22–24*, *26*, *29*, *31*

data.table, *31–33*
data.table::as.data.table(), *10*, *12*
data.table::fread(), *17, 18*
data.table::rbindlist(), *3*
detect_native_coord, 9
detect_native_coord(), *11*, *13*, *15, 16*, *20–22*, *24*, *27*, *29, 30*, *33*
detect_native_country, 11
detect_native_country(), *9–11*, *15, 16*, *21, 22*, *24*, *27*, *29, 30*, *33*
Distributions, 13, *19, 20*
doParallel, *33*

EnumOccurrenceIssue, 14
export_records, 15
export_records(), *24*, *30, 31*
extract_issues(), *6*, *8*, *31*, *33*

foreach, *33*

gh_encode(), *21*

head(), *25–27*

import_records, 16
import_records(), *3–5*, *8*, *26*, *29*

Level3maping, 18

map_records, 20
map_records(), *30, 31*
mapView(), *21*, *30*

par_clean_coordinates, 22
par_clean_coordinates(), *10*, *12*, *16*, *25*, *28, 29*, *33*
parallel::detectCores(), *28*
print(), *8*
print.CoordinateRefined, 25
print.CoordinateRefined(), *24*
print.customFiltered, 25
print.import, 26
print.import(), *18*
print.nativeDetected, 26
print.nativeDetected(), *11*, *13*
print.occ_taxa, 27
print.occ_taxa(), *4*

rnaturalearth::ne_download(), *35*

set_threads, 28
set_threads(), *23, 24*
stringi, *32, 33*

terra, *32*
terra::buffer(), *10*, *30*
terra::extract(), *9*, *30*, *33*
TNRS, *29*
TNRS::TNRS(), *3, 4*

unzip(), *18*

VasGBIF, 29

WGSRPD3, *19*, *20*, 34
WorldLandMap, 34