Package: wddsWizard (via r-universe)

June 3, 2026

Title Data Wizard for a Minimal Wildlife Disease Data Standard

Version 0.2.5

Description Facilitates compliance with and use of the Wildlife Disease Data Standard stored on Zenodo <https://doi.org/10.5281/zenodo.15020049>. It allows users to restructure and validate datasets.

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Imports assertthat, cli, curl, dplyr, fs, here, jsonlite, purrr, R6, readr, rlang, snakecase, stringr, tidyr

Suggests knitr, rmarkdown, janitor, kableExtra, readxl, tibble, jsonvalidate, testthat (>= 3.0.0), taxize, deposits (>= 0.2.1.63)

Config/testthat/edition 3

VignetteBuilder knitr

Depends R (>= 4.1.0)

LazyData true

License MIT + file LICENSE

URL https://viralemergence.github.io/wddsWizard/

BugReports https://github.com/viralemergence/wddsWizard/issues

Remotes ropenscilabs/deposits

Config/pak/sysreqs cmake make libicu-dev libuv1-dev libssl-dev libx11-dev

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

Date/Publication 2025-12-05 16:44:26 UTC

RemoteUrl https://github.com/viralemergence/wddsWizard

RemoteRef HEAD

RemoteSha 46e597a2ea9355d291afc73e35ceff1d492995b5

Contents

`batch_download_deposit_versions` *Batch download deposit versions*

Description

This is `download_deposit_version` wrapped in a `purr::pmap` call.

Usage

```
batch_download_deposit_versions(df = list_deposit_versions(), dir_path)
```

Arguments

`df`	Data frame. Has the same structure as the output of `list_deposit_versions()`. Default is `list_deposit_versions()` so that it downloads all versions of the deposit.
`dir_path`	Character. Path to folder where files should be downloaded.

Value

List of download locations.

See Also

Other WDDS deposit: download_deposit_version(), list_deposit_versions(), sanitize_version(), set_wdds_version(), wdds_data_templates(), wdds_example_data(), wdds_json()

Examples

```
## Not run:
# download all versions
batch_download_deposit_versions(dir_path = "data")

## End(Not run)
```

---

becker_disease_data *Becker et al. dataset*

Description

A bat coronavirus dataset that conforms to the wildlife disease data standard. See data standard for field descriptions

Usage

```
becker_disease_data
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 2 rows and 21 columns.

Source

https://pharos.viralemergence.org/projects/?prj=prjRPayEvMecN

See Also

Other Data: becker_project_metadata, disease_data_required_fields, minimal_disease_data, minimal_project_metadata, project_metadata_required_fields, schema_required_fields, spdx_licenses, wdds_to_dcmi_map, wdds_to_pharos_map

becker_project_metadata *Becker et al. project metadata*

Description

The project metadata for a bat coronavirus dataset that conforms to the wildlife disease data standard. See data standard for field descriptions.

Usage

```
becker_project_metadata
```

Format

An object of class `list` of length 11.

Source

https://www.ebi.ac.uk/pride/archive/projects/PXD031075

See Also

Other Data: `becker_disease_data`, `disease_data_required_fields`, `minimal_disease_data`, `minimal_project_metadata`, `project_metadata_required_fields`, `schema_required_fields`, `spdx_licenses`, `wdds_to_dcmi_map`, `wdds_to_pharos_map`

clean_field_names *Clean Field Names*

Description

Clean Field Names

Usage

```
clean_field_names(x)
```

Arguments

x	Data frame or other named object

Value

object with names in `snakecase::to_lower_camel_case` format

See Also

Other JSON Prep: get_entity(), prep_affiliation(), prep_array(), prep_array_objects(), prep_atomic(), prep_creators(), prep_data(), prep_descriptions(), prep_for_json(), prep_from_metadata_template(), prep_fundingReferences(), prep_identifier(), prep_language(), prep_methodology(), prep_methods(), prep_nameIdentifiers(), prep_object(), prep_publicationYear(), prep_relatedIdentifiers(), prep_rights(), prep_subjects(), prep_titles()

Examples

```
df <- data.frame("Sample ID" = 1:10, "Name" = "Fred", "Host Identification" = "Pinus strobus")

clean_field_names(df)
```

---

create_object_docs *Create Docs Section for a schema object*

Description

Create Docs Section for a schema object

Usage

```
create_object_docs(x, idx, required_fields, schema_dir)
```

Arguments

x	List. Schema property or definition
idx	Name from schema property
required_fields	Character. Vector of required fields
schema_dir	Character. directory where the schema is stored

Value

Character formatted markdown text

See Also

Other Schema Docs: create_schema_docs(), get_ref(), get_required_fields(), increase_docs_depth(), paste_reduce(), paste_reduce_ul()

create_schema_docs *Create Documentation for a schema*

Description

Produces nested markdown that documents a schema. This is a recursive set of function

Usage

```
create_schema_docs(schema_path = the$current_schema_path, sep = "\n")
```

Arguments

schema_path	Character. Path to a json-schema. Default is the current schema path set in the package environment the.
sep	Character. separator to be used by paste_reduce(). Default is "\n" to create line breaks.

Value

character vector of markdown text

See Also

Other Schema Docs: create_object_docs(), get_ref(), get_required_fields(), increase_docs_depth(), paste_reduce(), paste_reduce_ul()

Examples

```
## Not run:
create_schema_docs()

## End(Not run)
```

datacite_schema *Datacite Data Stadnard*

Description

See data standard JSON file for field descriptions.

Usage

```
datacite_schema
```

Format

An object of class `list` of length 9.

See Also

Other Schema: `disease_data_schema`, `project_metadata_schema`, `schema_obj`, `schema_properties`, `schema_terms`, `wdds_schema`

`disease_data_required_fields` *Required fields in the disease data object*

Description

See data standard JSON file for field descriptions.

Usage

```
disease_data_required_fields
```

Format

An object of class `character` of length 9.

See Also

Other Data: `becker_disease_data`, `becker_project_metadata`, `minimal_disease_data`, `minimal_project_metadata`, `project_metadata_required_fields`, `schema_required_fields`, `spdx_licenses`, `wdds_to_dcmi_map`, `wdds_to_pharos_map`

`disease_data_schema` *Wildlife Disease Data Standard - data*

Description

See data standard JSON file for field descriptions.

Usage

```
disease_data_schema
```

Format

An object of class `list` of length 7.

See Also

Other Schema: datacite_schema, project_metadata_schema, schema_obj, schema_properties, schema_terms, wdds_schema

---

download_deposit_version *Download deposit version*

Description

Downloads and extracts some version of the deposit. This function is specific to the structure of the wdds repo.

Usage

```
download_deposit_version(zenodo_id, version, latest_version, dir_path)
```

Arguments

zenodo_id	String. ID for a Zenodo deposit. Should correspond to the version of a deposit.
version	String. Version number/id for the deposit (e.g. v.1.1.1).
latest_version	Logical. Indicates that the work is designated as the latest version.
dir_path	String. Path to directory where the files should be downloaded e.g. "inst/extdata/wdds_archive" note no trailing slash on the path.

Value

String. Path to downloaded version.

See Also

Other WDDS deposit: batch_download_deposit_versions(), list_deposit_versions(), sanitize_version(), set_wdds_version(), wdds_data_templates(), wdds_example_data(), wdds_json()

Examples

```
# list all deposit versions
list_deposit_versions()

# download the deposit

## Not run:
download_deposit_version("15270582", "v.1.0.3", TRUE, "data")

## End(Not run)
```

download_oa_item *Rate limited download of OA items*

Description

Checks if file exists in a directory, downloads the file if its not found. Sleeps for a given amount of time to respect rate limits on openalex servers.

Usage

```
download_oa_item(entity, oa_id, dir_temp = tempdir(), sleep_time = 1)
```

Arguments

entity	Character. What kind of openalex item is it?
oa_id	Character. ID from openalex
dir_temp	Character. path to directory where jons is stored.
sleep_time	Numeric. Seconds of sleep.

Value

Character. File path to json file

See Also

Other Project Metadata: expand_tidy_dfs(), extract_metadata_from_doi(), extract_metadata_oa(), generate_metadata_csv(), generate_repeat_dfs(), make_simple_df()

expand_tidy_dfs *Expand tidy dataframes to project metadata template format*

Description

Creates a JSON-like structure in the csv that can be processed using established workflows in this package.

Usage

```
expand_tidy_dfs(tidy_df, group_prefix)
```

Arguments

tidy_df	data frame. Each row corresponds to a complete entry.
group_prefix	character. A repeatable metadata property in the project metadata section of WDDS. See https://viralemergence.github.io/wddsWizard/articles/schema_overview.html#project_metad

Value

Data frame. The data frame contains the fields Group, Variable, and Value.

See Also

Other Project Metadata: download_oa_item(), extract_metadata_from_doi(), extract_metadata_oa(), generate_metadata_csv(), generate_repeat_dfs(), make_simple_df()

Examples

```
# a nice tidy dataset
creators_tidy <- data.frame("Name" = paste(letters[1:10],LETTERS[1:10]),
         "Given Name" = letters[1:10],
         "Family Name" = LETTERS[1:10],
         "Name Identifier" = sample(1:100,10,FALSE),
         "Affiliation" = letters[11:20],
         "Affiliation Identifier" = 11:20,
         check.names =FALSE)

# an expanded dataset that matches the template format.
creators_tidy |>
 expand_tidy_dfs(group_prefix = "Creators")
```

---

extract_metadata_from_doi

Extract Project Metadata from DOI

Description

Some works are explicitly connected to a publication and the metadata for that publication are fairly complete. Instead of re-writing the metadata, it would be better to extract it and transform it.

Usage

```
extract_metadata_from_doi(doi, file_path, write_output = TRUE)
```

Arguments

doi	String. DOI for a published work
file_path	String. Where should the output be written?
write_output	Logical. Should the output be written to a file?

Value

data frame. A data frame structured in the same way as the metadata template csv.

See Also

Other Project Metadata: download_oa_item(), expand_tidy_dfs(), extract_metadata_oa(), generate_metadata_csv(), generate_repeat_dfs(), make_simple_df()

Examples

```
doi <-"doi.org/10.1038/s41597-025-05332-x"
extract_metadata_from_doi(doi = doi,write_output=FALSE)
```

---

extract_metadata_oa *Extract Metadata from Open Alex record*

Description

Uses the DOI for a work to extract metadata from OpenAlex - https://openalex.org/. The OpenAlex data model does not included some fields that are part of the wdds project metadata related identifiers.

Usage

```
extract_metadata_oa(doi)
```

Arguments

doi	Character. A digital object identifier for a published work.

Details

Carefully review and edit the metadata produced.

We recommend writing the metadata to a csv, editing the csv, then processing it as demonstrated in the project metadata tutorial.

Value

data frame. A data frame structured in the same way as the metadata template CSV.

See Also

Other Project Metadata: download_oa_item(), expand_tidy_dfs(), extract_metadata_from_doi(), generate_metadata_csv(), generate_repeat_dfs(), make_simple_df()

Examples

```
doi <- "doi.org/10.1038/s41597-025-05332-x"
extract_metadata_oa(doi = doi)
```

`generate_metadata_csv` *Generate minimal project metadata template*

Description

This function allows you to generate a minimal metadata template for your project. You provide certain values and it generates a csv based on those values. Any parameter that starts with num takes an integer and creates repeat entries in the metadata csv. All other values take a string or logical input and will prepopulate that section of the metadata csv.

Usage

```
generate_metadata_csv(
  file_path,
  event_based,
  archival,
  num_creators,
  num_titles,
  identifier,
  identifier_type,
  num_subjects,
  publication_year,
  rights,
  language,
  num_descriptions,
  num_fundingReferences,
  num_related_identifiers,
  write_output = TRUE
)
```

Arguments

`file_path`	String. Where should the CSV file be saved?
`event_based`	Logical. Whether or not research was conducted in response to a known or suspected infectious disease outbreak, observed animal morbidity or mortality, etc.
`archival`	Logical. Whether samples were from an archival source (e.g., museum collections, biobanks).
`num_creators`	Integer. Number of creators for a work.
`num_titles`	Integer. Number of titles for a work.
`identifier`	String. A unique string that identifies a resource. Should be a DOI
`identifier_type`	String. Should be DOI
`num_subjects`	Integer. Number of subjects. Subject, keyword, classification code, or key phrase describing the resource

publication_year
: String. Year when work was published

rights
: String. Use one of the rights identifiers found here https://spdx.org/licenses/

language
: String. The primary language of the resource.

num_descriptions
: Integer. Number of descriptions to add to the csv. All additional information that does not fit in any of the other categories. May be used for technical information or detailed information associated with a scientific instrument

num_fundingReferences
: Integer. Number of funders to add to the csv. Name and other identifying information of a funding provider

num_related_identifiers
: Integer. Number of other works you would like to link to.

write_output
: Logical. Should the file be written?

Value

data.frame

See Also

Other Project Metadata: download_oa_item(), expand_tidy_dfs(), extract_metadata_from_doi(), extract_metadata_oa(), generate_repeat_dfs(), make_simple_df()

Examples

```
generate_metadata_csv(file_path = "test.csv",
event_based = TRUE,
archival = FALSE,
num_creators = 10,
num_titles = 1,
identifier = "https://doi.org/10.1080/example.doi",
identifier_type = "doi",
num_subjects = 5,
publication_year = "2025",
rights = "cc-by",
language = "en",
num_descriptions = 1,
num_fundingReferences = 4,
num_related_identifiers= 5,
write_output = FALSE) # change to TRUE to write the csv
```

generate_repeat_dfs *generate_repeat_dfs*

Description

generate_repeat_dfs

Usage

```
generate_repeat_dfs(num_groups, group_prefix, group_variables)
```

Arguments

num_groups	Numeric. Number of groups
group_prefix	Character. A group name
group_variables	Character. A comma separated scalar string of variables.

Value

data frame. Structured appropriately for the metadata csv.

See Also

Other Project Metadata: download_oa_item(), expand_tidy_dfs(), extract_metadata_from_doi(), extract_metadata_oa(), generate_metadata_csv(), make_simple_df()

Examples

```
related_ids_df <- generate_repeat_dfs(num_groups = 5,
group_prefix = "Related Identifiers",
group_variables = "Related Identifier,Related Identifier Type,Relation Type")
```

get_entity *Get entity*

Description

The get_entity function creates standard entities that will be easier to transform json.

Usage

```
get_entity(x)
```

Arguments

x	data frame. A "long" form data frame with the fields Group, entity_id, Value, and variable.

Details

Pivots data from long to wide and formats column names.

Value

data frame in "wide" form

See Also

Other JSON Prep: clean_field_names(), prep_affiliation(), prep_array(), prep_array_objects(), prep_atomic(), prep_creators(), prep_data(), prep_descriptions(), prep_for_json(), prep_from_metadata_template(), prep_fundingReferences(), prep_identifier(), prep_language(), prep_methodology(), prep_methods(), prep_nameIdentifiers(), prep_object(), prep_publicationYear(), prep_relatedIdentifiers(), prep_rights(), prep_subjects(), prep_titles()

Examples

```
df <- data.frame(Group = 1, entity_id = 1, Value = 1:3, Variable = letters[1:3])

get_entity(df)
```

---

get_ref *Get schema references*

Description

Parses $ref calls in a schema. Can retrieve internal ('"$ref":"#/definitions/someDef") or external references ('"$ref":"schemas/datacite/datacite.json"').

Usage

```
get_ref(x, schema_dir)
```

Arguments

x	List. Must have property "$ref"
schema_dir	Character. Directory for the current schema.

Details

For external references, it can handle both pointers and references to entire schemas. This function navigates between parent and child schemas by manipulating variables in the package environment the.

Value

List or Character. Character is only returned if an entire schema is referenced.

See Also

Other Schema Docs: create_object_docs(), create_schema_docs(), get_required_fields(), increase_docs_depth(), paste_reduce(), paste_reduce_ul()

---

get_required_fields *Get the required fields*

Description

Gets the required fields for an object or schema

Usage

```
get_required_fields(schema_list)
```

Arguments

schema_list	List from jsonlite::read_json

Value

character vector of required fields

See Also

Other Schema Docs: create_object_docs(), create_schema_docs(), get_ref(), increase_docs_depth(), paste_reduce(), paste_reduce_ul()

Examples

```
schema_list <- jsonlite::read_json(wdds_json("latest", "schemas/disease_data.json"))
get_required_fields(schema_list)
```

increase_docs_depth *Increase documentation depth*

Description

Pads the left side of any list items with an extra 4 spaces

Usage

```
increase_docs_depth(string)
```

Arguments

string	Character. item to be parsed

Value

character

See Also

Other Schema Docs: create_object_docs(), create_schema_docs(), get_ref(), get_required_fields(), paste_reduce(), paste_reduce_ul()

list_deposit_versions *List Versions of a deposit on Zenodo*

Description

This function list all the versions of a deposit associated with a parent id. The parent id is used to identify a set of works that are different versions of the same work. The parent id is provided from the Zenodo API. If you download a JSON representation of the deposit (export to json), there will be an attribute in that json called parent that looks like "https://zenodo.org/api/records/15020049". The 8 digit string at the end of the url is the parent id.

Usage

```
list_deposit_versions(parent_id = "15020049")
```

Arguments

parent_id	String. Identifier for a Zenodo deposit with multiple versions. Default is the parent id for the wdds zenodo deposit.

Value

Data frame. The data frame contains the Zenodo id for each version of the deposit, as well as the version name, and logical field called latest that indicates if this is the latest version.

See Also

Other WDDS deposit: `batch_download_deposit_versions()`, `download_deposit_version()`, `sanitize_version()`, `set_wdds_version()`, `wdds_data_templates()`, `wdds_example_data()`, `wdds_json()`

Examples

```
list_deposit_versions()
```

---

list_wdds_templates *File paths for wdds templates*

Description

Displays file paths for Wildlife Disease Data Standard templates

Usage

```
list_wdds_templates(template_file = NULL)
```

Arguments

`template_file`	character. file name for a template. Default is `NULL` to return template files

Details

If path is null, displays all files in the templates folder.

Value

file paths or, if path = NULL, a list of file names

See Also

Other Templates: `use_wdds_template()`

make_simple_df *A convenience function for making non-repeating items*

Description

A convenience function for making non-repeating items

Usage

```
make_simple_df(property, value)
```

Arguments

property	string. Metadata group and variable name
value	A value for that property.

Value

data frame. A data frame that conforms to non-repeatable structure in template.

See Also

Other Project Metadata: download_oa_item(), expand_tidy_dfs(), extract_metadata_from_doi(), extract_metadata_oa(), generate_metadata_csv(), generate_repeat_dfs()

Examples

```
language_df <- make_simple_df(property = "language", value = "fr")
```

minimal_disease_data *An example of minimal disease data*

Description

This is a minimal disease data example. It is a data frame with the minimal items required for disease data.

Usage

```
minimal_disease_data
```

Format

An object of class data.frame with 3 rows and 15 columns.

See Also

Other Data: becker_disease_data, becker_project_metadata, disease_data_required_fields, minimal_project_metadata, project_metadata_required_fields, schema_required_fields, spdx_licenses, wdds_to_dcmi_map, wdds_to_pharos_map

---

`minimal_project_metadata` *An example of minimal project metadata*

Description

This is a minimal project metadata example. It is a list with the minimal items required for project metadata.

Usage

```
minimal_project_metadata
```

Format

An object of class `list` of length 7.

See Also

Other Data: becker_disease_data, becker_project_metadata, disease_data_required_fields, minimal_disease_data, project_metadata_required_fields, schema_required_fields, spdx_licenses, wdds_to_dcmi_map, wdds_to_pharos_map

---

`na_to_blank` *Convert NA's to blanks*

Description

Converts all columns to character then converts all NA's to blanks.

Usage

```
na_to_blank(df)
```

Arguments

`df`	data frame. A data frame where NAs should be coverted to blanks. Cannot be a tibble with nested columns.

Value

data frame. All columns will be character and all NA's will be replaced with "".

See Also

Other Standards Mapping: `translate_to_dcmi()`, `wdds_to_dcmi()`, `wdds_to_pharos()`

Examples

```
data.frame(a = 1:10, b = c(1:9,NA)) |>
  na_to_blank()
```

`paste_reduce` *Paste Reduce*

Description

A paste function that can be used with `purrr::reduce` to build up nested documentation items

Usage

```
paste_reduce(x, y, sep = "\n")
```

Arguments

`x`	Character
`y`	Character
`sep`	Character. Default is a line break `"\n"`

Value

Character

See Also

Other Schema Docs: `create_object_docs()`, `create_schema_docs()`, `get_ref()`, `get_required_fields()`, `increase_docs_depth()`, `paste_reduce_ul()`

Examples

```
text_a <- "hello"
text_b <- "world"
paste_reduce(text_a, text_b)
```

paste_reduce_ul *Paste Reduce unordered list item*

Description

A paste function that can be used with `purrr::reduce` to build up nested documentation items

Usage

```
paste_reduce_ul(x, y, sep = "\n - ")
```

Arguments

x	Character
y	Character
sep	Character. Default is a line break followed by a dash `"\n - "` to create an unordered list in markdown.

Value

Character

See Also

Other Schema Docs: `create_object_docs()`, `create_schema_docs()`, `get_ref()`, `get_required_fields()`, `increase_docs_depth()`, `paste_reduce()`

Examples

```
text_a <- "hello"
text_b <- "world"
paste_reduce_ul(text_a, text_b)
```

prep_affiliation *prep affiliation*

Description

There are affiliations associated with a creator.

Usage

```
prep_affiliation(x)
```

Arguments

x	Data frame from prep_creators

Details

Affiliation in datacite is an array of objects with properties name, affiliationIdentifier, affiliationIdentifierScheme, and schemeURI. This function takes the affiliation fields and restructures as a list within the dataframe.

Affiliation fields to be converted to a list: "affiliation", #' "affiliationIdentifier", "affiliationIdentifierScheme" , "schemeUri"

Value

Data frame with affiliation fields in a list column called `affiliation`

See Also

Other JSON Prep: `clean_field_names()`, `get_entity()`, `prep_array()`, `prep_array_objects()`, `prep_atomic()`, `prep_creators()`, `prep_data()`, `prep_descriptions()`, `prep_for_json()`, `prep_from_metadata_template()`, `prep_fundingReferences()`, `prep_identifier()`, `prep_language()`, `prep_methodology()`, `prep_methods()`, `prep_nameIdentifiers()`, `prep_object()`, `prep_publicationYear()`, `prep_relatedIdentifiers()`, `prep_rights()`, `prep_subjects()`, `prep_titles()`

Examples

```
creator_df <- wddsWizard::becker_project_metadata$creators[[1]]
creator_df_aff_prepped <- prep_affiliation(creator_df)
```

---

prep_array *Prep array*

Description

Prep array

Usage

```
prep_array(x)
```

Arguments

x	a list object.

Value

unnamed vector

See Also

Other JSON Prep: clean_field_names(), get_entity(), prep_affiliation(), prep_array_objects(), prep_atomic(), prep_creators(), prep_data(), prep_descriptions(), prep_for_json(), prep_from_metadata_template(), prep_fundingReferences(), prep_identifier(), prep_language(), prep_methodology(), prep_methods(), prep_nameIdentifiers(), prep_object(), prep_publicationYear(), prep_relatedIdentifiers(), prep_rights(), prep_subjects(), prep_titles()

Examples

```
# this form can arise because of the csv template
nested_list <- list(list("formats" = list("formats" = "csv",
"formats" = "fasta")))

prep_array(nested_list)
```

---

prep_array_objects *Prepare an array of objects*

Description

wraps a data frame in a list and or unboxes list items that are 1 row dataframes. This will result in an array of objects being created.

Usage

```
prep_array_objects(x, unbox = TRUE)
```

Arguments

x	list of data frames or a data frame
unbox	logical. Should the things be unboxed?

Value

list of single row unboxed data frames

See Also

Other JSON Prep: clean_field_names(), get_entity(), prep_affiliation(), prep_array(), prep_atomic(), prep_creators(), prep_data(), prep_descriptions(), prep_for_json(), prep_from_metadata_template(), prep_fundingReferences(), prep_identifier(), prep_language(), prep_methodology(), prep_methods(), prep_nameIdentifiers(), prep_object(), prep_publicationYear(), prep_relatedIdentifiers(), prep_rights(), prep_subjects(), prep_titles()

Examples

```
# note that you cannot unbox data frames with more than 1 row

x <- list(
  tibble::tibble(age = 1, group = letters[1]),
  tibble::tibble(age = 2, group = letters[2])
)

# running jsonlite::toJSON on an unmodified object results in
# extra square brackets - an array of arrays of objects
jsonlite::toJSON(x, pretty = TRUE)

# with the prepped data we get an array of objects
x_prepped <- prep_array_objects(x)

x_prepped |>
  jsonlite::toJSON(pretty = TRUE)
```

`prep_atomic` *Prepare atomic*

Description

This is a thin wrapper for `jsonlite::unbox`. It stops `jsonlite` from representing single character, numeric, logical, etc. items as arrays.

Usage

```
prep_atomic(x, unbox = TRUE)
```

Arguments

`x`	vector or single row data frame
`unbox`	Logical. Should the value be unboxed? See `jsonlite::unbox`

Details

This is useful when a property or definition is of type string, number, logical and of length 1.

Value

an unboxed dataframe with 1 row

See Also

Other JSON Prep: clean_field_names(), get_entity(), prep_affiliation(), prep_array(), prep_array_objects(), prep_creators(), prep_data(), prep_descriptions(), prep_for_json(), prep_from_metadata_template(), prep_fundingReferences(), prep_identifier(), prep_language(), prep_methodology(), prep_methods(), prep_nameIdentifiers(), prep_object(), prep_publicationYear(), prep_relatedIdentifiers(), prep_rights(), prep_subjects(), prep_titles()

Examples

```
x <- 1

# values in x are stored in an array
x |>
  jsonlite::toJSON()
# output is [1]

# values in x are NOT stored in an array (no square brackets)
prep_atomic(x) |>
  jsonlite::toJSON()
# output is 1
```

---

prep_creators *Prepare creators*

Description

The creator object can be complex so we prepare components of the final object (e.g. affiliation, nameIdentifiers) then run prep_array_objects

Usage

```
prep_creators(x)
```

Arguments

x	data frame or named list.

Value

List of unboxed data frames

See Also

Other JSON Prep: clean_field_names(), get_entity(), prep_affiliation(), prep_array(), prep_array_objects(), prep_atomic(), prep_data(), prep_descriptions(), prep_for_json(), prep_from_metadata_template(), prep_fundingReferences(), prep_identifier(), prep_language(), prep_methodology(), prep_methods(), prep_nameIdentifiers(), prep_object(), prep_publicationYear(), prep_relatedIdentifiers(), prep_rights(), prep_subjects(), prep_titles()

Examples

```
wddsWizard::becker_project_metadata$creators |>
  prep_creators()
```

`prep_data` *Prepare Data*

Description

Prepares an object of arrays.

Usage

```
prep_data(x)
```

Arguments

x	named vector, list, or data frame

Details

Note that unboxing will only work on items where you have 1:1 key value pair. So if you have a dataframe with multiple rows or a list with multiple values at a given position, it won't work.

Value

List of formatted objects

See Also

Other JSON Prep: `clean_field_names()`, `get_entity()`, `prep_affiliation()`, `prep_array()`, `prep_array_objects()`, `prep_atomic()`, `prep_creators()`, `prep_descriptions()`, `prep_for_json()`, `prep_from_metadata_template()`, `prep_fundingReferences()`, `prep_identifier()`, `prep_language()`, `prep_methodology()`, `prep_methods()`, `prep_nameIdentifiers()`, `prep_object()`, `prep_publicationYear()`, `prep_relatedIdentifiers()`, `prep_rights()`, `prep_subjects()`, `prep_titles()`

Examples

```
cars_small <- datasets::cars[1:10, ]

# creates an array of objects where each
# row is an object
cars_small |>
  jsonlite::toJSON(pretty = TRUE)

# creates an object with 2 arrays
prep_object(cars_small) |>
  jsonlite::toJSON(pretty = TRUE)
```

```
# this makes no difference
x <- list("hello" = 1:10, "world" = "Earth")

prep_object(x) |>
  jsonlite::toJSON(pretty = TRUE)
```

`prep_descriptions` *Prepare descriptions*

Description

Wrapper for `prep_array_objects`.

Usage

```
prep_descriptions(x)
```

Arguments

x	Data frame/Tibble containing description items

Value

List with x marked as unbox (do not make an array)

See Also

Other JSON Prep: `clean_field_names()`, `get_entity()`, `prep_affiliation()`, `prep_array()`, `prep_array_objects()`, `prep_atomic()`, `prep_creators()`, `prep_data()`, `prep_for_json()`, `prep_from_metadata_template()`, `prep_fundingReferences()`, `prep_identifier()`, `prep_language()`, `prep_methodology()`, `prep_methods()`, `prep_nameIdentifiers()`, `prep_object()`, `prep_publicationYear()`, `prep_relatedIdentifiers()`, `prep_rights()`, `prep_subjects()`, `prep_titles()`

Examples

```
x <- wddsWizard::becker_project_metadata$descriptions

prep_descriptions(x) |> jsonlite::toJSON()
```

prep_for_json *Prepare data for json*

Description

Usespurrr::modify_at to apply a set of methods at specific locations in a list.

Usage

```
prep_for_json(x, prep_methods_list = prep_methods())
```

Arguments

x	list. Named list of data frames, lists, or vectors. For methods to be applied, the names of the list items should match the names in the methods list
prep_methods_list	list. Named list of methods where each items is a function to applied to corresponding items in x. Default is full list of methods from prep_methods().

Value

Named list where methods have been applied.

See Also

Other JSON Prep: clean_field_names(), get_entity(), prep_affiliation(), prep_array(), prep_array_objects(), prep_atomic(), prep_creators(), prep_data(), prep_descriptions(), prep_from_metadata_template(), prep_fundingReferences(), prep_identifier(), prep_language(), prep_methodology(), prep_methods(), prep_nameIdentifiers(), prep_object(), prep_publicationYear(), prep_relatedIdentifiers(), prep_rights(), prep_subjects(), prep_titles()

Examples

```
wddsWizard::becker_project_metadata |>
  prep_for_json()

a <- list("hello_world" = 1:10)
methods_list <- list(
  "hello_world" = function(x) {
    x * 2
  },
  "unused_method" = function(x) {
    x / 2
  }
)
prep_for_json(a, methods_list)
```

prep_from_metadata_template *Prepare metadata created from the metadata template for conversion to JSON*

Description

A convenience function for those who used the metadata template to create their project metadata data.

Usage

```
prep_from_metadata_template(
  project_metadata,
  prep_methods_list = prep_methods(),
  schema_properties = wddsWizard::schema_properties,
  json_prep = TRUE
)
```

Arguments

project_metadata	Data frame. Should correspond to the structure of the project_metadata_template.csv
prep_methods_list	list. Named list of methods where each items is a function to applied to corresponding items in x.Default is prep_methods().
schema_properties	Data frame. A data frame of schema properties and their types.
json_prep	Logical. Should the metadata be prepped for JSON?

Details

Does some light data formatting to make conversion to json easier.

Value

Named list ready to be converted to json

See Also

Other JSON Prep: clean_field_names(), get_entity(), prep_affiliation(), prep_array(), prep_array_objects(), prep_atomic(), prep_creators(), prep_data(), prep_descriptions(), prep_for_json(), prep_fundingReferences(), prep_identifier(), prep_language(), prep_methodology(), prep_methods(), prep_nameIdentifiers(), prep_object(), prep_publicationYear(), prep_relatedIdentifiers(), prep_rights(), prep_subjects(), prep_titles()

Examples

```
## Not run:
# create
wddsWizard::use_template("project_metadata_template.csv",
  folder = "data",
  file_name = "my_project_metadata.csv"
)
project_metadata <- read.csv("data/my_project_metadata.csv")

prepped_project_metadata <- wddsWizard::prep_from_metadata_template(project_metadata)

project_metadat_json <- jsonlite::toJSON(prepped_project_metadata, pretty = TRUE)

## End(Not run)
```

---

prep_fundingReferences

Prepare funding references

Description

creates an array of objects

Usage

```
prep_fundingReferences(x)
```

Arguments

x	list of tibbles/data frames or a tibble/data frame

Value

list of single row unboxed data frames

See Also

Other JSON Prep: clean_field_names(), get_entity(), prep_affiliation(), prep_array(), prep_array_objects(), prep_atomic(), prep_creators(), prep_data(), prep_descriptions(), prep_for_json(), prep_from_metadata_template(), prep_identifier(), prep_language(), prep_methodology(), prep_methods(), prep_nameIdentifiers(), prep_object(), prep_publicationYear(), prep_relatedIdentifiers(), prep_rights(), prep_subjects(), prep_titles()

Examples

```
wddsWizard::becker_project_metadata$fundingReferences |>
  prep_fundingReferences()
```

prep_identifier *Prep identifier*

Description

Prepare identifier for a scholarly work. Wrapper for prep_array_objects

Usage

```
prep_identifier(x)
```

Arguments

x	data frame with identifier properties

Value

List with x marked as do not unbox

See Also

Other JSON Prep: clean_field_names(), get_entity(), prep_affiliation(), prep_array(), prep_array_objects(), prep_atomic(), prep_creators(), prep_data(), prep_descriptions(), prep_for_json(), prep_from_metadata_template(), prep_fundingReferences(), prep_language(), prep_methodology(), prep_methods(), prep_nameIdentifiers(), prep_object(), prep_publicationYear(), prep_relatedIdentifiers(), prep_rights(), prep_subjects(), prep_titles()

Examples

```
wddsWizard::becker_project_metadata$identifier |> prep_identifier()
```

prep_language *Prep language*

Description

Prepare the language property - this should describe the language of the scholarly work.

Usage

```
prep_language(x)
```

Arguments

x	named list, vector, or data.frame of with 1:1 name:value pairs

Value

an unboxed dataframe with 1 row

See Also

Other JSON Prep: clean_field_names(), get_entity(), prep_affiliation(), prep_array(), prep_array_objects(), prep_atomic(), prep_creators(), prep_data(), prep_descriptions(), prep_for_json(), prep_from_metadata_template(), prep_fundingReferences(), prep_identifier(), prep_methodology(), prep_methods(), prep_nameIdentifiers(), prep_object(), prep_publicationYear(), prep_relatedIdentifiers(), prep_rights(), prep_subjects(), prep_titles()

Examples

```
a <- data.frame("language" = "en")

prep_language(a)
```

---

prep_methodology *Prep methodology for conversion to json*

Description

Prep methodology for conversion to json

Usage

```
prep_methodology(x)
```

Arguments

x	List. methodology component of a list

Value

properly formatted list

See Also

Other JSON Prep: clean_field_names(), get_entity(), prep_affiliation(), prep_array(), prep_array_objects(), prep_atomic(), prep_creators(), prep_data(), prep_descriptions(), prep_for_json(), prep_from_metadata_template(), prep_fundingReferences(), prep_identifier(), prep_language(), prep_methods(), prep_nameIdentifiers(), prep_object(), prep_publicationYear(), prep_relatedIdentifiers(), prep_rights(), prep_subjects(), prep_titles()

Examples

```
## Not run:
prepped_list <- project_metadata_list_entities
prepped_list$methodology <- prep_methodology(project_metadata_list_entities$methodology)

OR

prepped_list <- purrr::modify_at(project_metadata_list_entities, "methodology", prep_methodology)

## End(Not run)
```

`prep_methods` *Prepare methods*

Description

Collection of methods for preparing data conveniently named to make preparing easier

Usage

```
prep_methods()
```

Value

list of methods

See Also

Other JSON Prep: `clean_field_names()`, `get_entity()`, `prep_affiliation()`, `prep_array()`, `prep_array_objects()`, `prep_atomic()`, `prep_creators()`, `prep_data()`, `prep_descriptions()`, `prep_for_json()`, `prep_from_metadata_template()`, `prep_fundingReferences()`, `prep_identifier()`, `prep_language()`, `prep_methodology()`, `prep_nameIdentifiers()`, `prep_object()`, `prep_publicationYear()`, `prep_relatedIdentifiers()`, `prep_rights()`, `prep_subjects()`, `prep_titles()`

Examples

```
prep_methods()
```

`prep_nameIdentifiers` *Prepare Name identifiers*

Description

These are Persistent identifiers associated with a creator.

Usage

```
prep_nameIdentifiers(x)
```

Arguments

x	Data frame from "creators"

Details

Name identifiers in datacite is an array of objects with properties "nameIdentifier", "nameIdentifierScheme" , and "schemeUri". This function takes the `name identifiers` fields and restructures as a list within the data frame.

Value

data frame with a nameIdentifiers column as list

See Also

Other JSON Prep: `clean_field_names()`, `get_entity()`, `prep_affiliation()`, `prep_array()`, `prep_array_objects()`, `prep_atomic()`, `prep_creators()`, `prep_data()`, `prep_descriptions()`, `prep_for_json()`, `prep_from_metadata_template()`, `prep_fundingReferences()`, `prep_identifier()`, `prep_language()`, `prep_methodology()`, `prep_methods()`, `prep_object()`, `prep_publicationYear()`, `prep_relatedIdentifiers()`, `prep_rights()`, `prep_subjects()`, `prep_titles()`

Examples

```
creator_df <- wddsWizard::becker_project_metadata$creators[[1]]
creator_df_nameID_prepped <- prep_nameIdentifiers(creator_df)
```

prep_object *Prepare an object*

Description

Converts a named vector, list, or data frame to a list, and optionally unboxes it, so that its recorded as an object.

Usage

```
prep_object(x, unbox = FALSE)
```

Arguments

x	named vector, list, or data frame
unbox	logical Should items be unboxed (not arrays)? Default is FALSE meaning items will remain as arrays when converted to json.

Details

Note that unboxing will only work on items where you have 1:1 key value pair. So if you have a dataframe with multiple rows or a list with multiple values at a given position, it won't work.

Value

List of formatted objects

See Also

Other JSON Prep: clean_field_names(), get_entity(), prep_affiliation(), prep_array(), prep_array_objects(), prep_atomic(), prep_creators(), prep_data(), prep_descriptions(), prep_for_json(), prep_from_metadata_template(), prep_fundingReferences(), prep_identifier(), prep_language(), prep_methodology(), prep_methods(), prep_nameIdentifiers(), prep_publicationYear(), prep_relatedIdentifiers(), prep_rights(), prep_subjects(), prep_titles()

Examples

```
cars_small <- datasets::cars[1:10, ]

# creates an array of objects where each
# row is an object
cars_small |>
  jsonlite::toJSON(pretty = TRUE)

# creates an object with 2 arrays
prep_object(cars_small) |>
  jsonlite::toJSON(pretty = TRUE)
```

```
# this makes no difference
x <- list("hello" = 1:10, "world" = "Earth")

prep_object(x) |>
  jsonlite::toJSON(pretty = TRUE)
```

`prep_publicationYear` *Prepare publication year items*

Description

wrapper for prep atomic

Usage

```
prep_publicationYear(x)
```

Arguments

x	Named vector, data frame, or list

Value

an unboxed dataframe with 1 row

See Also

Other JSON Prep: `clean_field_names()`, `get_entity()`, `prep_affiliation()`, `prep_array()`, `prep_array_objects()`, `prep_atomic()`, `prep_creators()`, `prep_data()`, `prep_descriptions()`, `prep_for_json()`, `prep_from_metadata_template()`, `prep_fundingReferences()`, `prep_identifier()`, `prep_language()`, `prep_methodology()`, `prep_methods()`, `prep_nameIdentifiers()`, `prep_object()`, `prep_relatedIdentifiers()`, `prep_rights()`, `prep_subjects()`, `prep_titles()`

Examples

```
pub_year <- data.frame("publicationYear" = "2025")

prep_language(pub_year)
```

prep_relatedIdentifiers *Prepare related identifiers*

Description

Prepare related identifiers

Usage

```
prep_relatedIdentifiers(x)
```

Arguments

x	data frame with related identifier properties

Value

List with x marked as do not unbox

See Also

Other JSON Prep: clean_field_names(), get_entity(), prep_affiliation(), prep_array(), prep_array_objects(), prep_atomic(), prep_creators(), prep_data(), prep_descriptions(), prep_for_json(), prep_from_metadata_template(), prep_fundingReferences(), prep_identifier(), prep_language(), prep_methodology(), prep_methods(), prep_nameIdentifiers(), prep_object(), prep_publicationYear(), prep_rights(), prep_subjects(), prep_titles()

Examples

```
wddsWizard::becker_project_metadata$relatedIdentifiers |> prep_relatedIdentifiers()
```

prep_rights *Prepare rights*

Description

Prepares an array of objects

Usage

```
prep_rights(x)
```

Arguments

x	named list, vector, or data.frame of with 1:1 name:value pairs

Value

list of unboxed data frames

See Also

Other JSON Prep: clean_field_names(), get_entity(), prep_affiliation(), prep_array(), prep_array_objects(), prep_atomic(), prep_creators(), prep_data(), prep_descriptions(), prep_for_json(), prep_from_metadata_template(), prep_fundingReferences(), prep_identifier(), prep_language(), prep_methodology(), prep_methods(), prep_nameIdentifiers(), prep_object(), prep_publicationYear(), prep_relatedIdentifiers(), prep_subjects(), prep_titles()

Examples

```
wddsWizard::becker_project_metadata$rights |> prep_rights()
```

---

prep_subjects *Prepare subjects*

Description

Subjects or keywords describing a work. Prepares an array of objects.

Usage

```
prep_subjects(x)
```

Arguments

x	named list, vector, or data.frame of with 1:1 name:value pairs

Value

list of unboxed data frames

See Also

Other JSON Prep: clean_field_names(), get_entity(), prep_affiliation(), prep_array(), prep_array_objects(), prep_atomic(), prep_creators(), prep_data(), prep_descriptions(), prep_for_json(), prep_from_metadata_template(), prep_fundingReferences(), prep_identifier(), prep_language(), prep_methodology(), prep_methods(), prep_nameIdentifiers(), prep_object(), prep_publicationYear(), prep_relatedIdentifiers(), prep_rights(), prep_titles()

Examples

```
wddsWizard::becker_project_metadata$subjects |> prep_subjects()
```

prep_titles *Prepare Titles*

Description

Prepares an array of objects

Usage

```
prep_titles(x)
```

Arguments

x	list of data frames or a data frame

Value

list of single row unboxed data frames

See Also

Other JSON Prep: clean_field_names(), get_entity(), prep_affiliation(), prep_array(), prep_array_objects(), prep_atomic(), prep_creators(), prep_data(), prep_descriptions(), prep_for_json(), prep_from_metadata_template(), prep_fundingReferences(), prep_identifier(), prep_language(), prep_methodology(), prep_methods(), prep_nameIdentifiers(), prep_object(), prep_publicationYear(), prep_relatedIdentifiers(), prep_rights(), prep_subjects()

Examples

```
wddsWizard::becker_project_metadata$titles |>
  prep_titles()
```

project_metadata_required_fields *Required fields in the project metadata object*

Description

See data standard JSON file for field descriptions.

Usage

```
project_metadata_required_fields
```

Format

An object of class `character` of length 7.

See Also

Other Data: `becker_disease_data`, `becker_project_metadata`, `disease_data_required_fields`, `minimal_disease_data`, `minimal_project_metadata`, `schema_required_fields`, `spdx_licenses`, `wdds_to_dcmi_map`, `wdds_to_pharos_map`

`project_metadata_schema` *Wildlife Disease Data Standard - project_metadata*

Description

See data standard JSON file for field descriptions.

Usage

```
project_metadata_schema
```

Format

An object of class `list` of length 6.

See Also

Other Schema: `datacite_schema`, `disease_data_schema`, `schema_obj`, `schema_properties`, `schema_terms`, `wdds_schema`

`sanitize_version` *Sanitize version ids*

Description

This function replaces periods with under scores. The different versions of the data standard are stored in folders with their respective names; however, having periods in folder names can cause problems on certain operating systems and makes it more difficult to parse file extensions.

Usage

```
sanitize_version(version)
```

Arguments

`version`	Character. Version identifier.

Value

Character. Version identifier with no periods.

See Also

Other WDDS deposit: batch_download_deposit_versions(), download_deposit_version(), list_deposit_versions(), set_wdds_version(), wdds_data_templates(), wdds_example_data(), wdds_json()

Examples

```
sanitize_version("v.1.1.0")
```

---

schema_obj *Schema Object*

Description

A class for getting schema properties.

Value

List of of data frames. Create a list from a schema object

Creates a data.frame with the fields name and type

data frame with type and name Get schema references

Parses $ref calls in a schema. Can retrieve internal ('"$ref":"#/definitions/someDef") or external references ('"$ref":"schemas/datacite/datacite.json"').

For external references, it can handle both pointers and references to entire schemas. This function navigates between parent and child schemas by manipulating variables in the package environment `the`.

data frame with name or type. Process Array Items

Processes array items so they can be added to a data frame.

data frames with name and type for array items that are objects or character strings atomic (string, null, Boolean, etc) array items.

Public fields

`schema_path (character(1))`
: path to the schema file.

`schema_list_out (list())`
: List of data frames with schema properties.

`wdds_version (character(1))`
: version of wdds used

`current_schema_path` (`character(1)`)
: current schema file path

`current_schema_dir` (`character(1)`)
: current schema directory path

`current_sub_schema_dir` (`character(1)`)
: current sub schema directory path

`parent_schema_path` (`character(1)`)
: parent schema file path

`parent_schema_dir` (`character(1)`)
: parent schema directory

`array_items` (`c()`)
: array items

`array_items_skip` (`logical(1)`)
: array items to skip

`array_items_parent` (`logical(1)`)
: parent array items

Methods

Public methods:

- `schema_obj$new()`
- `schema_obj$create_schema_list()`
- `schema_obj$create_object_list()`
- `schema_obj$get_ref_list()`
- `schema_obj$process_array_items()`
- `schema_obj$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
schema_obj$new(schema_path, wdds_version = "latest")
```

Arguments:

`schema_path` Character. File path for the schema (`character(1)`)

`wdds_version` Character. Version of wdds used (`character(1)`)

Method `create_schema_list()`**:** Create an expanded schema object

Produces a list of data frame with name and type for the schema. This is a recursive set of function and may be expanded to get other properties.

Usage:

```
schema_obj$create_schema_list(schema_path = self$current_schema_path)
```

Arguments:

`schema_path` Character. Path to a json-schema. Default is the current schema path from the package environment,

Method `create_object_list()`:

Usage:

```
schema_obj$create_object_list(x, idx, schema_dir)
```

Arguments:

`x` List. Schema property or definition

`idx` Name from schema property

`schema_dir` Character. directory where the schema is stored

Method `get_ref_list()`:

Usage:

```
schema_obj$get_ref_list(x, schema_dir)
```

Arguments:

`x` List. Must have property "$ref"

`schema_dir` Character. Directory for the current schema.

Method `process_array_items()`:

Usage:

```
schema_obj$process_array_items(array_items, out)
```

Arguments:

`array_items` list. List of array items for processing.

`out` data frame. Data frame with name and type.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
schema_obj$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other Schema: `datacite_schema`, `disease_data_schema`, `project_metadata_schema`, `schema_properties`, `schema_terms`, `wdds_schema`

`schema_properties` *Wildlife Disease Data Standard - schema properties*

Description

A data frame of schema names and types.

Usage

```
schema_properties
```

Format

An object of class `data.frame` with 78 rows and 4 columns.

See Also

Other Schema: `datacite_schema`, `disease_data_schema`, `project_metadata_schema`, `schema_obj`, `schema_terms`, `wdds_schema`

`schema_required_fields` *Wildlife Disease Data Standard required fields*

Description

See data standard JSON file for field descriptions. This is a vector of the required fields for the entire schema.

Usage

```
schema_required_fields
```

Format

An object of class `character` of length 2.

See Also

Other Data: `becker_disease_data`, `becker_project_metadata`, `disease_data_required_fields`, `minimal_disease_data`, `minimal_project_metadata`, `project_metadata_required_fields`, `spdx_licenses`, `wdds_to_dcmi_map`, `wdds_to_pharos_map`

`schema_terms` *Wildlife Disease Data Standard - schema terms*

Description

Markdown of schema terms

Usage

```
schema_terms
```

Format

An object of class `character` of length 1.

See Also

Other Schema: datacite_schema, disease_data_schema, project_metadata_schema, schema_obj, schema_properties, wdds_schema

---

`set_wdds_version` *Set the wdds version for the package*

Description

Used to keep the package and data standard in alignment.

Usage

```
set_wdds_version(version = "latest")
```

Arguments

`version`	Character. identifier for a version e.g. "v.1.0.2" or "latest". Default is "latest".

Value

Character. Current schema version.

See Also

Other WDDS deposit: batch_download_deposit_versions(), download_deposit_version(), list_deposit_versions(), sanitize_version(), wdds_data_templates(), wdds_example_data(), wdds_json()

`spdx_licenses` *Data frame of SPDX licenses*

Description

A table with SPDX license metadata. Use spdx_licenses$licenseId when uploading data to Zenodo.

Usage

```
spdx_licenses
```

Format

An object of class `data.frame` with 706 rows and 9 columns.

Source

https://github.com/spdx/license-list-data/blob/main/json/licenses.json

See Also

Other Data: becker_disease_data, becker_project_metadata, disease_data_required_fields, minimal_disease_data, minimal_project_metadata, project_metadata_required_fields, schema_required_fields, wdds_to_dcmi_map, wdds_to_pharos_map

---

`translate_to_dcmi` *translate to dcmi*

Description

translate to dcmi

Usage

```
translate_to_dcmi(item, translation_map)
```

Arguments

item	List. Item to be translated.
translation_map	List. Instructions for translating the item

Value

List. Item that has been translated to DCMI

See Also

Other Standards Mapping: na_to_blank(), wdds_to_dcmi(), wdds_to_pharos()

`use_wdds_template` *Use a wildlife disease data standard template*

Description

This function allows you to easily copy and open a template from the package.

Usage

```
use_wdds_template(
  template_file = NULL,
  folder = fs::path_wd(),
  file_name = NULL,
  open = rlang::is_interactive(),
  overwrite = FALSE
)
```

Arguments

template_file	character. File name for a template. Defaults to NULL to return all template files.
folder	character. Where should the template be copied to? Default is the current working directory.
file_name	character. What should the copied file be called? Default is to use whatever value is supplied to template_file.
open	logical. Should the file be opened? Defaults to TRUE if interactive.
overwrite	logical. Should a file with the same name in the destination folder be overwritten? Default is FALSE to avoid accidentally overwriting data.

Value

Character. If no template_file value is provided, lists all template files in the package. If a file is created, it returns the file path for that new file.

See Also

Other Templates: list_wdds_templates()

Examples

```
# return available templates
use_wdds_template()

## Not run:

# makes a copy of the disease data template in the current working directory
use_wdds_template("disease_data_template.csv")

## End(Not run)
```

---

wdds_data_templates *Provides Access to Versioned Data Template Files*

Description

Since schema versions may change during the life cycle of project, it is important that users have access to all schema versions via this package. This function allows you to quickly retrieve whichever version of the data templates you may need.

Usage

```
wdds_data_templates(version = NULL, file = NULL)
```

Arguments

version	Character. Version of the wdds deposit. Leave as NULL to see all versions.
file	Character. Specific file from the wdds deposit. Leave as NULL to see all files in a version.

Details

This function does three things.

1. Shows all versions of the schema in the package if version is NULL
2. Provides paths to all example data files associated with a version of the schema if version is not NULL and file is NULL
3. Provides a specific file path in a specific version of the example data if both version and file are specified.

Value

Character. Either version identifiers or file paths.

See Also

Other WDDS deposit: batch_download_deposit_versions(), download_deposit_version(), list_deposit_versions(), sanitize_version(), set_wdds_version(), wdds_example_data(), wdds_json()

Examples

```
# see which versions are in the package

wdds_data_templates()

# see files associated with a version

wdds_data_templates(version = "latest")

# get the file path for a specific file

wdds_data_templates(version = "v_1_0_2", file = "disease_data_template.csv")
```

---

wdds_example_data *Provides Access to Versioned Example Data Files*

Description

Since schema versions may change during the life cycle of project, it is important that users have access to all schema versions via this package. This function allows you to quickly retrieve whichever version of the example data you may need.

Usage

```
wdds_example_data(version = NULL, file = NULL)
```

Arguments

version	Character or NULL. Version of the wdds deposit. Leave as NULL to see all versions. Default is NULL to return a character vector of versions.
file	Character or NULL. Specific file from the wdds deposit. Leave as NULL to see all files in a version. Default is NULL to return all files associated with a given version.

Details

This function does three things.

1. Shows all versions of the schema in the package if version is NULL.
2. Provides paths to all example data files associated with a version of the schema if version is provided and file is NULL.
3. Provides a specific file path in a specific version of the example data if both file and version are provided.

Value

Character. Either version identifiers or file paths.

See Also

Other WDDS deposit: batch_download_deposit_versions(), download_deposit_version(), list_deposit_versions(), sanitize_version(), set_wdds_version(), wdds_data_templates(), wdds_json()

Examples

```
# see which versions are in the package

wdds_example_data()

# see files associated with a version

wdds_example_data(version = "latest")

# get the file path for a specific file

wdds_example_data(version = "v_1_0_2", file = "Becker_demo_dataset.xlsx")
```

wdds_json *Provides Access to Versioned Schema Files*

Description

Since schema versions may change during the life cycle of project, it is important that users have access to all schema versions via this package. This function allows you to quickly retrieve whichever schema version you may need.

Usage

```
wdds_json(version = NULL, file = NULL)
```

Arguments

version	Character or NULL. Version of the wdds deposit. Leave as NULL to see all versions. Default is NULL to return character vector of versions.
file	Character or NULL. Specific file from the wdds deposit. Leave as NULL to see all files in a version. Default is NULL to return character vector of relative file paths.

Details

This function does three things:

1. Shows all versions of the schema in the package if both version and file are NULL.
2. Provides relative paths to all schema files associated with a version of the schema if only version is provided.
3. Provides a specific file path in a specific version of the schema if version and file path are provided.

Value

Character. Either version identifiers, relative file paths within a version, or a specific file path.

See Also

Other WDDS deposit: batch_download_deposit_versions(), download_deposit_version(), list_deposit_versions(), sanitize_version(), set_wdds_version(), wdds_data_templates(), wdds_example_data()

Examples

```
# see which versions are in the package

wdds_json()

# see files associated with a version

wdds_json(version = "latest")

# get the file path for a specific file

wdds_json(version = "v_1_0_2", file = "schemas/disease_data.json")
```

---

wdds_schema *Wildlife Disease Data Standard*

Description

See data standard JSON file for field descriptions.

Usage

```
wdds_schema
```

Format

An object of class `list` of length 6.

See Also

Other Schema: `datacite_schema`, `disease_data_schema`, `project_metadata_schema`, `schema_obj`, `schema_properties`, `schema_terms`

---

wdds_to_dcmi *WDDS to the Dublin Core Metadata Initiative*

Description

Converts WDDS project metadata to Zenodo flavored DCMI metadata.

Usage

```
wdds_to_dcmi(
  metadata_to_translate,
  translation_map = wddsWizard::wdds_to_dcmi_map
)
```

Arguments

`metadata_to_translate`
: List. Metadata that conforms to the WDDS data standard but is not prepped for JSON. See `prep_from_metadata_template`

`translation_map`
: List. A list that describes how to translate from WDDS to DCMI.

Value

List. Translated metadata with appropriate names.

See Also

Other Standards Mapping: `na_to_blank()`, `translate_to_dcmi()`, `wdds_to_pharos()`

Examples

```
project_metadata <- wdds_example_data(version = "latest",
                                      file = "example_project_metadata.csv")|>
    read.csv()

test_pmd <- project_metadata |>
  prep_from_metadata_template(json_prep = FALSE)

test_pmd$rights$rights <- "CC0-1.0"

dcmi_metadata <- wdds_to_dcmi(metadata_to_translate = test_pmd,
                              translation_map =  wddsWizard::wdds_to_dcmi_map)
```

---

wdds_to_dcmi_map *WDDS to DCMI metadata mapping*

Description

A list that maps variables between WDDS and the DCMI data standards.

Usage

```
wdds_to_dcmi_map
```

Format

An object of class `list` of length 2.

Source

https://github.com/ropenscilabs/deposits/blob/main/inst/extdata/dc/schema.json

See Also

Other Data: becker_disease_data, becker_project_metadata, disease_data_required_fields, minimal_disease_data, minimal_project_metadata, project_metadata_required_fields, schema_required_fields, spdx_licenses, wdds_to_pharos_map

---

wdds_to_pharos *Convert WDDS disease data to PHAROS data*

Description

As of 11 September 2025, WDDS and the PHAROS data model are not fully aligned. This function converts data that conforms to WDDS into the PHAROS data model. See wdds_to_pharos_map for the data model crosswalk.

Usage

```
wdds_to_pharos(wdds_disease_data)
```

Arguments

wdds_disease_data
: Data frame. A Disease Data set that conforms to the wdds data standard.

Value

Data frame. A tabular data set that conforms to the PHAROS data model.

See Also

Other Standards Mapping: na_to_blank(), translate_to_dcmi(), wdds_to_dcmi()

Examples

```
wdds_to_pharos(wdds_disease_data = wddsWizard::minimal_disease_data)

# data must be written to CSV then uploaded to PHAROS
```

wdds_to_pharos_map *WDDS to PHAROS metadata mapping*

Description

A table that maps variables between WDDS and the PHAROS data standard (11 September 2025). Will be deprecated once PHARSO and wdds are aligned.

Usage

```
wdds_to_pharos_map
```

Format

An object of class `spec_tbl_df` (inherits from `tbl_df`, `tbl`, `data.frame`) with 45 rows and 2 columns.

Source

https://pharos.viralemergence.org/

See Also

Other Data: becker_disease_data, becker_project_metadata, disease_data_required_fields, minimal_disease_data, minimal_project_metadata, project_metadata_required_fields, schema_required_fields, spdx_licenses, wdds_to_dcmi_map

Index