Package: iidda (via r-universe)

September 14, 2024

Type Package

Title Processing Infectious Disease Datasets in IIDDA.

Version 0.4.0

Maintainer Steve Walker <`swalk@mcmaster.ca`>

Description Part of an open toolchain for processing infectious
disease datasets available through the IIDDA data repository.

License GPL (>= 3)

Depends R (>= 3.5.0), dplyr, jsonlite, tibble, tidyr

Imports lubridate, memoise, tidyxl, ggplot2, tidyselect, methods,
utils, readr

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3.9000

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

RemoteUrl https://github.com/canmod/iidda-tools

RemoteRef HEAD

RemoteSha bcb8703ebfc05a79c51c691e3b9cca6ef97cd41e

Contents

`add_basal_disease` *Add Basal Disease*

Description

Add column ‘basal_disease‘ to tidy dataset

Usage

```
add_basal_disease(data, lookup)
```

Arguments

`data`	A tidy data set with a ‘disease‘ column
`lookup`	A lookup table with ‘disease‘ and ‘nesting_disease‘ columns that describe a global disease hierarchy that will be applied to find the basal disease of each ‘disease‘ in data

Value

tidy dataset with basal disease

`add_column_summaries` *Add Column Summaries*

Description

Add lists of unique values and ranges of values to a the metadata of an IIDDA data set.

Usage

```
add_column_summaries(tidy_data, dataset_name, metadata)
```

Arguments

`tidy_data`	Data frame of prepared data that are ready to be packaged as an IIDDA tidy data set.
`dataset_name`	Character string giving IIDDA identifier of the dataset.
`metadata`	Output of `get_tracking_metadata`.

`add_filter_group_values` *Add Filter Group Values*

Description

Add lists of unique sets of values for a given filter group

Usage

```
add_filter_group_values(tidy_data, dataset_name, metadata)
```

Arguments

`tidy_data`	Data frame of prepared data that are ready to be packaged as an IIDDA tidy data set.
`dataset_name`	Character string giving IIDDA identifier of the dataset.
`metadata`	Output of `get_tracking_metadata`.

`add_metadata` *Add Metadata*

Description

Add title and description metadata to a table and its columns.

Usage

```
add_metadata(table, table_metadata, column_metadata)
```

Arguments

`table`	dataframe (or dataframe-like object)
`table_metadata`	named list (or list-like object) such that `table_metadata$Title` and `table_metadata$Description` are strings containing the title and description of the table
`column_metadata`	dataframe with rownames equal to the columns in `table`, and `Title` and `Description` columns giving the title and description of each column in `table`

Value

version of `table` with added metadata `attributes`

`add_provenance` *Add Provenance*

Description

Add provenance information to an IIDDA dataset, by creating columns containing the scan and digitization IDs associated with each record.

Usage

```
add_provenance(tidy_data, tidy_dataset)
```

Arguments

`tidy_data`	Data frame in IIDDA tidy form.
`tidy_dataset`	The IIDDA identifier associated with the dataset for which ‘tidy_data‘ serves as an intermediate object during its creation.

`all_prep_script_outcomes` *Prep Script Outcomes*

Description

Prep Script Outcomes

Usage

```
all_prep_script_outcomes()

successful_prep_script_outcomes()

failed_prep_script_outcomes()

error_tar(tar_name)
```

Arguments

`tar_name`	Name of a tar archive to be created with log files of failed prep script outcomes.

Value

Data frame with all prep script outcomes in the project.

Functions

- `successful_prep_script_outcomes()`: Data frame with all successful prep script outcomes
- `failed_prep_script_outcomes()`: Data frame with all failed prep script outcomes
- `error_tar()`: Tar archive with log files of failed prep script outcomes.

`basal_disease` *Basal Disease*

Description

Basal Disease

Usage

```
basal_disease(disease, disease_lookup, encountered_diseases = character())
```

Arguments

`disease`	Disease for which to determine basal disease
`disease_lookup`	Table with two columns – disease and nesting_disease
`encountered_diseases`	Character vector of diseases already found. Typically this left at the default value of an empty character vector.

Value

The root disease that input disease maps to in disease_lookup.

`blob_to_raw` *Blob to Raw*

Description

Convert URL in GitHub blob storage format to GitHub raw data format.

Usage

```
blob_to_raw(urls)
```

Arguments

`urls`	Character vector of GitHub URLs in blob storage

Examples

```
blob_to_raw("https://github.com/canmod/iidda-tools/blob/main/R/iidda/R/github_parsing.R")
```

`check_metadata_cols` *Error if columns in the tidy data are not in metadata Schema and if all values in a column are NA*

Description

Error if columns in the tidy data are not in metadata Schema and if all values in a column are NA

Usage

```
check_metadata_cols(tidy_data, metadata)
```

Arguments

`tidy_data`	data.frame resulting from data prep scripts
`metadata`	Nested named list describing metadata for the tidy data

`check_tidy_data_cols` *Error if columns in the metadata Schema are not in tidy data*

Description

Error if columns in the metadata Schema are not in tidy data

Usage

```
check_tidy_data_cols(table, column_metadata)
```

Arguments

`table`	dataframe (or dataframe-like object)
`column_metadata`	dataframe with rownames equal to the columns in `table`, and `Title` and `Description` columns giving the title and description of each column in `table`

`collapse_xlsx_value_columns` *Collapse xlsx Value Columns*

Description

Collapse all value columns into a single `character` column for data frames that have one row per cell in an xlsx file.

Usage

```
collapse_xlsx_value_columns(data)
```

Arguments

`data`	Data frame representing an xlsx file.

`combine_weeks` *Combine Weeks*

Description

Combine data from different Excel sheets associated with specific weeks in 1956-2000 Canadian communicable disease incidence data prep pipelines.

Usage

```
combine_weeks(cleaned_sheets, sheet_dates, metadata)
```

Arguments

`cleaned_sheets`	List of data frames – one for each sheet
`sheet_dates`	Data frame describing sheet dates (TODO: more info needed)
`metadata`	Output of `get_tracking_metadata`.

`convert_harmonized_metadata` *Convert Harmonized Metadata*

Description

Get metadata for a harmonized data source, given metadata for the corresponding tidy data source metadata and initial harmonized data source metadata.

Usage

```
convert_harmonized_metadata(
  tidy_metadata,
  harmonized_metadata,
  tidy_source,
  harmonized_dataset_id,
  tidy_source_metadata_path
)
```

Arguments

`tidy_metadata`	Metadata from `read_tracking_tables` for a tidy data source.
`harmonized_metadata`	Initial metadata from `read_tracking_tables` for a harmonized data source.
`tidy_source`	IIDDA data source ID for a data source that is being harmonized.
`harmonized_dataset_id`	ID of dataset being harmonized.
`tidy_source_metadata_path`	Output of `convert_metadata_path`.

`convert_metadata_path` *Convert Metadata Path*

Description

Convert a metadata path to one corresponding to tidy data being harmonized.

Usage

```
convert_metadata_path(metadata_path, harmonized_source, tidy_source)
```

Arguments

`metadata_path`	Path to a collection of tracking tables.
`harmonized_source`	IIDDA data source ID for a harmonized source.
`tidy_source`	IIDDA data source ID for a data source that is being harmonized.

`cp_git_version` *Copy old git File Version*

Description

Create a temporary file containing a copy of a file under git version control for a particular revision of that file.

Usage

```
cp_git_version(file, version_hash)
```

Arguments

`file`	Path to file.
`version_hash`	Git version hash.

Value

Temporary file path containing the copy.

`csv_to_json_files` *CSV to JSON Files*

Description

Create a directory of JSON files from a CSV file.

Usage

```
csv_to_json_files(csv_path, json_dir, name_field, use_extension = FALSE)
```

Arguments

`csv_path`	Path to the CSV file.
`json_dir`	Path to the directory for saving the JSON files.
`name_field`	Name of the field in the CSV file that contains the names for each JSON file. All values in this field must be unique.
`use_extension`	If there is a column in the CSV file called ‘extension‘, should it be used to produced json filenames of the form ‘value-in-name-field.value-in-extension-field.json‘?

`data_to_json_files` *Data to JSON Files*

Description

Create a directory of JSON files from a data frame.

Usage

```
data_to_json_files(data, json_dir, name_field, use_extension = FALSE)
```

Arguments

`data`	Data frame
`json_dir`	Path to the directory for saving the JSON files.
`name_field`	Name of the field in the CSV file that contains the names for each JSON file. All values in this field must be unique.
`use_extension`	If there is a column in the CSV file called ‘extension‘, should it be used to produced json filenames of the form ‘value-in-name-field.value-in-extension-field.json‘?

`disease_coverage_heatmap` *Creates a heatmap that shows disease coverage over the years*

Description

Values are TRUE if that particular disease occurred at least once in a period that ended in that particular year, and FALSE otherwise.

Usage

```
disease_coverage_heatmap(table, disease_col = "disease")
```

Arguments

`table`	dataframe (or dataframe-like object). Tidy dataset of all compiled datasets
`disease_col`	specifies level of disease (i.e. disease_family, disease, disease_subclass)

`drop_empty_rows` *Drop Empty Rows*

Description

Drop empty rows in a table using `is_empty`.

Usage

```
drop_empty_rows(table)
```

Arguments

`table`	data frame

`empty_column_report` *Empty Column Report*

Description

Save the records of a dataset that contain empty values in ‘columns‘. This report will be saved in the ‘supporting-output/dataset_id‘ directory.

Usage

```
empty_column_report(data, columns, dataset_id)
```

Arguments

`data`	Data frame.
`columns`	Character vector of columns giving the columns to check for emptiness.
`dataset_id`	ID for the dataset that `data` will become, likely after further processing.

`empty_is_blank` *Empty is Blank*

Description

Force empty strings to be blank. See `is_empty`.

Usage

```
empty_is_blank(x)
```

Arguments

`x`	object to test

`empty_to_na` *Convert all missing values to NA*

Description

Convert all missing values to NA

Usage

```
empty_to_na(data)
```

Arguments

`data`	data frame resulting from data prep scripts

`epiweek_end_date` *End-Dates of Epiweeks*

Description

End-Dates of Epiweeks

Usage

```
epiweek_end_date(year, week)
```

Arguments

`year`	Integer vector of years.
`week`	Integer vector of weeks.

Value

Date vector of the end-dates of each specified epiweek.

`extract_between_paren` *Extract Substring Between Parentheses*

Description

Note that unless you specify an appropriate `contents_pattern` `extract_between_paren` will not work as you probably expect if there are multiple sets of parentheses. You can use exclusion patterns to make this work better (e.g. `content_pattern = '[^)]*'`).

Usage

```
extract_between_paren(
  x,
  left = "\\(",
  right = "\\)",
  contents_pattern = ".*"
)

extract_all_between_paren(
  x,
  left = "\\(",
  right = "\\)",
  contents_pattern = ".*",
  max_iters = 100
)
```

Arguments

`x`	Character vector
`left`	Left parenthetical string
`right`	Right parenthetical string
`contents_pattern`	Regex pattern for the contents between parentheses
`max_iters`	maximum number of items to return

Value

Character vector with NA's for elements in `x` that do not have parentheses and the substring between the first matching parentheses.

Examples

```
x = c("-", "", NA, "1", "3", "1 (Alta.)", "(Sask) 20")
extract_between_paren(x)
```

`extract_char_or_blank` *Extract Character or Blank*

Description

Extract a character vector from a list or return a blank string if it doesn't exist or if a proper list isn't passed.

Usage

```
extract_char_or_blank(l, e)
```

Arguments

`l`	List
`e`	Name of the focal element

`extract_or_blank` *Extract or Blank*

Description

Try to extract a list element, and return a blank list if it doesn't exist or if a proper list is not passed.

Usage

```
extract_or_blank(l, e)
```

Arguments

`l`	List
`e`	Name of the focal element

`fill_and_wrap` *Fill Template and Wrap the Results*

Description

Convenience function to do `fill_re_template` and `wrap_age_patterns` in one step.

Usage

```
fill_and_wrap(re_templates, which_bound, purpose, prefix = "")
```

Arguments

`re_templates`	a set of `re_templates` each passed to `fill_re_template`
`which_bound`	resolve the template to match lower or upper bounds, neither (the default), or single
`purpose`	character string indicating the purpose of the resulting regular expression
`prefix`	pattern to match at the beginning of the string that marks the beginning of age information

fill_re_template *Fill Regex Template*

Description

Resolve a length-1 character vector containing a regex template into a regular expression for matching age bound information in disease category names

Usage

```
fill_re_template(re_template, which_bound = "neither")
```

Arguments

`re_template`	template that resolve to regular expressions for matching age information contained in category names
`which_bound`	resolve the template to match lower or upper bounds, neither (the default), or single

fix_csv *Fix CSV*

Description

Fix the format of a CSV file that is not in IIDDA format.

Usage

```
fix_csv(filename)
```

Arguments

`filename`	Path to the CSV file

Value

Logical value that is ‘TRUE‘ if the CSV needed fixing and ‘FALSE‘ otherwise.

`freq_to_by` *Frequency to By*

Description

Convert words describing frequencies to phrases.

Usage

```
freq_to_by(freq)
```

Arguments

`freq`	one of ‘"weekly"‘ (becomes ‘"7 days"‘), ‘"4-weekly"‘ (becomes ‘"28 days"‘), ‘"monthly"‘ (becomes ‘"1 month"‘)

`freq_to_days` *Frequency to Days*

Description

Convert words describing frequencies to corresponding numbers of days

Usage

```
freq_to_days(freq)
```

Arguments

`freq`	one of ‘"weekly"‘ (becomes ‘7‘), ‘"4-weekly"‘ (becomes ‘28‘), ‘"monthly"‘ (returns an error)

`get_all_dependencies` *Get all Dependencies*

Description

Get all Dependencies

Usage

```
get_all_dependencies(source, dataset)
```

Arguments

`source`	Source ID.
`dataset`	dataset ID.

get_canmod_digitization_metadata *Get CANMOD Digitization Metadata*

Description

Superseded by functionality in ‘iidda.api‘.

Usage

```
get_canmod_digitization_metadata(tracking_list)
```

Arguments

`tracking_list`	output of `read_tracking_tables`

get_dataset_metadata *Get Dataset Metadata*

Description

Get an object with metadata information about a particular dataset from tracking tables.

Usage

```
get_dataset_metadata(dataset)
```

Arguments

`dataset`	Dataset identifier.

get_dataset_path *Get Dataset path*

Description

Get Dataset path

Usage

```
get_dataset_path(source, dataset, ext = "csv")
```

Arguments

`source`	Source ID.
`dataset`	dataset ID.
`ext`	Dataset file extension.

get_elements *Get Elements*

Description

Synonym for the `` `[` `` operator for use in pipelines.

Usage

```
get_elements()
```

get_firsts *Get Firsts*

Description

Get the first item in each sublist of sublists (ugh ... I know).

Usage

```
get_firsts(l, key)
```

Arguments

`l`	A list of lists of lists
`key`	Name of focal sublist (TODO: needs better description/motivation)

Examples

```
l = list(
  a = list(
    A = list(
      i = 1,
      ii = 2
    ),
    B = list(
      i = 3,
      ii = 4
    )
 ),
 b = list(
    A = list(
      i = 5,
      ii = 6
    ),
    B = list(
      i = 7,
      ii = 8
    )
  )
)
get_firsts(l, "A")
get_firsts(l, "B")
```

`get_items` *Get Items*

Description

Get list of items within each inner list of a list of lists

Usage

```
get_items(l, keys)
```

Arguments

`l`	A list of lists.
`keys`	Name of the items in the inner lists.

`get_lookup_table` *Get Lookup Table*

Description

Get Lookup Table

Usage

```
get_lookup_table(table_name = c("location_iso"))
```

Arguments

`table_name`	Name of a lookup table

`get_main_script` *Get Main Script*

Description

Get Main Script

Usage

```
get_main_script(source, dataset)
```

Arguments

`source`	Source ID.
`dataset`	dataset ID.

`get_source_path` *Get Source Path*

Description

Get Source Path

Usage

```
get_source_path(source)
```

Arguments

`source`	Source ID.

`get_tracking_metadata` *Read Tracking Metadata*

Description

Read in CSV files that contain the single-source-of-truth for metadata to be used in a data prep script.

Usage

```
get_tracking_metadata(
  tidy_dataset,
  digitization,
  tracking_path,
  original_format = TRUE,
  for_lbom = FALSE
)
```

Arguments

`tidy_dataset`	key to the tidy dataset being produced by the script
`digitization`	key to the digitization being used by the script
`tracking_path`	string giving path to the tracking data
`original_format`	should the original tracking table format be used?
`for_lbom`	are these data being read for the LBoM repo?

Details

This function currently assumes that a single tidy dataset is being produced from a single digitized file.

`get_unique_col_values` *Unique Column Values*

Description

Unique Column Values

Usage

```
get_unique_col_values(l)
```

Arguments

`l`	list of data frames with the same column names

Value

list of unique values in each column

`get_with_key` *Get with Key by Regex*

Description

Get with Key by Regex

Usage

```
get_with_key(l, key, pattern, ...)
```

Arguments

`l`	list of lists
`key`	name of item in inner list
`pattern`	regex pattern with which to match values of the key
`...`	additional arguments to pass to `grepl`

Value

subset of elements of `l` that match the pattern

`git_path_to_raw_github` *Convert GitHub URLs into Raw Format (not working)*

Description

Convert GitHub URLs into Raw Format (not working)

Usage

```
git_path_to_raw_github(urls, branch = "master")
```

Arguments

`urls`	TODO
`branch`	TODO

`group_with_dash` *Simplify String with List of Numbers Grouped by Dashes*

Description

Simplify String with List of Numbers Grouped by Dashes

Usage

```
group_with_dash(x)
```

Arguments

`x`	atomic vector

Value

length-1 character string giving a sorted list of numbers with contiguous numbers grouped by dashes.

Examples

```
group_with_dash(c("3840", "34", "2", "3", "1", "33", '5-50'))
group_with_dash(group_with_dash(c("3840", "34", "2", "3", "1", "33", '5-50')))
```

`harmonization_lookup_tables` *Harmonization Lookup Tables*

Description

List of lookup tables for harmonizing historical inconsistencies in naming.

Usage

```
harmonization_lookup_tables
```

Format

A list of data frames, one for each column with historical naming inconsistencies:

location **location** Unique names of locations found in IIDDA
- **iso_3166** National jurisdiction codes
- **iso_3166_2** Sub-national jurisdiction codes

sex **sex** Unique names of sexes found in IIDDA
- **iso_5218** Numeric sex codes

Details

For example, NFLD and Newfoundland can both be represented using the iso-3166-2 standard as CA-NL. These tables can be joined to data in IIDDA to produce standardized variables that harmonize historical inconsistencies.

`icd_finder` *ICD Finder*

Description

Return the Shortest ICD-10 Codes that Match a Regex Pattern. Requires an internet connection.

Usage

```
icd_finder(disease_pattern, maximum_number_results = 10L, ...)
```

Arguments

`disease_pattern`	Regex pattern describing a disease.
`maximum_number_results`	Integer giving the maximum number of ICD codes to return, with preference given to shorter codes.
`...`	Arguments to pass on to `grepl`. It is recommended to set `ignore.case` = `TRUE` and often `perl` = `TRUE`.

Examples

```
icd_finder("chick")  ## Struc by chicken!!
```

`identify_scales` *Identify Scales*

Description

Identifies time scales (wk, mo, qr, yr) and location types (province or country) within a tidy dataset.

Usage

```
identify_scales(
  data,
  location_type_fixer = canada_province_scale_finder,
  time_scale_identifier = identify_time_scales
)
```

Arguments

`data`	Data frame in IIDDA tidy format to add time scale and location scale information.
`location_type_fixer`	Function that takes a data frame in IIDDA tidy format and adds or fixes the ‘location_type‘ field.
`time_scale_identifier`	Function that takes a data frame in IIDDA tidy format and adds the ‘time_scale‘ field.

`iidda_data_dictionary` *IIDDA Data Dictionary*

Description

Get the global data dictionary for IIDDA

Usage

```
iidda_data_dictionary()
```

Details

This function requires an internet connection.

`iidda_from_single_file` *Create New IIDDA Dataset from Single File*

Description

Create New IIDDA Dataset from Single File

Usage

```
iidda_from_single_file(single_file, new_repo, lifecycle)
```

Arguments

`single_file`	path to single data file
`new_repo`	path to new IIDDA repository
`lifecycle`	character vector giving the lifecycle state (https://github.com/davidearn/iidda/blob/main/LIFECYCLE.md Probably ’Unreleased’, but it could in principle be ’Static’, ’Dynamic’, or ’Superseded’.

Value

No return value. Call to produce a new directory structure in a new IIDDA git repository containing a single source data file.

`in_git_repo` *In Git Repo*

Description

In Git Repo

Usage

```
in_git_repo()
```

`iso_3166_codes` *ISO-3166 and ISO-3166-2 Codes*

Description

Converts geographical location information, as it was described in a source document, to equivalent ISO-3166 and ISO-3166-2 codes.

Usage

```
iso_3166_codes(tidy_data, locations_iso)
```

Arguments

`tidy_data`	data frame containing a field called `location` containing geographical location information extracted from a source document
`locations_iso`	table containing three columns: `location` with all unique location identifiers in the `tidy_data`, `iso_3166` containing equivalent ISO-3166 codes (if applicable), and `iso_3166_2` containing equivalent ISO-3166-2 codes (if applicable)

`iso_8601_dateranges` *ISO-8601 Date Ranges*

Description

Converts start and end dates into ISO-8601-compliant date ranges.

Usage

```
iso_8601_dateranges(start_date, end_date)
```

Arguments

`start_date`	date vector
`end_date`	date vector

`iso_8601_dates` *ISO-8601 Dates*

Description

Convert date vectors into string vectors with ISO-8601 compliant format.

Usage

```
iso_8601_dates(dates)
```

Arguments

`dates`	date vector

`iso_codes` *Iso Codes*

Description

Superseded by `iso_3166_codes`.

Usage

```
iso_codes(tidy_data, locations_iso = read.csv("tracking/locations_ISO.csv"))
```

Arguments

`tidy_data`	data frame containing a field called `location` containing geographical location information extracted from a source document
`locations_iso`	table containing three columns: `location` with all unique location identifiers in the `tidy_data`, `iso_3166` containing equivalent ISO-3166 codes (if applicable), and `iso_3166_2` containing equivalent ISO-3166-2 codes (if applicable)

`is_empty` *Is Empty*

Description

Return `TRUE` if a string is empty. Emptiness means that any one of the following is true: `NA`, `NaN`, `nchar(as.character(x)) == 0L`, `tolower(as.character(x)) == "na"`

Usage

```
is_empty(x)
```

Arguments

`x`	object to test

`json_files_to_csv` *JSON Files to CSV*

Description

Create a CSV file from a set of JSON files.

Usage

```
json_files_to_csv(json_paths, csv_path)
```

Arguments

`json_paths`	Vector of paths to JSON files.
`csv_path`	Path for saving the resulting CSV file.

`json_files_to_data` *JSON Files to Data*

Description

Create a data frame from a set of JSON files.

Usage

```
json_files_to_data(json_paths)
```

Arguments

`json_paths`	Vector of paths to JSON files.

`key_val` *Key-Value*

Description

Create a set of key-value pairs by extracting elements from within a list of named-lists.

Usage

```
key_val(l, key, value)
```

Arguments

`l`	A list of named lists
`key`	A name of an element in each list in `l`
`value`	A name of an element in each list in `l`

Examples

```
f = system.file("example_data_dictionary.json", package = "iidda")
d = jsonlite::read_json(f)
key_val(d, "name", "type")
```

`list_dataset_ids` *List Dataset IDs*

Description

List Dataset IDs

Usage

```
list_dataset_ids(source)
```

Arguments

`source`	Source ID.

`list_dataset_ids_by_source` *List Dataset IDs by Source*

Description

List Dataset IDs by Source

Usage

```
list_dataset_ids_by_source()
```

`list_dependency_ids` *List Dependency IDs*

Description

List Dependency IDs

Usage

```
list_dependency_ids(
  source,
  dataset,
  type = c("PrepScripts", "Scans", "Digitizations", "AccessScripts")
)
```

Arguments

`source`	Source ID.
`dataset`	Dataset ID.
`type`	Type of resource.

`list_dependency_ids_for_source` *List Dependency IDs for Source*

Description

List Dependency IDs for Source

Usage

```
list_dependency_ids_for_source(
  source,
  type = c("PrepScripts", "Scans", "Digitizations", "AccessScripts")
)
```

Arguments

`source`	IIDDA source ID, which should correspond to metadata in ‘metadata/sources/souce.json‘ and a folder in ‘pipelines‘.
`type`	Type of dependency.

`list_dependency_paths` *List Dependency Paths*

Description

List Dependency Paths

Usage

```
list_dependency_paths(
  source,
  dataset,
  type = c("PrepScripts", "Scans", "Digitizations", "AccessScripts")
)
```

Arguments

`source`	Source ID.
`dataset`	dataset ID.
`type`	Type of resource.

`list_extract` *List Extract*

Description

Extract list items by regular expression matching on their names.

Usage

```
list_extract(x, pattern, ...)
```

Arguments

`x`	A list.
`pattern`	A regular expression
`...`	Arguments to pass to `grepl`

`list_file_id` *List File ID*

Description

List File ID

Usage

```
list_file_id(..., ext)
```

Arguments

`...`	Path components to directory containing the resources.
`ext`	Optional string giving the file extension of the resources. If missing then all resources are given.

Value

List of matching files without their extensions.

`list_prep_script_ids` *List Prep Script IDs*

Description

List Prep Script IDs

Usage

```
list_prep_script_ids(source)
```

Arguments

`source`	Source ID.

`list_resource_ids` *List Resources IDs*

Description

List Resources IDs

Usage

```
list_resource_ids(
  source,
 type = c("TidyDatasets", "PrepScripts", "Scans", "Digitizations", "AccessScripts")
)
```

Arguments

`source`	Source ID.
`type`	Type of resource.

`list_source_ids` *List Source IDs*

Description

List Source IDs

Usage

```
list_source_ids()
```

`list_xpath` *List XPath*

Description

Extract elements of lists using x-path-like syntax.

Usage

```
list_xpath(l, ...)
```

Arguments

`l`	A hierarchical list.
`...`	Character strings describing the path down the hierarchy.

Examples

```
l = list(
  a = list(
    A = list(
      i = 1,
      ii = 2
    ),
    B = list(
      i = 3,
      ii = 4
    )
 ),
 b = list(
    A = list(
      i = 5,
      ii = 6
    ),
    B = list(
      i = 7,
      ii = 8
    )
  )
)
list_xpath(l, "A", "i")
list_xpath(l, "B", "ii")
```

lookup *Lookup Value*

Description

Lookup Value

Usage

```
lookup(named_keys, l)
```

Arguments

named_keys	named character vector with values giving keys to lookup in l
l	list with names to match against the values of keys

make_age_hash_table *Make Age Hash Table*

Description

Create a lookup function that takes a character vector of disease category names and returns a vector of equal length containing either the lower or upper age bounds contained in the categories. If no bound is present then NA is returned.

Usage

```
make_age_hash_table(
  categories,
  re_templates,
  which_bound = c("lower", "upper", "neither", "single"),
  prefix = ""
)
```

Arguments

categories	character vector of disease category names
re_templates	list of templates that resolve to regular expressions for matching age information contained in category names
which_bound	resolve the template to match lower or upper bounds, neither (the default), or single
prefix	pattern to match at the beginning of the string that marks the beginning of age information

Value

vector containing either the lower or upper age bounds contained in the categories

`make_compilation_dependencies` *Make Compilation Dependencies*

Description

Create a dependency file and prep script for a dataset that is a compilation of other datasets. These files are created once and any edits should be made manually to the created files.

Usage

```
make_compilation_dependencies(compilation_dataset, dataset_paths)
```

Arguments

`compilation_dataset`	Dataset ID for which dependencies are being declared.
`dataset_paths`	Relative paths to dependencies.

`make_config` *Create IIDDA Config File*

Description

Create IIDDA Config File

Usage

```
make_config(
  path = file.path(getwd(), "config.json"),
  iidda_owner = "",
  iidda_repo = "",
  github_token = "",
  .overwrite = FALSE
)
```

Arguments

`path`	path for storing config file
`iidda_owner`	TODO
`iidda_repo`	TODO
`github_token`	TODO
`.overwrite`	should existing config.json files be overwritten

`make_dataset_dependencies` *Make Dataset Dependencies*

Description

Create a dependency file for a dataset. This file is created once and any edits should be made manually to the created file.

Usage

```
make_dataset_dependencies(tidy_dataset, paths)
```

Arguments

`tidy_dataset`	Dataset ID for which dependencies are being declared.
`paths`	Relative paths to dependencies.

`make_dataset_metadata` *Make Dataset Metadata*

Description

Make Dataset Metadata

Usage

```
make_dataset_metadata(tidy_dataset, type, ...)
```

Arguments

`tidy_dataset`	Dataset ID for which metadata is being produced.
`type`	Type of dataset (e.g., CDI, Mortality).
`...`	Additional metadata fields to provide. If invalid fields are supplied, an error message will be given.

`make_data_cite_tidy_data` *Make DataCite JSON Metadata*

Description

Make DataCite JSON Metadata

Usage

```
make_data_cite_tidy_data(metadata, file)
```

Arguments

`metadata`	Output of get_tracking_metadata
`file`	Path to metadata file

`make_resource_metadata` *Make Resource Metadata*

Description

Make one json metadata file for each resource (i.e., prep/access script or digitization/scan of data)) in a source pipeline associated with a data source (i.e., a sub-directory of ‘pipelines‘). Existing metadata files will not be overwritten.

Usage

```
make_resource_metadata(source)
```

Arguments

`source`	Source ID.

`make_source_directory` *Make Source Directory*

Description

Make a sub-directory of ‘pipelines‘ containing a data and/or code source.

Usage

```
make_source_directory(source, files)
```

Arguments

`source`	Source ID.
`files`	Character vector of files that are either already in the pipeline or that should be added.

`make_source_metadata` *Make Source Metadata*

Description

Make a json file associated with a new data source (i.e., a sub-directory of ‘pipelines‘).

Usage

```
make_source_metadata(source, organization, location, ...)
```

Arguments

`source`	Source ID.
`organization`	Organization from which the source was obtained.
`location`	Location for which data was collected.
`...`	Additional metadata fields to provide. If invalid fields are supplied, an error message will be given.

melt_tracking_table_keys — *Melt Tracking Table Keys (Deprecated)*

Description

To be used in conjunction with tracking_table_keys.

Usage

```
melt_tracking_table_keys(keys)
```

Arguments

keys	Character vector of

MissingHandlers — *Missing Handlers*

Description

Construct an object with functions for handling missing values.

Usage

```
MissingHandlers(
  unclear = c("Unclear", "unclear", "uncleaar", "uncelar", "r"),
  not_reported = c("", "Not available", "*", "Not reportable", "missing"),
  zeros = "-"
)
```

Arguments

unclear	Character vector giving values corresponding to numbers that were unclear to data enterers.
not_reported	Character vector giving values corresponding to numbers that were not reported in the original source.
zeros	Character vector giving values corresponding to ‘0‘ but that were entered as another character to resemble the original source.

Value

An environment with functions for handling missing values.

`mock_api_hook` *Mock API Hook*

Description

Mock API Hook

Usage

```
mock_api_hook(repo_path)
```

Arguments

`repo_path`	Path to an IIDDA repository.

`nlist` *Self-Naming List*

Description

Copied from `lme4:::namedList`.

Usage

```
nlist(...)
```

Arguments

`...`	a list of objects

`non_numeric_report` *Non-Numeric Report*

Description

Save the records of a dataset that contain non-numeric data within a specified numeric field. This report will be saved in the ‘supporting-output/dataset_id‘ directory.

Usage

```
non_numeric_report(data, numeric_column, dataset_id)
```

Arguments

`data`	Data frame.
`numeric_column`	Name of a numeric column in `data`.
`dataset_id`	ID for the dataset that `data` will become, likely after further processing.

`normalize_diseases` *Normalize Diseases*

Description

Normalize the names of diseases to simplify the harmonization of disease names across historical sources.

Usage

```
normalize_diseases(diseases)
```

Arguments

`diseases`	Character vector of disease names

`open_locally` *Open a Path on Mac OS or Windows*

Description

Open a Path on Mac OS or Windows

Usage

```
open_locally(urls, command = "open", args = character())

open_resources_locally(
  id,
  type = c("scans", "digitizations", "prep-scripts", "access-scripts")
)

open_all_resources_locally(id)

open_scans_locally(id)

open_digitizations_locally(id)
```

Arguments

`urls`	Character vector of GitHub URLs in blob storage
`command`	Command-line function to use to open the file (not applicable on Windows systems.
`args`	Additional options to pass to `command` (ignored on Windows systems).
`id`	Resource ID.
`type`	Type of resource.

Functions

- `open_resources_locally()`: Open IIDDA pipeline resources locally.
- `open_all_resources_locally()`: Open all pipeline resources regardless of resource type.
- `open_scans_locally()`: Open scans locally.
- `open_digitizations_locally()`: Open digitizations locally.

`or_pattern` *Or Pattern*

Description

Construct regex for Boolean-or.

Usage

```
or_pattern(x, at_start = TRUE, at_end = TRUE)
```

Arguments

`x`	Character vector of alternative patterns.
`at_start`	Match only at the start of strings.
`at_end`	Match only at the end of strings.

`pager` *Pager*

Description

Pager

Usage

```
pager(page, n_per_page, rev = TRUE)
```

Arguments

`page`	What page should be returned?
`n_per_page`	How many entries on each page?
`rev`	Should page one be at the end?

Value

Function of ‘x‘ to return the ‘page‘th ‘page‘ of size ‘n_per_page‘ of ‘x‘.

`paste_operators` *Paste Operators*

Description

Syntactic sugar for common string pasting operations.

Usage

```
x %_% y

x %+% y

x %.% y

x %-% y
```

Arguments

`x`	character vector
`y`	character vector

Details

`%+%` Paste with a blank separator, like python string concatenation

`%_%` Paste with underscore separator

`%.%` Paste with dot separator – useful for adding file extensions

`%-%` Paste with dash separator – useful for representing contiguous numbers

Value

`x` concatenated with `y`

Examples

```
'google' %.% 'com'
'snake' %_% 'case'
```

pipeline_exploration_starter *Pipeline Exploration Quick-Start*

Description

Create an R script providing a place to start when exploring an IIDDA pipeline.

Usage

```
pipeline_exploration_starter(script_filename, exploration_project_path, ...)
```

Arguments

`script_filename`	Name for the generated script.
`exploration_project_path`	Path to the folder for containing the script. If this path doesn't exist, then it is created. If `script_filename` exists in `exploration_project_path`, an error is returned.
`...`	Additional arguments to pass to `file.copy`. A useful argument here is 'overwrite', which indicates whether an existing exploration script should be overwritten.

Details

The R script has the following:

1. Example code for printing out the data sources and datasets in the IIDDA pipeline repository. 2. Code for finding the paths to datasets and to the scripts for generating them. 3. Code for generating and/or reading in a user-selected IIDDA dataset.

Once the data are read in, the user is free to do whatever they want to with it.

proj_path *Project Path*

Description

Return a path in absolute form (if that is how it is specified) or relative to the IIDDA project root found using `proj_root`.

Usage

```
proj_path(...)
```

Arguments

`...`	Path components for `file.path`.

`proj_root` *Project Root*

Description

Find the root path of an IIDDA-associated project (or any project with a file of a specific name in the root).

Usage

```
proj_root(filename = ".iidda", start_dir = getwd(), default_root = start_dir)

in_proj(filename = ".iidda", start_dir = getwd())
```

Arguments

`filename`	String giving the name of the file that identifies the project.
`start_dir`	Optional directory from which to start looking for ‘filename‘.
`default_root`	Project root to use if ‘filename‘ is not found.

Details

Recursively walk up the file tree from ‘start_dir‘ until ‘filename‘ is found, and return the path to the directory containing ‘filename‘. If ‘filename‘ is not found, return ‘default_root‘

Functions

- `in_proj()`: Is a particular directory inside a project as indicated by ‘filename‘.

`raw_github` *Construct an URL to Download Single Files from GitHub*

Description

Uses the Raw GitHub API

Usage

```
raw_github(owner, repo, path, user = NULL, token = NULL, branch = "master")
```

Arguments

`owner`	User or Organization of the repo
`repo`	Repository name
`path`	Path to the file that you want to obtain
`user`	Your username (only required for private repos)
`token`	OAuth personal access token (only required for private repos)
`branch`	Name of the branch (defaults to 'master')

`readme_classic_iidda` *README File Template*

Description

(Deprecated)

Usage

```
readme_classic_iidda
```

Format

An object of class `character` of length 1.

`read_column_metadata` *Read Column Metadata*

Description

Read Column Metadata

Usage

```
read_column_metadata(dataset, pattern)
```

Arguments

`dataset`	IIDDA dataset ID.
`pattern`	Regular expression pattern for filtering candidate paths to be read from.

`read_data_columns` *Read Data Columns*

Description

Read Data Columns

Usage

```
read_data_columns(filename)
```

Arguments

`filename`	Path to a CSV file in IIDDA format.

`read_data_frame` *Read Data Frame*

Description

Read in a data frame from a CSV file using the CSV dialect adopted by IIDDA.

Usage

```
read_data_frame(filename, col_classes = "character")
```

Arguments

`filename`	String giving the filename.
`col_classes`	See `colClasses` from `read.table`.

`read_digitized_data` *Read Digitized Data*

Description

Read in digitized data to be prepared within the IIDDA project.

Usage

```
read_digitized_data(metadata)
```

Arguments

`metadata`	Output of `get_tracking_metadata`.

`read_global_metadata` *Read Global Metadata*

Description

Read Global Metadata

Usage

```
read_global_metadata(
  id,
  type = c("columns", "organization", "sources", "tidy-datasets")
)
```

Arguments

`id`	ID to the ‘type‘ of entity.
`type`	Type of entity.

`read_lookup` *Read Lookup*

Description

Read Lookup

Usage

```
read_lookup(lookup_id)
```

Arguments

`lookup_id`	IIDDA ID associated with an item in a ‘lookup-tables‘ directory in an IIDDA repository.

`read_prerequisite_data` *Read Prerequisite Data*

Description

Read Prerequisite Data

Usage

```
read_prerequisite_data(dataset_id, numeric_column_for_report = NULL)
```

Arguments

`dataset_id`	IIDDA dataset ID.
`numeric_column_for_report`	Optional numeric column name to specify for producing a report using `non_numeric_report`.

`read_prerequisite_metadata` *Read Prerequisite Metadata*

Description

Read Prerequisite Metadata

Usage

```
read_prerequisite_metadata(dataset, pattern)
```

Arguments

`dataset`	IIDDA dataset ID.
`pattern`	Regular expression pattern for filtering candidate paths to metadata.

read_prerequisite_paths

Read Prerequisite Paths

Description

Read Prerequisite Paths

Usage

```
read_prerequisite_paths(dataset, pattern)
```

Arguments

`dataset`	IIDDA dataset ID.
`pattern`	Regular expression pattern for filtering candidate paths to be read from.

read_resource_metadata

Read Resource Metadata

Description

Read Resource Metadata

Usage

```
read_resource_metadata(dataset, pattern)
```

Arguments

`dataset`	IIDDA dataset ID.
`pattern`	Regular expression pattern for filtering candidate paths to be read from.

`read_tidy_data` *Read Tidy Data and Metadata files*

Description

Read Tidy Data and Metadata files

Usage

```
read_tidy_data(tidy_data_path, just_csv = FALSE)
```

Arguments

`tidy_data_path`	path to folder containing 4 files: tidy data and resulting metadata for each prep script
`just_csv`	return only the tidy csv file or a list with the csv and its metadata

`read_tracking_tables` *Read Tracking Tables*

Description

Read metadata tracking tables for an IIDDA project.

Usage

```
read_tracking_tables(path)
```

Arguments

`path`	Path containing tracking tables.

`register_prep_script` *Register Prep Script*

Description

Convenience function for a one-time setup of all metadata required for a new prep script. The assumptions are that (1) the prep script is a '.R' file in the 'prep-scripts' directory of a directory within the 'pipelines' directory and (2) that this script produces a csv file in the 'derived-datasets' directory with the same 'basename()' as this '.R' file. Messages are printed with paths to newly created and/or existing metadata, derived data, and dependency files that should be checked manually. Sometimes it is helpful to delete some of these files and rerun 'register_prep_script'. However, this 'register_prep_script' function should not be used in a script that is intended to be run multiple times, as going forward the metadata and dependency files should be edited manually.

Usage

```
register_prep_script(script_path, type)
```

Arguments

`script_path`	Path to the prep-script being registered.
`type`	Type of the dataset being produced (e.g., CDI, Mortality). TODO: Give a list of acceptable values. Should be programmatically produced.

`relative_paths` *Relative Paths*

Description

Convert a set of absolute paths to relative paths with respect to a specified 'containing_path'

Usage

```
relative_paths(paths, containing_path = proj_root())
```

Arguments

`paths`	Vector of absolute paths.
`containing_path`	Target working directory to be relative to.

`remote_iidda_git` *Remote IIDDA Git*

Description

Remote IIDDA Git

Usage

```
remote_iidda_git()
```

`remove_age` *Remove Age*

Description

Remove age information from a vector of category names

Usage

```
remove_age(categories, re_templates, prefix = "")

memoise_remove_age(categories, re_templates, prefix = "")
```

Arguments

`categories`	vector of category names
`re_templates`	list of templates that resolve to regular expressions for matching age information contained in category names
`prefix`	pattern to match at the beginning of the string that marks the beginning of age information

remove_between_paren *Remove Parenthesized Substring*

Description

Remove Parenthesized Substring

Usage

```
remove_between_paren(
  x,
  left = "\\(",
  right = "\\)",
  contents_pattern = ".*"
)
```

Arguments

x	Character vector
left	Left parenthetical string
right	Right parenthetical string
contents_pattern	Regex pattern for the contents between parentheses

Value

Version of x with first parenthesized substrings removed

Examples

```
x = c("-", "", NA, "1", "3", "1 (Alta.)", "(Sask) 20")
remove_between_paren(x)
```

return_matched_age_bound *Matched Age Bound*

Description

Process output from regmatches to return the correct age bound. Used in the lookup function created by make_age_hash_table

Usage

```
return_matched_age_bound(x)
```

Arguments

`x`	character vector from the list output of regmatches, containing regex matches of age bound information contained in disease category names. each `x` corresponds to a single category.

Value

Character string with matched age bound

`rm_trailing_slash` *Remove Trailing / Leading Slash*

Description

Remove Trailing / Leading Slash

Usage

```
rm_trailing_slash(x)

rm_leading_slash(x)
```

Arguments

`x`	Character vector with paths.

Value

Character vector without trailing/leading slash.

`save_result` *Save Results of a Data Prep Script*

Description

Save the resulting objects of a data prep script into an R data file. The names of the resulting objects are given by the names of the result list.

Usage

```
save_result(result, metadata)
```

Arguments

`result`	Named list of data resulting from data prep scripts
`metadata`	Nested named list describing metadata for the result. It must have a `$Product[["Path to tidy data"]]` component, which is a GitHub URL describing the ultimate location of the R data file. The GitHub component of the URL will be removed to produce a path that will correspond to the location within a cloned git repository – note that the path is relative to the top-level of the cloned repository.

`set_ext` *Set Extension*

Description

Set Extension

Usage

```
set_ext(paths, ext)
```

Arguments

`paths`	Character vector giving file paths.
`ext`	String giving the file extension to add to the `paths`.

`set_iidda_col_types` *Set IIDDA Column Types*

Description

Deprecated – iidda.api package is not more robust.

Usage

```
set_iidda_col_types(data)
```

Arguments

`data`	Dataset from IIDDA Api

set_types *Set Data Frame Column Types*

Description

Set the types of the columns of a data frame.

Usage

```
set_types(data, types)
```

Arguments

`data`	data frame
`types`	dict-like list with keys giving column names and values giving types

Value

data frame with changed column types – note that the returned data frame is a plain base R `data.frame` (i.e. not a `tibble` or `data.table`).

source_from_digitization_id *Source from Digitization ID*

Description

Source from Digitization ID

Usage

```
source_from_digitization_id(digitization_ids)
```

Arguments

`digitization_ids`	Character vector of digitization IDs

Value

Character vector of source IDs associated with digitization.

`sprintf_named` *Lightweight Templating*

Description

Version of the `sprintf` base R function that adds basic templating – https://stackoverflow.com/a/55423080/2047693.

Usage

```
sprintf_named(template, ..., .check = TRUE)
```

Arguments

`template`	template
`...`	Named arguments with strings that fill template variables of the same name between %{ and }s
`.check`	Should the consistency between the arguments and the template be checked?

Details

Because this is based on the sprintf function, use `%%` when you would like a single `%` to appear in the template. However, when supplying a single `%` to a named argument will result in a single `%` in the output.

You can use syntactically invalid names for arguments by enclosing them in backticks in the argument list, but not in the template.

Examples

```
sprintf_named("You might like to download datasets from %{repo}s.", repo = "IIDDA")
```

`standards` *Standards*

Description

List of lists of lists that exploits tab completion to make it convenient to get vectors of all synonyms associated with a particular standard code. This mechanism is useful when searching for data in IIDDA.

Usage

```
standards
```

Format

List of lists of character vectors containing the original historical names:

location **iso_3166** Historical national names associated with each iso-3166 code.

- **iso_3166_inclusive** Historical national and sub-national names associated with each iso-3166 code.
- **iso_3166_2** Historical sub-national names associated with each iso-3166-2 code.

sex **iso_5218** Historical names referring to sexes associated with each iso-5218 code.

`statcan_mort_prep` *Prepare Mortality Data from Statistics Canada*

Description

Prepare Mortality Data from Statistics Canada

Usage

```
statcan_mort_prep(data)
```

Arguments

`data`	Output of `read_digitized_data` that has been filtered to include only the cell range that contains data.

Value

Data frame complying with the IIDDA requirements for tidy datasets.

`strip_blob_github` *Strip Blob*

Description

Strip the ’blob part’ of a GitHub URL so that it is a path relative to a local clone of the associated repo.

Usage

```
strip_blob_github(urls)
```

Arguments

`urls`	Character vector of GitHub URLs in blob storage

Examples

```
strip_blob_github("https://github.com/canmod/iidda-tools/blob/main/R/iidda/R/github_parsing.R")
```

`summarise_dates` *Summarise Dates*

Description

Consecutive or overlapping date ranges are summarised into a single date range, non-consecutive date ranges are kept as is.

Usage

```
summarise_dates(x_start, x_end, range_operator = " to ", collapse = TRUE)
```

Arguments

`x_start`	vector of period starting dates.
`x_end`	vector of period ending dates.
`range_operator`	string to go between the start and end date, defaults to " to ".
`collapse`	boolean to collapse all dates into one comma separated string, defaults to TRUE.

Value

vector or single string of summarised date ranges.

`summarise_diseases` *Summarise Diseases*

Description

Summarise disease name columns in an IIDDA dataset.

Usage

```
summarise_diseases(data)
```

Arguments

`data`	Data frame hopefully containing at least one of ‘disease‘ or ‘historical_disease‘. If all are missing then the output summary is a blank string.

Value

A string summarizing the data in the columns.

`summarise_integers` *Summarise Integers*

Description

Consecutive or overlapping integers separated by commas or semi-colons are summarised into a single integer range, non-consecutive integer ranges are kept as is.

Usage

```
summarise_integers(x, range_operator = "-", collapse = TRUE)
```

Arguments

`x`	vector of integers
`range_operator`	string to go between the starting and ending integer in the range, defaults to "-".
`collapse`	boolean to collapse all integer ranges into one comma separated string, defaults to TRUE.

Value

vector or single string of summarised integer ranges.

`summarise_locations` *Summarise Locations*

Description

Summarise several columns in an IIDDA dataset that specify the geographic location of each row.

Usage

```
summarise_locations(data)
```

Arguments

`data`	Data frame hopefully containing at least one of ‘iso_3166‘, ‘iso_3166_2‘, or ‘location‘. If all are missing then the output summary is a blank string.

Value

A string summarizing the data in the columns.

summarise_periods *Summarise Periods*

Description

Summarise time periods in an IIDDA dataset.

Usage

```
summarise_periods(data, cutoff = 50)
```

Arguments

`data`	Data frame hopefully containing both ‘period_start_date‘ and ‘period_end_date‘. If either are missing an error results.
`cutoff`	Number of characters, above which the output string takes the form ‘max-date to min-date (with gaps)‘.

Value

A string summarizing the data in the columns

summarise_strings *Summarise Strings*

Description

Summarise vector of strings separated by commas or semi-colons into a single character separated string. Removes empty strings, repeated strings and trims white space.

Usage

```
summarise_strings(x, sep = ", ")
```

Arguments

`x`	vector
`sep`	character separator, defaults to ", "

Value

single string of summarised strings.

`test_result` *Test Results*

Description

Test the results of a data prep script (not finished).

Usage

```
test_result(result)
```

Arguments

`result`	Named list of data resulting from data prep scripts

`time_series_islands` *Time Series Islands*

Description

Find 'island rows' in a dataset with ordered rows. Islands have a series variable that is not ‘NA‘ surrounded by ‘NA‘ values in that same variable.

Usage

```
time_series_islands(data, series_variable, time_variable = NULL)
```

Arguments

`data`	A dataset (must be ordered if ‘time_variable‘ is ‘NULL‘).
`series_variable`	Name of a series variable.
`time_variable`	Optional variable to use for ordering the dataset before islands are located.

`tracking_tables_with_column` *Which Tracking Tables have a Particular Column*

Description

Which Tracking Tables have a Particular Column

Usage

```
tracking_tables_with_column(metadata, col_nm)
```

Arguments

`metadata`	Output of `read_tracking_tables`.
`col_nm`	Name of a column.

`tracking_table_keys` *Tracking Table Keys*

Description

Tracking Table Keys

Usage

```
tracking_table_keys
```

Format

An object of class `list` of length 5.

`two_field_format` *Two Field Format*

Description

Attempt to automatically convert a dataset from ‘disease|_subclass|_family‘ format of disease ID to the ‘|nesting_disease‘ format.

Usage

```
two_field_format(dataset)
```

Arguments

`dataset`	A tidy data set with ‘disease|_subclass|_family‘ columns.

`unlist_char_list` *Unlist a List of Character Vectors*

Description

Replacing list elements with `list('')` for each element that is null, not a character vector, or length zero.

Usage

```
unlist_char_list(x)
```

Arguments

`x`	list of character vectors

`vsub` *Vectorized String Substitution*

Description

Vectorized String Substitution

Usage

```
vsub(pattern, replacement, x, ...)
```

Arguments

`pattern`, `replacement`, `x`	first three arguments to `sub`, but the first is allowed to be a vector
`...`	additional arguments to pass on to `sub`

`wrap_age_patterns` *Wrap Age Patterns*

Description

Wrap list of regular expressions for matching age bounds in disease category names, so that the resulting regular expressions can be used for different purposes (extraction, removal, or validation)

Usage

```
wrap_age_patterns(
  patterns,
  purpose = c("extraction", "removal", "validate"),
  prefix = ""
)
```

Arguments

`patterns`	vector of regular expressions for matching age bound information in disease category names
`purpose`	character string indicating the purpose of the resulting regular expression
`prefix`	pattern to match at the beginning of the string that marks the beginning of age information

`write_data_frame` *Write Data Frame*

Description

Write a data frame to a CSV file using the CSV dialect adopted by IIDDA.

Usage

```
write_data_frame(data, filename)
```

Arguments

`data`	A data frame to write
`filename`	string giving the filename

`write_local_data_dictionaries` *Write Local Data Dictionaries*

Description

Write Local Data Dictionaries

Usage

```
write_local_data_dictionaries(metadata, path)
```

Arguments

`metadata`	Output of `read_tracking_tables`.
`path`	Path to a new JSON file.

`write_tidy_data` *Write Tidy Digitized Data and Metadata*

Description

Write Tidy Digitized Data and Metadata

Usage

```
write_tidy_data(tidy_data, metadata, tidy_dir = NULL)
```

Arguments

`tidy_data`	Data frame of prepared data that are ready to be packaged as an IIDDA tidy data set.
`metadata`	Output of `get_tracking_metadata`.
`tidy_dir`	If `NULL` taken from the `metadata`.

Value

file names where data were written

`xlsx_diff` *Compare Two Excel Files*

Description

Report on the differences between two xlsx files.

Usage

```
xlsx_diff(path_one, path_two, ...)
```

Arguments

`path_one`	Path to an Excel file.
`path_two`	Path to an Excel file.
`...`	Additional arguments to pass to `xlsx_cells`.

Value

Either ‘TRUE‘ if the two files are identical, or a list with the following items. * ‘all_equal‘ : Result of applying `all.equal` to the data frames representing each Excel file. * ‘in_both_but_different‘ : Data frame containing cells that are in both Excel files but with different values. * ‘in_one_only‘ : Data frame containing cells that are in the first Excel file but not the second. * ‘in_two_only‘ : Data frame containing cells that are in the second Excel file but not the first.

`xlsx_to_csv` *Excel to CSV*

Description

Convert an Excel file to a CSV file.

Usage

```
xlsx_to_csv(xlsx_path, csv_path)
```

Arguments

`xlsx_path`	Path to an Excel file.
`csv_path`	Path to a new CSV file.

Index